Measuring the efficiency and effectiveness of WineFS

Harshit Gupta
Texas A&M University

Rajendra Sahu
Texas A&M University

1. Project Objectives

The objective of our project is to recreate and evaluate WineFS by testing it using various workloads aged using multiple aging profiles. WineFS is a Persistent Memory (PM) file system that preserves huge pages even when the file system has been aged significantly.

The conventional file systems treat storage as block-level devices whereas PM is a byte addressable memory. In addition to that, the software overhead for conventional file systems turns out to be higher than the access latency itself in the case of PM. This is why there has been a lot of research on new file system designs for PM that can extract the maximum performance out of them.

PM File Systems usually employ the Memory Mapped path since it cuts down the system call overhead and accesses the PM using processor load and store instructions. TLB thrashing is a major cause of performance degradation in the translation of memory mapped instructions. Using hugepages is beneficial in memory mapped PM accesses as it reduces the number of TLB and page table entries. Aging fragments the disk and reduces the free space extents available for hugepage allocation. WineFS targeted this particular scenario and proposed a robust solution to tackle this problem of aging.

For this project, we are re-evaluating WineFS by recreating it on our hardware, a scaled down version of the hardware used in original experiments, to check if the results are consistent across hardware and workloads. We are using RAM based emulated PM as opposed to Intel Optane DC Persistent Memory that was used in the original evaluation for WineFS in its original paper. Since it is a RAM based emulated PM, we had to scale down the workloads to fit the capacity of our system's RAM.

We intend to perform and contrast memory mapped performance tests on unaged and aged WineFS, NOVA and Ext4 file systems. The WineFS paper uses RocksDB and LMDB memory mapped applications to run YCSB and fillseqbatch workloads respectively on unaged and aged file systems. They have used the Agrawal aging scheme to evaluate all the results. In our project, we re-evaluate these workloads on the same memory mapped applications and also evaluate the fillseq, fillrandom, overwrite, fillseqsync, fillrandsync, fillrandbatch, readseq, readreverse and readrandom on Agrawal, Meyer and a custom aging scheme.

The individual contribution percentage is roughly 50% for each group member. The first leg of the project was carried out on the Virtual Machine, on Harshit's Laptop to experiment with Kernel boot and PM emulation. The second leg of the project was carried out on a Real Machine, in Rajendra's room in WEB, that we loaned from IT for this project. Both of the

contributors have put in efforts whenever and wherever they had an opportunity to do so in each stage of the project. Major challenges were faced while getting WineFS to boot, with emulating PM, exploring new memory mapped applications and workloads that can be tried on WineFS, figuring out Geriatrix, the aging tool, and debugging the memory mapped applications flow required to replicate the experiments performed in the paper.

2. Methodology

To measure the efficiency and effectiveness of WineFS, we needed to boot WineFS introduced by Kadekodi et al on our system. After thorough examination of the WineFS paper and after going through the GitHub page of WineFS, we came to know that the results published in the paper can be recreated using a two socket 28 core, 112 thread bare-metal machine with 64 GB DRAM, 500 GB Intel Optane DC Persistent Memory module and an Intel cascadelake processor. This recommended hardware setup was too expensive for an academic project and could not be arranged. Also, they had listed these specifications as system requirements for booting WineFS. For this reason, we got a bare-metal machine to preemptively prevent any set up related issues in the future. We also performed all the workload and performance measurements on the bare-metal machine since it is not recommended to perform measurements on Virtual Machines due to the TLB conflicts and Memory ballooning effects. We wanted to get a system with as much RAM as possible since we also needed to create a partition for emulating the PM. However, the system with the maximum RAM that we could arrange with the help of IT and Dr. Reddy was 16 GB. Therefore, we decided to move forward with this system after a discussion with Dr. Reddy.

WineFS source code has been embedded in Linux Kernel version 5.1. We booted our system with the latest kernel of Ubuntu and following the instructions on the WineFS GitHub page, compiled the source code and created the image of the Linux kernel 5.1. After booting from the image however, we were unable to spot the WineFS kernel module. Upon debugging, we found another version of WineFS published in the author's personal GitHub repository. With this version of WineFS, we were successfully able to create and insert the WineFS Kernel module. We diffed the two versions of WineFS and found only some configuration changes, so we decided to move ahead with the latter version of WineFS from the author's personal GitHub repository.

For the DRAM based Persistent Memory emulation, we followed two documents that explained the emulation steps. The steps involved updating the config file to enable DAX and PM support, reserving memory regions in the grub configuration, and using the grub configuration to update the kernel images used while booting. However, following either of those documents, we were unable to locate the PM device (/dev/pmem0) in our 5.1 Linux Kernel. Upon debugging, we noticed that the PM device was already present in the original linux kernel image. We figured out that the config file updates should be made Y (built-in) for PM & DAX support, in contrast to the document's instructions to make it M (modular). On changing it to built-in, we were able to locate /dev/pmem0 in linux kernel 5.1 which already had wineFS.

Another challenge that we faced with emulating PM was that we were observing a smaller PM than what we were configuring in the grub file. Since we wanted to maximize the size of PM, we tried out multiple memory reservations. We observed inconsistent PM device sizes across these memory reservations and in some cases the kernel was even crashing. On deeper study and analysis, we found that DRAM has some reserved memory areas and some usable areas. The emulated PM should always belong to these usable memory areas and should not point to the reserved areas. Since we were not careful in our earlier allocations we were observing inconsistent PM sizes and even running into kernel crashes. We allocated the largest usable memory area for the PM and this resolved the aforementioned issues. In this process, we ended up corrupting the linux kernel and getting stuck at a non-booting GRUB 2 shell. We then used a Ubuntu bootable pendrive to clear the system's hard disk and reinstall Ubuntu, since just reinstalling Ubuntu was keeping the same GRUB settings.

