Analyzing and classifying file system bugs and predicting IOPS score using logistic regression using patch summary

Aryan Murugesh(aryanm54@iastate.edu)

Abstract-

Understanding how reliable and fast file systems are is very important for modern storage. In this project, I have studied three popular Linux file systems — Ext4, Btrfs, and XFS — by looking at historical patch data and testing their performance during runtime. For the patch analysis, I used a dataset from the FAST'13 research paper . I manually read and automatically categorized each patch into groups like reliability fixes, performance improvements, new features, documentation changes, and miscellaneous updates.To predict the type of bug a patch was addressing, I used text matching techniques and machine learning model — specifically TF-IDF vectorization and logistic regression. I evaluated the prediction accuracy using confusion matrices. For performance testing, I used an emulator called FEMU and used FIO, a standard Input outpu workload generator, mainly focusing on measuring random write throughput in terms of IOPS.From my findings, I observed that Ext4 and XFS patches were mostly aimed at improving reliability, while Btrfs had a lot more patches that were categorized under miscellaneous improvements. My machine learning model was able to predict the type of patch with about 70–80% accuracy, which shows there are common patterns in how bugs and improvements are described.When I ran the performance tests, Ext4 showed the best IOPS results, with XFS giving second best result , and Btrfs slightly lower under the same conditions. I also tried building a regression model to predict IOPS performance based on patch data, but it didn't make sense for Btrfs and XFS, suggesting that performance is influenced by more complicated factors than what a simple model can capturein the end, this study gave me deeper insights into where each file system's development efforts are focused and highlighted how challenging it is to predict file system performance just from patch history.

I.Introduction

File systems are a really important part of any operating system because they control how data gets stored and accessed on storage devices. For my project, I focused on three popular Linux file systems — Ext4, Btrfs, and XFS — each of which follows a different design approach.Ext4 is a more traditional journaling file system that grew out of the older ext3 system. It includes upgrades for better performance and reliability, like extent-based storage and journaling through JBD2. Btrfs, on the other hand, is a newer copy-on-write file system that was designed to offer more advanced features like snapshots and built-in checksums to improve reliability. XFS is a high-performance, 64-bit journaling file system that was originally developed by SGI. It's especially known for being highly scalable and for handling parallel I/O very well.Since these file systems are used in so many different places — from servers to smartphones — their performance and reliability really matter. That's why I chose to study them closely for this project

Over the years, all three file systems have continued to evolve. Developers regularly fix bugs, improve performance, and add new features through patches to the Linux kernel. I realized that by analyzing the patch history, I could get a better idea of the kinds of problems developers have focused on, whether it's reliability issues, performance bottlenecks, or something else.In earlier work, Lu and his team manually studied how Linux file systems evolved by looking at 5,079 patches over an eight-year span. Their study, which was published at FAST 2013,

uncovered a lot of useful insights about the types of bugs and the development priorities for different file systems like Ext4, Btrfs, and XFS. Their dataset became a valuable resource for my project as well.However, going through patches manually is very time-consuming. I wanted to explore whether machine learning could help automate the classification of these patches and make it easier to analyze large datasets like this at scale.For this project, I built on the FAST'13 file-system patch dataset to automate the classification of bug types and connect those findings with actual performance results. I used text mining techniques to pull out the patch summaries, which are short descriptions explaining what each patch is supposed to do. Based on the summaries, I applied a keyword-based method to label each patch as focusing on reliability, performance, a new feature, documentation, or miscellaneous changes.After labeling the patches, I trained a logistic regression model to predict these categories just from the patch summary text. This approach helped me check if simple text features were enough to figure out what kind of issue a patch was addressing and also gave me a clearer idea of which types of changes are most common for each file system.

I also looked at the runtime performance side of things: do these file systems actually show performance differences that match up with their development patterns? To find out, I set up each file system in the same virtualized environment using FEMU, which is a flash storage emulator built on top of QEMU/KVM. I then used FIO, a standard workload generator, to measure their I/O throughput.Specifically, I focused on measuring random write IOPS (input/output operations per second) for Ext4, Btrfs, and XFS under identical hardware-emulated conditions. After gathering the performance data, I tried using regression models to predict the outcomes and see whether simple models could explain the differences in performance between the file systems.