After again booting Ubuntu and setting up WineFS and emulated PM, it was time to run and test using the workloads. For running the RocksDB application, we had to generate the YCSB workloads and faced system freeze during the generation process. Looking up solutions for this situation, we came to realize that the available DRAM was insufficient after reserving a huge chunk of it for the PM. Therefore, we removed the PM reservation from DRAM, so the entire DRAM would be available for YCSB workload generation. We reserved the DRAM region for PM again after the workload generation was complete.

The WineFS codebase has a script framework to run the YCSB workloads on RocksDB. After the YCSB workload generation, while running the RocksDB application we faced a numactl related issue. Numactl command runs a process with a specific NUMA scheduling or memory placement policy. In this case the process is the RocksDB application. The command was configured to run assuming the availability of two NUMA nodes whereas our system had a single NUMA node. On configuring the command to run on this available node we were able to get past this issue. Now although the YCSB workload started running, it crashed with an error hinting insufficient PM. Since we could not expand the PM further we had to downsize the record count of 50M in the workload. After multiple trials, we found that 5M was the maximum record count that the PM could fit. Therefore, we used 5M record count in YCSB for evaluating WineFS, NOVA and ext4 to be able to draw a fair comparison.

We targeted aging the file systems now, since the aforementioned workload run gave the results on throughput data for unaged file systems. Geriatrix was the tool used for aging the file systems in the WineFS paper. Geriatrix is an aging suite that induces free space fragmentation by performing controlled file creations and deletions. This is the research work of Kadekodi et al, from CMU and was published in USENIX ATC 2018. The authors of WineFS have instructed that they have provided the aged images on their GitHub page, but the images path does not exist. Therefore, we had to perform aging on the file systems in contrast to the authors' comment which specifically states that the evaluators need not perform aging since it takes too long for aging to complete. Geriatrix has a dependency on the Boost C++ libraries and requires Boost to be installed before it can be compiled. Following the instructions on Geriatrix GitHub page, we encountered an error where Geriatrix was not recognizing the path argument that was

being passed to it for Boost Dependency. Upon debugging, we realized that this path argument was 2 directory levels above Boost binary instead of 3 directory levels above, as was instructed. After correcting this, we were able to compile and install geriatrix.

Another issue that we encountered with Geriatrix was that it ran for about 30 minutes for Agrawal aging profile before it converged. This was initially acceptable since the number of runs was proportional to the file system's image size and we had reduced the image size from 500 GB to 12 GB. However, this run time persisted even if we increased the number of aging runs over the file system image size. This trend was followed in the case of some other aging profiles as well where some of them only ran for about 5 minutes before converging. We debugged a lot on this issue, but could not reach a conclusive solution on how to age it for a longer time period. We developed a workaround by appending and combining all the aging profiles present. We hypothesized that combining the profiles will increase the degree of fragmentation, eventually increasing the time required to reach convergence. We were able to get aged images for WineFS, NOVA and ext4 for 2 hours of aging on this combined profile. While trying for 4 hours however, we saw system crash issues.

The WineFS script framework also has features to create images of aged file systems and to mount unaged and aged file systems. While creating the images for ext4, we found an issue in the scripts used for creating images of the aged file system. An argument was missing in the e2image command which was preventing creation of aged ext4 filesystem images.
We had initially planned to compare wineFS against NOVA, ext4 and xfs file systems. However we had to drop xfs since we were not able to create its aged image. The create_aged_aimges script uses xfs_metadump command to create a xfs image. This command is a debugging tool that can only be used to copy metadata from an unmounted xfs file system or a read-only mounted xfs file system. Since we wanted to create an image from a mounted writable file system this command was throwing an error. Since the entire xfs setup was built around this command and replacing this command would have involved rework, we decided to drop it from our evaluation.

As we had planned to measure the performance of WineFS against new workloads, we tried using the Chirp, which is a Twitter-like workload generator framework. We were able to generate the workloads, but the workload was not compatible with RocksDB or LMDB. In addition to that, we were not able to find a DB application that could be used to run the generated workload. Therefore, we decided to not move forward with it.

We also came across a Persistent Memory version of Redis called pmem-Redis. This application would have been very much appropriate for our memory mapped use case had it worked. However, we ran into multiple setup problems with the compilation. On debugging, we found that the source code was outdated with respect to the Redis code base and we ended up making multiple changes to the actual code. Since this was not expected and these changes were not sufficient to fix all the compilation errors, we were forced to drop this as well.

While trying to run the workloads using the LMDB benchmarks, we found that the WineFS paper only compares the performance of the fillseqbatch workload. The LMDB benchmark comes preloaded with multiple other workloads. Therefore, we enabled those workloads and captured the results for comparison of performance across different File Systems.

A major logistical challenge that we faced during the execution of this project was the lack of remote access to the system. This ended up delaying the debugging of geriatrix aging time issue, YCSB workload executions on aged images and prevented us from resolving the xfs issue. We tried to counter this issue to the best of our ability by automating the processes, so the scripts can automatically get sourced one after another and by ensuring that correct arguments are passed for the same. We created an ager script that ran a loop on each of the file systems for all the following operations - mount the file system, run aging on the file systems, create an aged file system image. We encountered issues in this as well, since mount and unmount commands need sudo permission and geriatrix only works without sudo. This was leading to sudo getting timed out which further contributed to delaying our project execution.

the steps you took to reach (a), difficulties you encountered, changes in plans etc.

3. Experiments and Evaluation

For the purpose of this project, we have used one VM for forecasting setup issues and one bare-metal machine for system set up and performing the actual evaluation of WineFS and comparison with NOVA and ext4. The bare-metal machine has the system configuration of 1 socket, 2 threads, 16 GB DRAM used to emulate 12 GB PM and 4 GB DRAM and Intel Family 6 Haswell Microarchitecture.

As illustrated above, the system setup we have used is different from the setup that the WineFS paper suggests. Because of this, the results are expected to be different even on the same workloads. However, we expect similar trends in the results at least when we use the same aging profiles to age the file systems.

For running the YCSB workloads on RocksDB application, the original setup for testing WineFS contains 50M record entries for 500GB PM. However, this was leading to system freeze when tried on our system. As a consequence, we had to scale it down to 5M record entries for 12GB emulated PM. We used three aging profiles for aging the three file systems, WineFS, NOVA and ext4. These three aging profiles used are Agrawal, Meyer and Combined. Combined is a custom aging profile created with the intention to create even further fragmentation. This profile is produced by combining the three setup files of all the aging profiles available in the Geriatrix framework.

We got the results as shown in Figure 1(a),(b), (c) and (d). The Y-axis in these denote operations per second.

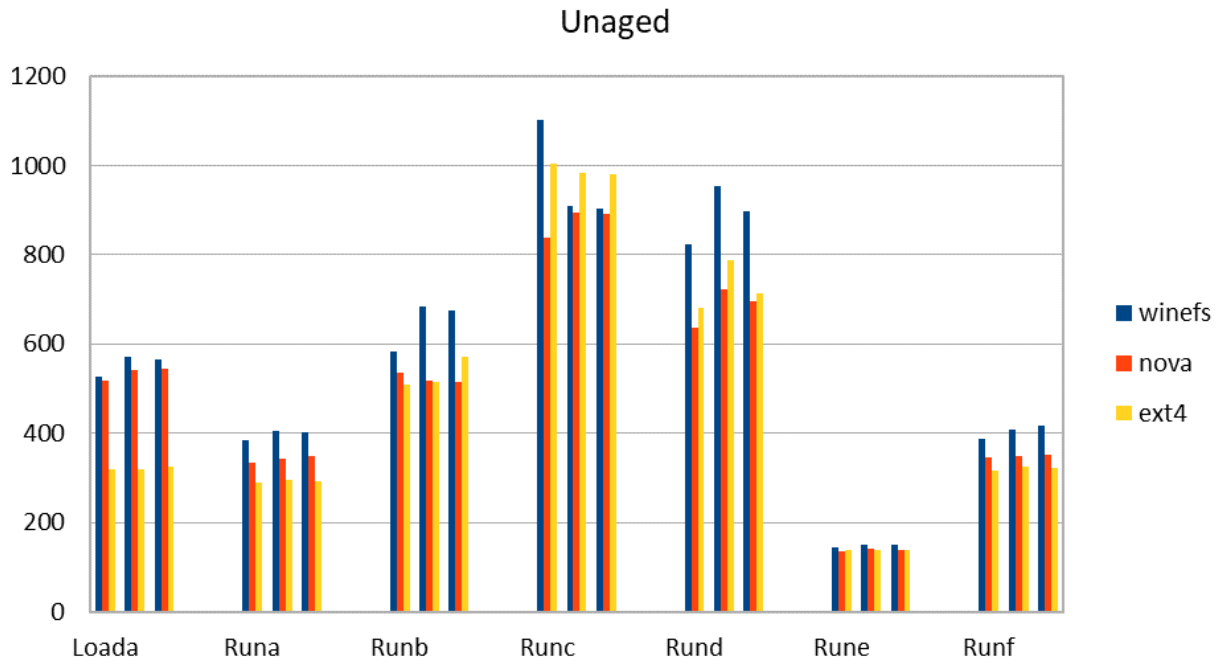


Figure 1(a): YCSB on RocksDB evaluation on unaged file systems

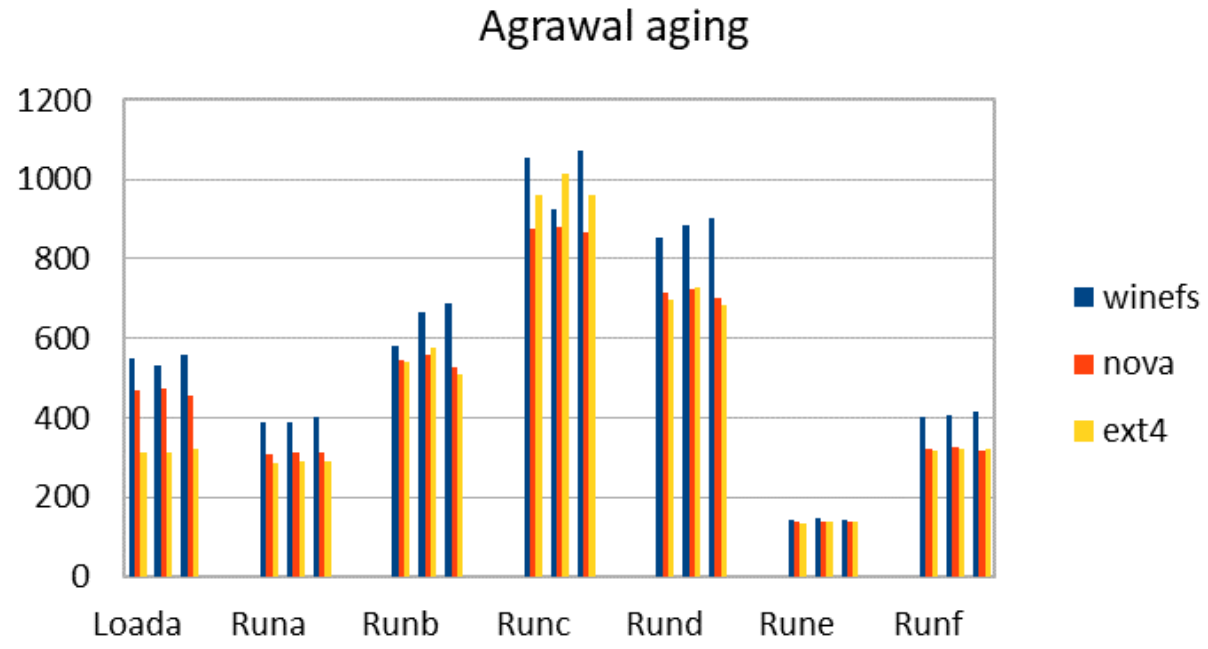


Figure 1(b): YCSB on RocksDB evaluation on file systems aged using Agrawal aging profile

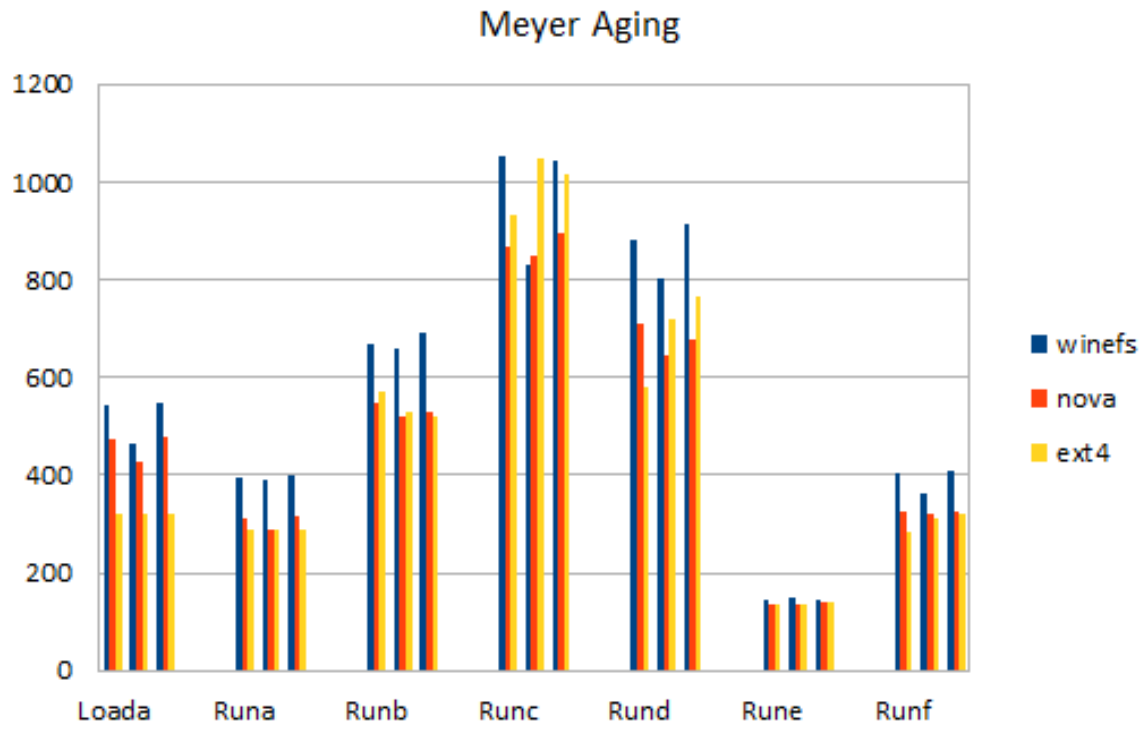


Figure 1(c): YCSB on RocksDB evaluation on file systems aged using Meyer aging profile