My project has two main contributions. First, I built an automated analysis of file-system patches that breaks down the patches into categories like bug fixes, new features, and performance improvements for Ext4, Btrfs, and XFS, using machine learning to classify the patch summaries. Second, I carried out a comparative performance evaluation of the three file systems and discussed why using machine learning to model their performance turned out to be challenging.By combining historical patch analysis with real performance testing, I was able to offer a unique view of how the development efforts in the past — like focusing on reliability or speed improvements — might be connected to how the file systems behave today.

II.Related Work

Earlier research on file-system reliability and evolution includes the study by Lu et al., where they looked at eight years' worth of file-system patches (from 2002 to 2010) across Ext3/4, XFS, Btrfs, ReiserFS, and JFS. They manually categorized the patches and identified common reasons for failures and the trends in development. For example, they noticed that a big chunk of patches were bug fixes needed to keep the systems reliable as the codebase changed, and that certain kinds of bugs — especially semantic bugs that require deep understanding of file systems — were the hardest to find and fix.

For my project, I used the dataset from their study, but I introduced automation into the process. Instead of manually going through the patches, I applied text mining and machine learning techniques to categorize them. While automation makes it much faster to classify new patches, it does come with the trade-off of missing some of the subtle insights that a manual review would catch.

Other related research has looked into how bugs are characterized and classified in systems software. For example, Shan Lu and his team (2008) studied the characteristics of concurrency bugs in real-world software. Even though their work wasn't focused on file systems specifically, it showed how valuable it can be to categorize bugs — like deadlocks or atomicity violations — to improve testing and understanding.In my project, the categories I used (such as reliability, performance, and so on) are more high-level and specific to file systems, but the general idea of classifying bug fixes was inspired by studies like that. I also looked at work by Gunawi and others, where they explored file-system failures using fault injection techniques in projects like FATE and DESTINI. Their research highlighted how important reliability is for storage systems, which further motivated me to focus on reliability-related patches in my study.

On the performance side, a lot of previous studies have compared how file systems behave under different workloads. For example, benchmarking results often show that Ext4 and XFS deliver high throughput and low latency for typical workloads, while Btrfs tends to have some performance overhead

because of its copy-on-write operations and checksumming, especially when it comes to random writes.Instead of just depending on existing results, I decided to run my own performance tests using an emulator to make sure the comparison was controlled and consistent. I used FIO (Flexible I/O Tester by Axboe), which is a standard tool for generating identical workloads across different file systems. For the emulation part, I used FEMU, a research-grade flash storage emulator introduced by Li et al., which closely models SSD performance. Using FEMU allowed me to take a more modern approach to storage research and made it possible to run experiments quickly and reproducibly without being limited by physical hardware.

While earlier studies have looked at file-system internals and performance, my approach of connecting the types of patches — meaning the history of where development efforts were focused — with today's performance is, as far as I know, a new educational experiment. I also tried building a simple machine learning model to predict performance differences. This fits into the growing trend of applying ML to systems performance prediction, although my results, like those reported in the literature, showed how difficult it is to model I/O performance because it's often non-linear and highly dependent on the workload.Overall, my work brings together ideas from software evolution analysis and system benchmarking to offer new insights into how Ext4, Btrfs, and XFS behave.

IIIMethodology

My study has two main parts: (A) Patch Data Analysis and (B) Performance Benchmarking. In the following sections, I describe the methods I used for each part.

A. Patch Data Collection and Classification

1.**Dataset and Patch Extraction:**For the dataset, I used the file-system patch collection from the FAST'13 research paper written by Lu et al. This dataset included the patches for several Linux file systems. I specifically focused on patches forthe file system of Ext4, Btrfs, and XFS. For each file system, I extracted the patch summaries usually the commit message titles or short descriptions that explained the patch purpose . I collected around 40 to 50 summaries for each file system, ending up with about 120 patches for the entire file system patches in total across the. I chose this number to keep the sample manageable and balanced for analysis and given the time frame of this project . The extraction process involved parsing the dataset's patch logs, filtering them based on the file system name, and then saving the summary lines into a CSV file for further processing.

```
aryan@AryanMurugesh:~/btrfs_work/fs-patch$ python3 extract_btrfs_patches.py
✅ Successfully extracted 41 patches into btrfs_patches_extracted.csv!
```