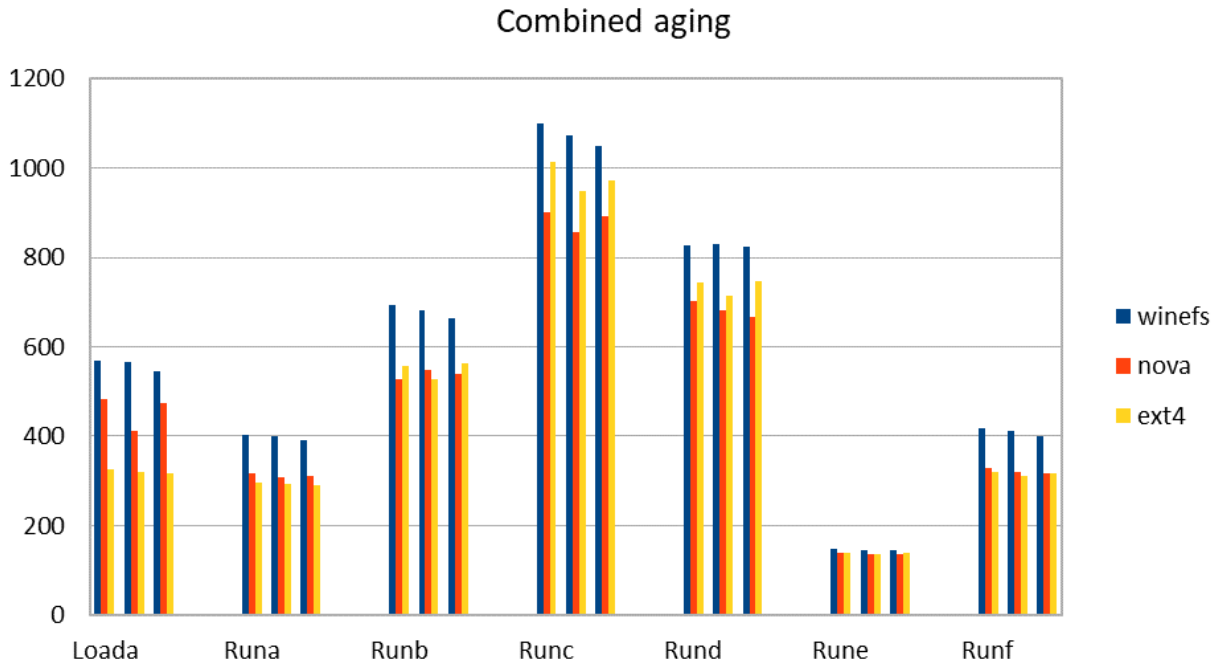


Figure 1(d): YCSB on RocksDB evaluation on file systems aged using Combined aging profile

From the figures and data, we inferred that WineFS when tested on YCSB on RocksDB overall performs better than NOVA and ext4. In both the aged and unaged file systems, WineFS

performs better than NOVA and ext4, whereas this gap between WineFS and NOVA/ext4 increases as the aging and fragmentation increases.

For running the fillseqbatch workload on LMDB applications, the setup used for evaluation in the WineFS paper for db_bench benchmark uses a 50M record count . However when we tried on our system, this was leading to false run capturing no performance data. On scaling it down to 500K and 5M records, we started observing real performance data. We used three aging profiles for aging the three file systems, WineFS, NOVA and ext4. These three aging profiles are Agrawal, Meyer and Combined.

We got the results for 5M records as shown in Figure 2. The Y-axis in these denote operations per second.

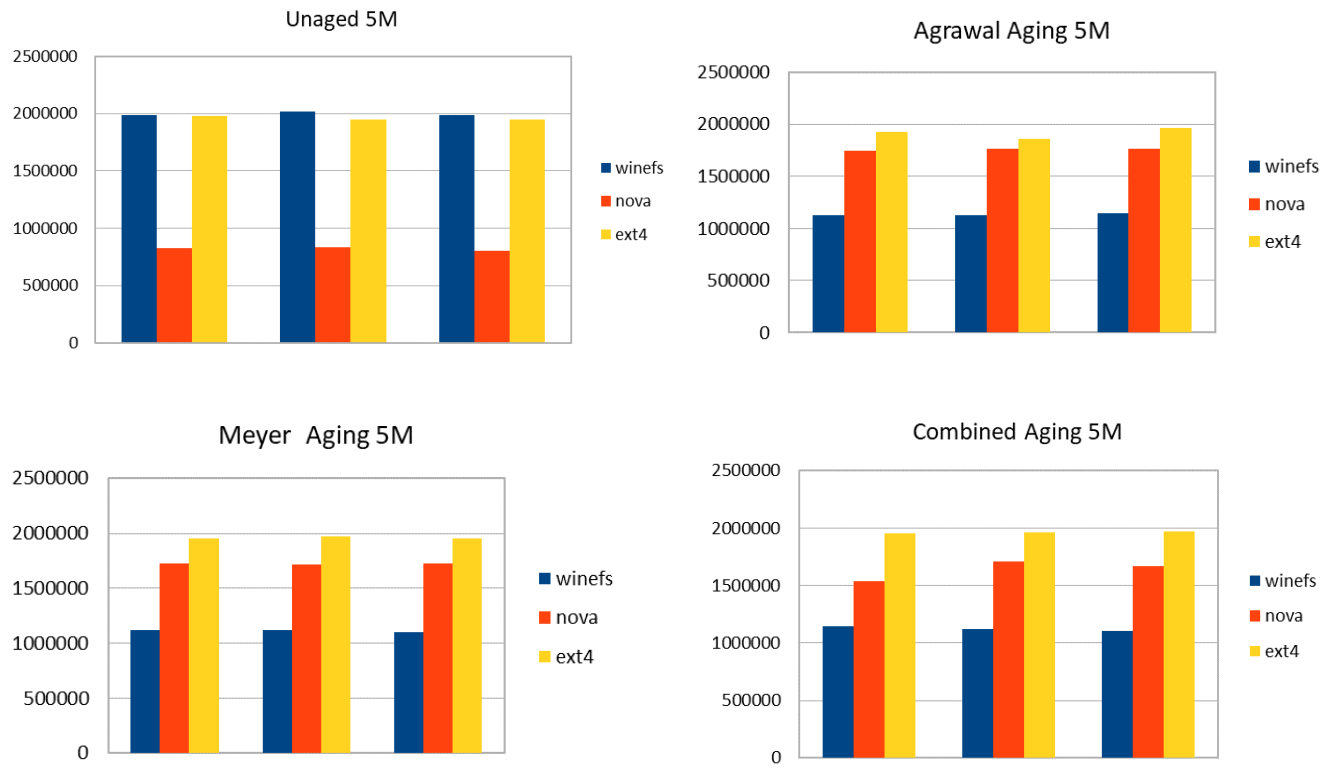
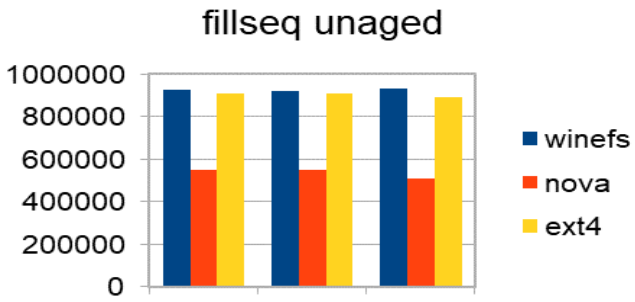
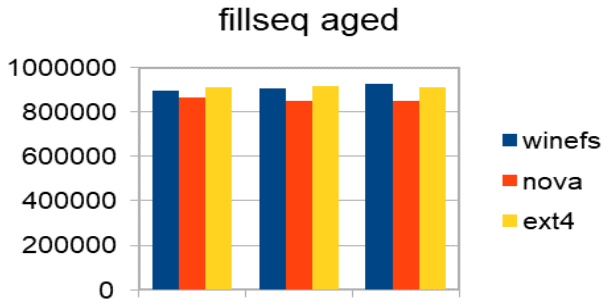
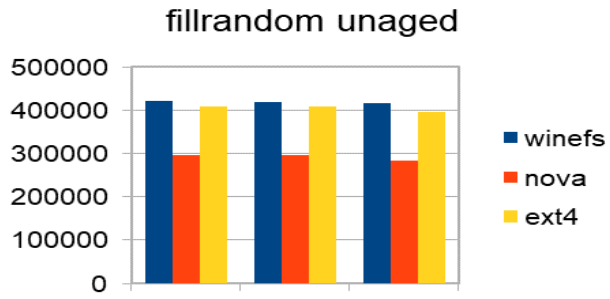


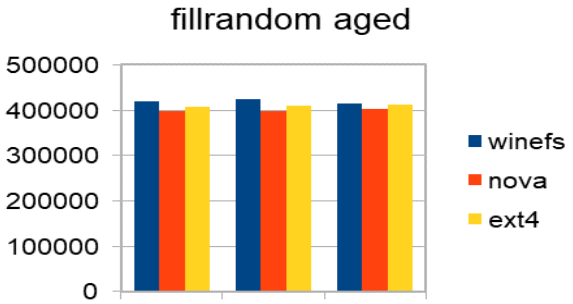
Figure 2: Evaluation of fillseqbatch on LMDB for 5M record count

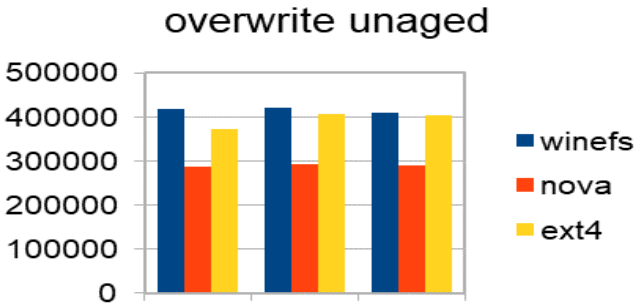
In Figure 2, we observe with 5 million record count that WineFS performance degrades the most as compared to the other File Systems. Since this result was not expected, we performed some experiments using the other available workloads in the LMDB application's dbbench benchmark.