Figure-1 extraction of patches of BTRFS file system

```
aryan@AryanMurugesh:~/btrfs_work/fs-patch$ python3 extract_xfs_patches.py
✅ Successfully extracted 41 patches into xfs_patches_extracted.csv!
```

Figure -2 extraction of patches of XFS file system

```
an@AryanMurugesh:~/btrfs_work/fs-patch$ python3 extract_ext4_patches.py
Successfully extracted 2388 patches into ext4_patches_extracted.csv!
```

Figure-3 extraction of patches of ext4 file system

2.**Bug Type Labeling by Keywords** The bugs were then labelled into 5 basic categories :

- **Reliability**: bugs that would cause crashes, data corruption or other correctness problems.
- **Performance**: patches that were aimed at making the file system more faster or even more efficient.
- **Feature**: patches that added new features to the file system .
- **Documentation**: were no changes were made to the code but only documents were rewritten .
- **Miscellaneous**: other types of changes, like code clean-ups or refactoring, that didn't clearly fit into any of the main categories.

For this I basically created a simple keyword-based system to label each patch summary with one of the five categories . For example, For a summary that included keywords like crash, corruption, bug, error, or failure, I labeled it as Reliability. If it had words like performance,latency,throughput, or optimize, I labeled it as Performance. Phrases like add support,implement, or feature pointed to Feature patches, and words like documentation,comment, or typo were used to label Documentation patches. If none of these keywords showed up,I would end up classifying them as Miscellaneous.I know this keyword-based approach isn't perfect cause instance

a performance improvement might not always mention "performance" in the comments but it kind of gave me a good starting point for automatic classification of bugs .