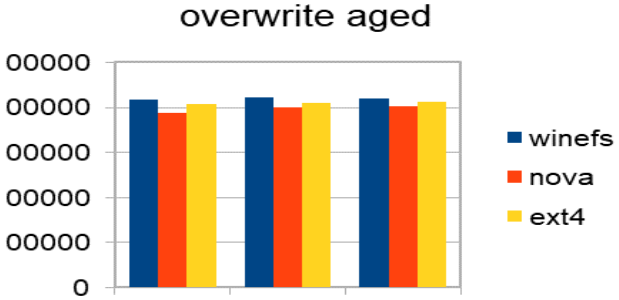

fillseq unaged
1000000
800000
600000
400000
200000
0
winefs
nova
ext4

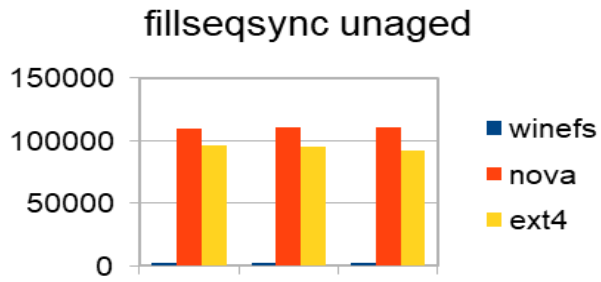

fillseq aged
1000000
800000
600000
400000
200000
0
winefs
nova
ext4

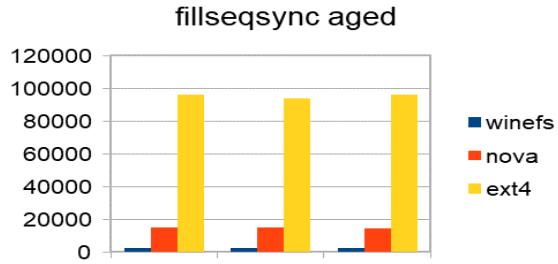

fillrandom unaged
500000
400000
300000
200000
100000
0
winefs
nova
ext4

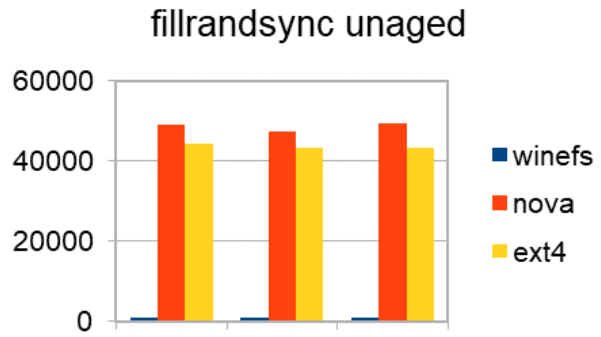

fillrandom aged
500000
400000
300000
200000
100000
0
winefs
nova
ext4

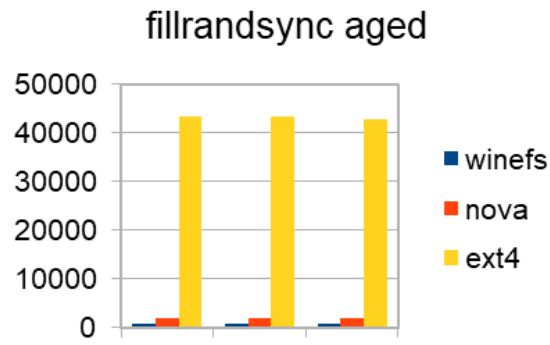

overwrite unaged
500000
400000
300000
200000
100000
0
winefs
nova
ext4


overwrite aged
00000
00000
00000
00000
00000
0
winefs
nova
ext4


fillseqsync unaged
150000
100000
50000
0
winefs
nova
ext4


fillseqsync aged
120000
100000
80000
60000
40000
20000
0
winefs
nova
ext4


fillrandsync unaged
60000
40000
20000
0
winefs
nova
ext4


fillrandsync aged
50000
40000
30000
20000
10000
0
winefs
nova
ext4

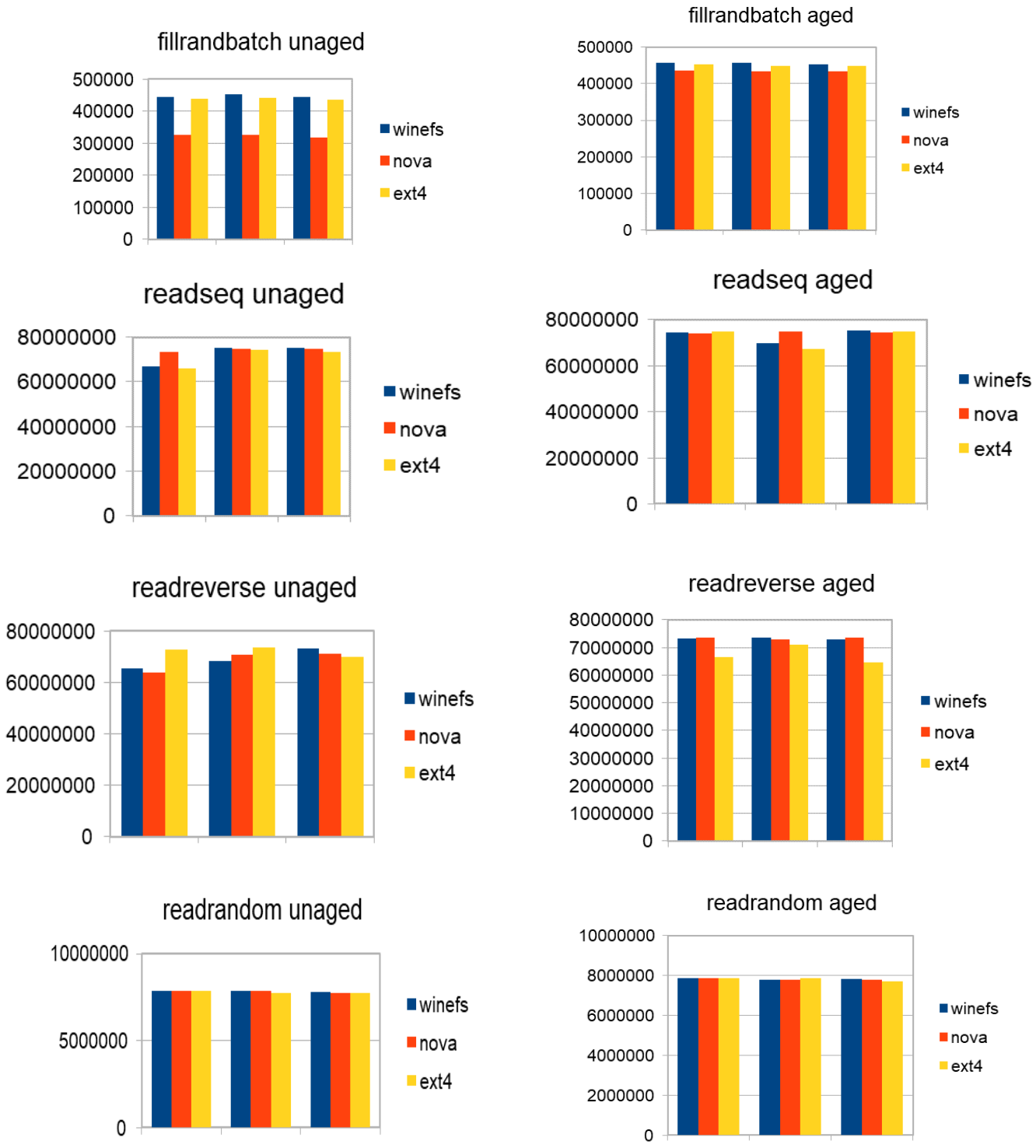


Figure 3: Evaluation of all dbbench workloads on LMDB for 5M record count on unaged and aged File Systems using Agrawal Aging Profile