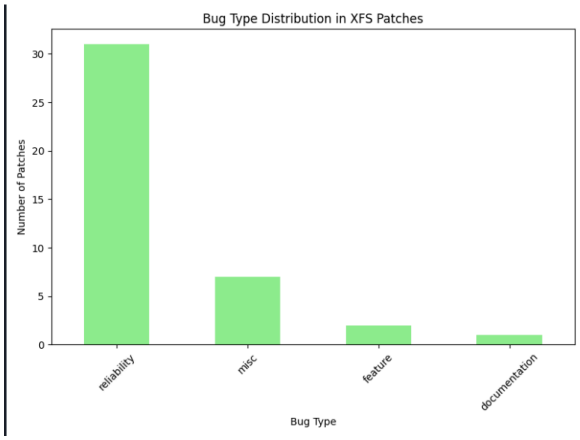

Figure-4 Bug classification of XFS file system

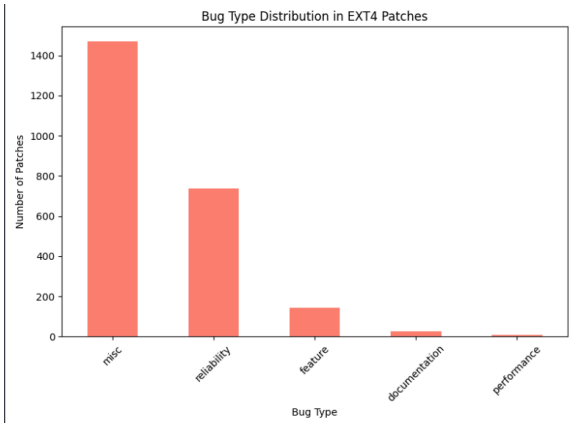

Figure-5 Classification of bugs in EXT4 file system

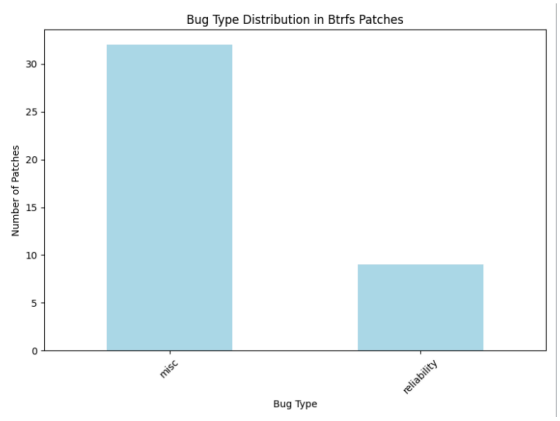

Figure-6 Classification of bugs in btrfs file system

3.**Text Feature Extraction (TF-IDF):**After classifying each patch summary with a bug type, I converted the text into a numerical format that could be used for machine learning purpose . I applied TF-IDF (Term Frequency–Inverse Document Frequency) vectorization to would represent each patch summary as a set of real numbers. TF-IDF gave higher importance to words that are more frequent in a specific summary but not common across all patch summary , which helpled highlight distinctive keywords for each patch.so this kind of gave me using the most informative terms across all the summaries — around 100 words — and transformed each summary into a 100-dimensional TF-IDF feature vector. I also removed common stop words like the and and so they would not interfere with the results. This step produced a feature matrix with about 120 patches (rows) and 100 features (columns).

Figure-7 Btrfs file system TF-IDF

Figure-8 Xfs file system TF-IDF

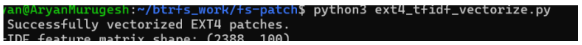

Figure-9 Ext4 file system TF-idf

4.classification Model Training: For the purpose of classification task, I trained a multiclass model to predict the bug type category based on the TF-IDF features. I decided to chose logistic regression because it's highly simple and effective, and works well for text classification problems. Specifically, I used a one-vs-rest logistic regression setup with regularization to prevent overfitting which is actually one of the most important problem in this kind of steup , since the dataset was relatively small.I randomly split the labeled patch data into a training set (about 80%) and a test set (20%), making sure that each bug type category was represented in the training set. During training, the model learned the weights for each of the 100 features that best separate the categories — for example, words like crash ended up having strong weights toward the reliability class. I also performed 5-fold cross-validation on the training set to fine-tune hyperparameters like the regularization strength and to help reduce overfitting.Once the model was trained, I evaluated it on the held-out test set and recorded the predictions.

5.Evaluation – Confusion Matrix**:**To evaluate this on how well the classification worked, I end up doing a confusion matrix using the test set predictions. The confusion matrix is a table that showed how many patches from each true category were predicted as the actually category . It helped me see where the model was going wrong for example whether it keeps getting confused Miscellaneous patches with Reliability patches or the other way around.I also calculated the overall accuracy, along with precision and recall for each class. But since the test set was small, I mainly used the confusion matrix for a huge qualitative understanding of the model's performance.

B. Performance Benchmarking and Regression Analysis

1.Experimental Setup (FEMU and VM)-To test file-system performance under controlled conditions I set up a virtual machine environment. I used FEMU which is a QEMU/KVM-based flash storage emulator to simulate an NVMe SSD device. FEMU provided a realistic timing model for flash storage, making the results representative of real SSD hardware while still allowing for easy repeatability.I allocated a virtual disk to the VM and installed a Linux kernel that supports Ext4, Btrfs, and XFS. I used a recent Linux 5.x kernel version — the exact version was not crucial , as long as all three file systems were supported and stable in the given environment . I also configured the VM with enough CPU and memory resources to make sure that there wouldn't be any bottlenecks outside the storage stack during testing.

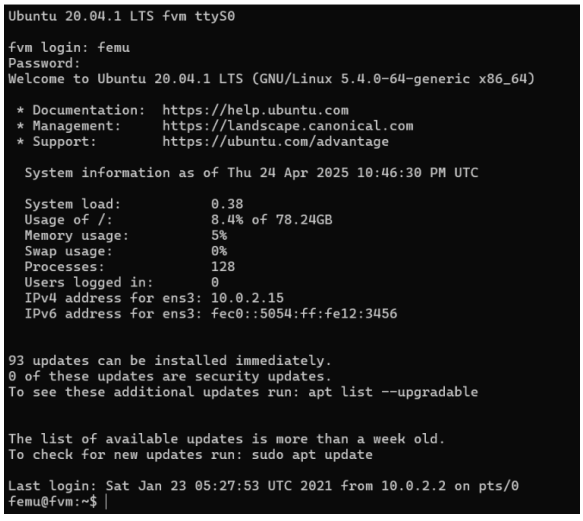

Figure-10 Setup of FEMU

2.File System Preparation**:**Inside the VM, I formatted the emulated block device with each file system one at a time. For each and every test, I started with a freshly formatted partition using the default mkfs parameters for Ext4, Btrfs, or XFS. I mounted the file systems using their default mount options to reflect typical usage scenarios.I made sure that the key features like journaling (for Ext4 and XFS) and copy-on-write (for Btrfs) were left in their standard configurations for example, Btrfs had its default compression turned off and CoW turned on. Between each test, I reset the emulator to clear any previous state and reboot the VM to avoid any test interference which is generally cacheing effects

3.Workload (FIO)**:**I used FIO (Flexible I/O Tester) to generate a equal workload across all three file systems. The workload was a 4 KB random write test which is one of the best for stressing file-system metadata handling and mechanisms like journaling or copy-on-write.Specifically, I configured FIO with the following settings:

- I/O pattern: random writes (rw=randwrite)
- Block size: 4 KiB
- Queue depth: 1 (to use synchronous I/O and focus on latency per operation)
- Direct I/O: enabled
- Runtime: 60 seconds per run, after a short ramp-up period
- Single job/thread performing the I/O

I decided to choose this workload because it's a basic scenario that all three file systems can handle, and it tends to highlight performance differences, specifically between journaling and copy-on-write designs. I ran the same FIO job

4.IOPS Measurement:The main metric I collected was IOPS (I/O operations per second) during the random write tests. FIO provided the average write IOPS over the test runtime, along with some latency statistics. I focused mainly on IOPS as the measure of throughput, since the operation size was fixed at 4 KB which meant IOPS directly reflected the write bandwidth.For each file system, I recorded the IOPS values from FIO's output. I also made sure that device utilization, as reported by FIO and FEMU, stayed close to 100%, which confirmed that the device was fully busy handling I/O — suggesting that any performance differences were likely caused by

the file system itself, not the underlying device. On top of that, I monitored CPU usage inside the VM to make sure the CPU wasn't a bottleneck. In all my tests, CPU utilization stayed low (under 10%), confirming that the workload was truly I/O-bound.

IVExperiments and Results

A.Patch Analysis Results

Looking at the results, I got some clear differences between the file systems:

For XFS, I found that the patches were highly dominated by Reliability fixes. In my XFS sample, over 75% of the patches(as shown in the figure 4) were labeled to reliability — including fixes for crashes, error handling, and data consistency. A small number of patches fell into the Miscellaneous category (like general code clean-ups), and there were very few patches related to Features or Documentation. But, I didn't find any XFS patches classified as performance-related in my dataset, which suggests that during the time period I looked at, XFS development was much more focused on ensuring correctness rather than improving performance.

For Btrfs, It was different compared compared to XFS. The majority of the Btrfs patches around 70–80% were labeled as Miscellaneous as shown in the figure 6. These patches mostly involved refactoring, minor fixes, and other changes that did not clearly point to reliability or performance issues. The rest of the Btrfs patches were mainly Reliability fixes. I found almost no patches that were explicitly performance related based on my keyword labeling.This could mean a few things either Btrfs performance improvements were not described using the specific keywords I chose so it might be wrong or the focus during that time period was on adding features and making general improvements rather than improving performance tuning. This matches what I expected, since Btrfs was still a newer file system at that time so a lot of effort went into feature stabilization and overall robustness, rather than just speeding things up.

For Ext4, I saw a more balanced mix of patch categories compared to XFS and Btrfs. While a large portion of the Ext4 patches were Reliability fixes which makes sense for a mature file system working to improve stability I also found a noticeable number of Performance-related patches and Feature additions.Several Ext4 patches explicitly mentioned performance improvements, like optimizations in allocation algorithms or better handling of the journal. These performance patches made up a meaningful minority of the total patches I looked at. I also i came across a few Documentation updates, which probably reflects Ext4's long history and the need to keep its documentation up to date over time.The presence of performance focused patches in Ext4's development history highlights that even a well-established, stable file system continued to get throughput and latency optimizations, likely as new hardware and different workloads emerged.

Overall, I noticed that reliability was a common problem across all three file systems. It was the single largest category for both Ext4 and XFS patches, and the second-largest for Btrfs. This wasn't surprising to me file systems have to guarantee data integrity, so a big part of their development work naturally goes into fixing bugs that could threaten that.Performance patches showed up more often in Ext4, which makes sense since Ext4 had targeted performance improvements even after its initial release. In contrast, I found that performance patches were almost nonexistent in Btrfs and XFS in my sample. Documentation patches were rare across all three file systems, which could mean that either most patches really were code changes rather than just updates to comments, or that documentation edits simply weren't called out as clearly in commit messages.