From Figure 3, we observe that WineFS performs the best among the three aged file systems in fillseq, fillrandom, overwrite, fillrandbatch and readreverse workloads. However, it is not the best performing file system in fillseqsync, fillrandsync, readseq and readrandom workloads. The data in Figure 3 shows aging done with Agrawal profile. The experiments using Combined aging profile also yielded similar results. Another peculiar observation from the figure is that the performance of NOVA appears to be improving upon aging. This contradicts the foundation concept established by the WineFS paper. Thus, to debug further, we looked at the number of page faults incurred in Unaged and Aged File Systems. Although we did not observe any differences in the number of hard page faults incurred, we observed that there is a significant difference in the number of soft page faults incurred. The soft page fault data for the fillseqbatch workload is illustrated in Table 1.

Unaged FS			
fillseqbatch	**Run10**	**Run11**	**Run12**
winefs	6609	6612	6608
nova	1680260	1680256	1680260
ext4	28073	28077	28079
Agrawal aged FS			
fillseqbatch	**Run10**	**Run11**	**Run12**
winefs	1561189	1554555	1554547
nova	200286	200280	200281
ext4	28074	28073	28068

Table 1: Number of Soft Page Faults for Unaged and Agrawal aged file system

The same trend follows for the rest of the workloads in the db_bench benchmark as well. The significant drop in NOVA's page fault backs the peculiar performance observed for NOVA above. However our main subject i.e. WineFS incurs the maximum page faults among the three FS after aging. This is a direct contradiction to the page fault result published in the WineFS paper.

4. Lessons Learned

This project provided us with multiple learning opportunities from various domains starting from Linux kernel to YCSB, RocksDB, LMDB frameworks and Persistent memory in itself. We have condensed our learning flow into the following bullet points.

1. We gained more knowledge about the Linux kernel compilation and booting process. We learnt to troubleshoot kernel corruption situations.
2. Through the project setup with PM emulation, we gained knowledge about DRAM organization. We got to know that there are certain areas in DRAM that are reserved by

the Kernel that are also used during Kernel boot and that there are certain usable sections in DRAM. To allocate the largest possible chunk to the PM, this organization must be understood and the PM should be allocated accordingly.

3. We gathered a good background on memory mapped access to PM in the context of the WineFS paper since we collected and analyzed extensive data on this kind of access.
4. We collected a good deal of information on the mounting and unmounting process in Linux and the way it links the storage device and file system to the mount point.
5. To generate the downscaled (50M -> 5M) YCSB workloads, we studied and made modifications to the YCSB framework, especially the process of generating workloads based on the parameters given.
6. RocksDB and LMDB benchmark runs using the NUMA policy enforcement through numactl command. We understood the working of the numactl command to resolve the cpunodebind issue.
7. Aging is a concept that we were not even aware of until we undertook this project. It was fascinating to learn how aging can impact fragmentation and thus, the memory mapped performance. We got the opportunity to conduct experiments and get hands-on experience of carrying out aging and observing its impact.
8. We used the Geriatrix tool in this project for aging the file systems. Geriatrix aging tool gave us an overview on aging. We learned that aging is a statistical process and is based on three parameters : size distribution, age distribution and directory distribution.
9. We got acquainted with the WineFS end to end flow in the process of running multiple workloads. This enabled us to modify and create new scripts in order to customize the flow for our use case.
10. We tried to run new workloads/benchmarks on PM aware applications that had not been carried out in the WineFS paper. In this process, we explored multiple KV store applications such as pmem-redis, LevelDB(industry based) and ChameleonDB, SLM-DB (research based). We might not have been able to incorporate them successfully into our project but we got to ride a good and challenging learning curve.

5. Conclusion

The objective of our project was to recreate and evaluate WineFS by testing it using various workloads aged using multiple aging profiles. WineFS, a Persistent Memory (PM) file system, preserves huge pages even after the file system is aged significantly.

For this project, we re-evaluated WineFS by recreating it on our hardware, a scaled down version of the hardware used in original experiments, to check if the results are consistent across hardware and workloads. Since we used RAM based emulated PM, we had to scale down the workloads to fit the capacity of our system's PM and DRAM.

Through the experiments conducted, we found that WineFS produces inconsistent performance results on the LMDB application's workloads. This contradicts the results and claims made by Kadekodi et al in the paper. We do not refute the claims made by the paper with absolute

certainty because there are multiple potential points of failure that could have led to these inconsistent results. We suspect one of the following aspects. First, since we did not verify the delay of the emulated PM, it could be that the emulated PM is just getting mounted as a device and not giving the delay corresponding to PM accesses. Second, the hardware setup and the workloads used in the WineFS paper were scaled down due to hardware limitations. This could have led to these inconsistent results. Third, it could be the case that the file system aging process is not working as intended. Fourth, since we are seeing inconsistent performance results and page fault numbers in LMDB workloads, it could be the case that the LMDB workload setup provided in the WineFS codebase has some inconsistencies. We have checked this quite thoroughly and compared it with the latest revision of the LMDB. There are some minor differences in the two codebases and this could be the reason behind the inconsistency in LMDB results. These aspects need to be analyzed further to identify the actual reason behind the inconsistent results or to refute the original claims made by the WineFS paper. This further analysis lies beyond the scope of this project and it can be addressed in future improvements.