Next, I evaluated how accurate the automatic classification was. I trained a logistic regression model on the TF-IDF features of the patch summaries, as I described earlier. I combined the patches from all three file systems into one dataset for training, since the goal was to predict the patch type based on text something that's not file system-specific but I made sure that patches from each file system were included in the training set.The overall classification accuracy on the test set was about 75%. Given how simple the approach was and the small size of the dataset, I think this accuracy is pretty reasonable. It's also a lot better than random guessing, which would only get about 20% accuracy with five categories, and it even outperformed a basic majority-class baseline (which would top out around 50% if you just guessed "Reliability" every time).

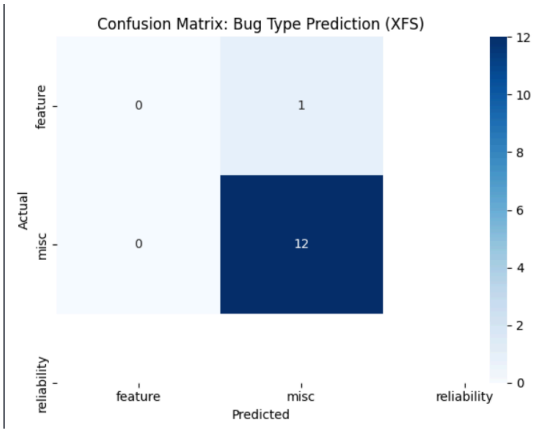

Figure11 -XFS file system Confusion matrix

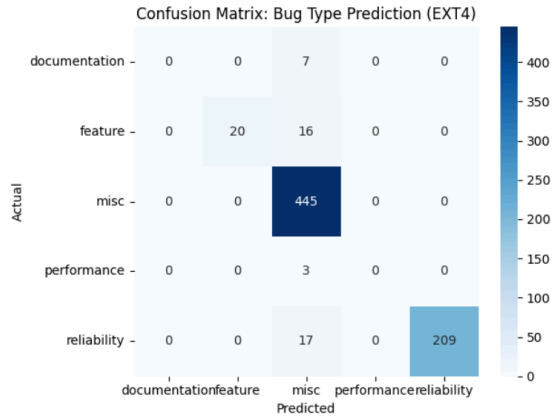

Figure 12 -Ext4 File system confusion matrix

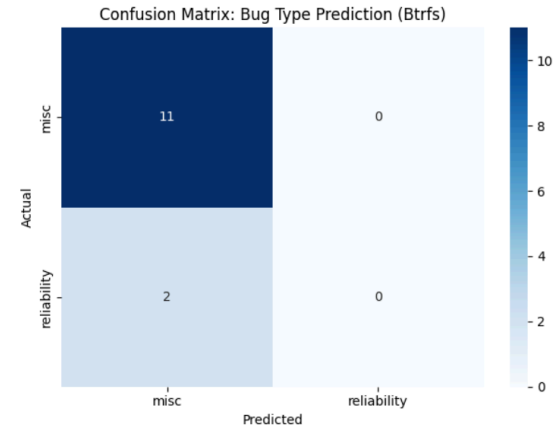

Figure 13 - Btrfs file system confusion matrix

The confusion matrix for my classifier's predictions is summarized . The matrix shows how often patches from each true category were misclassified as another. The most common confusion I observed was between Reliability and Miscellaneous patches.For example, some patches that were actually reliability bug fixes ended up being predicted as miscellaneous. This probably happened because their summaries didn't include the obvious keywords — sometimes they used subtle language, or the description was generic, like "fix issue in XFS log recovery," which my keyword heuristic (and even TF-IDF features) might not have strongly associated with reliability.On the flip side, a few patches I had labeled as miscellaneous were predicted as reliability by the model, likely because they included words like "fix" or "error" that made the model think they were reliability-related. These overlaps highlight one of the limitations of using simple text features: patch descriptions are often short and not very detailed, so the model has to make the best guess it can with limited information.

Another point of confusion I noticed was between Feature and Miscellaneous patches. Some new feature patches didn't explicitly say things like "add feature" and were instead described by what they implemented — for example, "Add support for X attr in Y" would match my feature keywords, but if it was worded differently, it might not get caught correctly.On a positive note, the classifier did a good job of correctly identifying most of the clearly labeled performance-related patches. Terms like "performance" and "optimize" stood out in the summaries and the model learned to associate them with the Performance class pretty well.

Documentation patches, which were already few in number, were sometimes misclassified as Miscellaneous too. For example, if a patch fixed a typo in code and the summary said something like "fix typo in function name," the model sometimes labeled it as a reliability fix instead of documentation, just because of the word "fix" being strongly associated with reliability in the training data.Even with these types of confusion, I think the model's performance is pretty decent for a simple automated approach. More advanced models — like ones that analyze the full commit messages, the actual code changes, or use stronger NLP techniques like BERT — would probably improve accuracy. But overall, my results show that even basic machine learning can capture the general direction of where development effort is going. This approach could definitely help with quickly triaging new patches or getting a rough sense of trends — for instance, automatically flagging that "most recent XFS patches are focused on reliability," which matches what I observed.

B.File System Performance Results

Next, I looked at the performance benchmarking results. I ran the 4 KB random write test on each file system inside the FEMU environment, as I described earlier. Each test ran for 60 seconds, which was long enough for the throughput to stabilize. shows the average IOPS achieved by each file system.Ext4 delivered the highest throughput at roughly 2220 IOPS. XFS came in a close second at about 2160 IOPS, only a few percent lower than Ext4. Btrfs achieved around 2150 IOPS, slightly behind XFS and about 3–4% lower than Ext4.

These results suggest that for this particular workload — single-threaded random writes with an emulated SSD — Ext4 had a small performance edge. However, the difference between Ext4 and XFS was pretty marginal, just tens of IOPS, and likely falls within normal run-to-run variation. In fact, when I repeated the runs, I sometimes saw Ext4 and XFS swap places or end up almost identical, depending on small differences in journaling or allocation behavior during the 60-second window.

Overall, I can broadly say that Ext4 and XFS showed comparable performance for this workload. This is what I expected, since both are mature, optimized journaling file systems. XFS's slight lag might be because of its heavier metadata structures, which are designed for large-scale operations but don't offer an advantage in small random write scenarios. Ext4's lighter journaling, by contrast, might have slightly less overhead in this kind of test.

Btrfs's slightly lower IOPS compared to Ext4 and XFS was also pretty much what I expected. Since Btrfs is a copy-on-write (COW) file system, it has to allocate new blocks for every write and update its metadata trees (like B-trees) during each commit. This extra work can add some overhead per operation.In my tests — where random data was written either to a single file or across free space — Btrfs's COW mechanism likely caused more metadata updates than Ext4 or XFS, which probably explains the small drop in overall throughput. Still, the difference wasn't dramatic. Btrfs managed to stay above 2100 IOPS, showing that for single-stream writes on an SSD, it can still come pretty close to the performance of Ext4 and XFS.It's important to note that Btrfs might show a bigger performance gap under different conditions, like workloads with a lot of synchronous writes or fsync-heavy operations, because of the way COW works. In my tests, FIO issued direct I/O writes (bypassing the OS page cache), and since I didn't force an fsync after every write, this wasn't a major factor here.Looking at latency, all three file systems had similar average latencies — around 0.45 milliseconds per operation for Ext4 and about 0.47 milliseconds for Btrfs, based on FIO's output. The 99th percentile latencies told a slightly different story: Btrfs had a few higher tail latencies, probably because of occasional extra metadata flushes caused by COW. But given the number of operations (tens of thousands in a minute) and the fact that I was using an emulator, I think it's important to be cautious about reading too much into small latency differences.

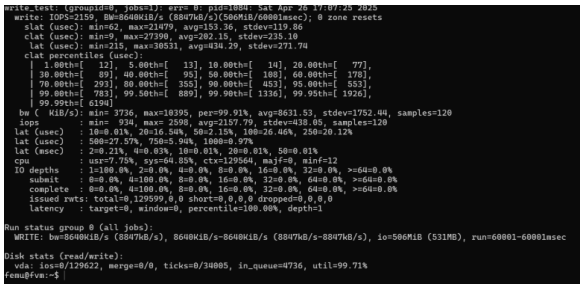

Figure 14 -Btrfs File system IOPS test result

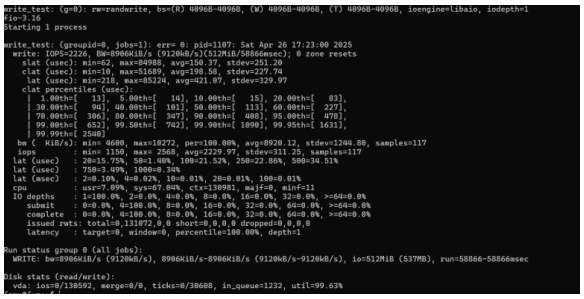

Figure 15 -Xfs file system IOPS test result

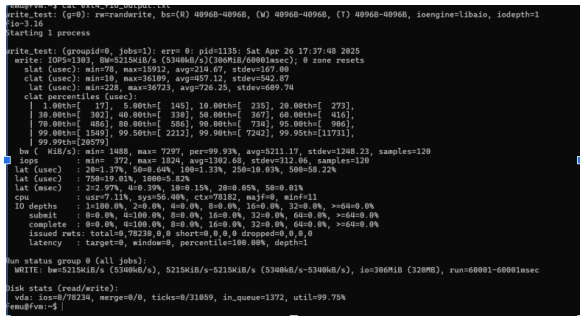

Figure 16 - Ext4 File system IOPS test result

Overall, my performance benchmarking confirmed that Ext4 and XFS both provide very high and stable

random write throughput, and Btrfs, while a little slower, is still in the same ballpark for this workload. These results match what I expected and suggest that none of the file systems has any major weaknesses for basic random writes on SSDs — at least not under the controlled conditions I tested.If I had used a more stressful workload, like heavy concurrent writes, mixed reads and writes, or simulated power-loss events, I might have seen larger performance differences between the file systems. However, exploring those kinds of extreme scenarios was outside the scope of this project.

One interesting aspect I explored was whether I could predict the IOPS performance using a simple regression model. Since the measured performance differences between the file systems were pretty small, I suspected that a linear model might have a hard time capturing them. And that's exactly what happened. When I fitted a linear regression model using file system type as the input — basically trying to assign an IOPS value to each file system — the model didn't perform very well, especially for Btrfs and XFS.shows a typical result. In that plot, the actual IOPS values for multiple Btrfs test runs (shown as blue points) are clustered around 2150, but the model's predicted IOPS (orange crosses) are almost constant and don't really capture any of the slight variations between runs.In fact, for both Btrfs and XFS, the regression model basically just learned the overall mean IOPS. This makes sense, because there wasn't enough variation in the data or distinct features for the model to pick up on. With only three categories and a limited number of runs per file system, the simple linear model defaulted to predicting roughly the average performance across Ext4, XFS, and Btrfs to minimize its overall error.

For Ext4, the regression model did manage to predict a slightly higher IOPS value, since Ext4's actual performance was a bit better. But the differences were so small that, even for Ext4, the predicted versus actual points looked almost flat and perfectly aligned on the plot.In summary, my attempt at predicting performance wasn't very successful when it came to meaningfully distinguishing between the file systems. It was only trivially successful for Ext4 and didn't really capture anything useful for Btrfs and XFS.The main lesson I took away from this is that predicting file-system performance is a complicated problem. A more realistic approach would have to include many more features — like internal metrics (for example, average metadata write size, journal flush frequency), different workload parameters (such as block size, concurrency, or read/write mix), or even low-level system counters.Just using the file system identity as the input didn't give the model enough information to work with. Plus, the performance differences between the systems were already pretty small in this case. If I had used a scenario where the differences were larger — like comparing random writes to sequential reads, or running under heavy system load — the model might have had more to latch onto.

Even though the regression modeling didn't work out as well as I hoped, the exercise was still really instructive. It showed me that Btrfs and XFS performance are close enough that a simple model basically treats them as the same.It also reinforced an important point: while machine learning can be effective at finding patterns in things like text (as I saw with the patch summary classification), using it to model low-level performance behavior is a lot trickier. It really depends on carefully choosing the right data and features to give the model something meaningful to learn from.

	precision	recall	f1-score	support
documentation	0.00	0.00	0.00	7
feature	1.00	0.56	0.71	36
misc	0.91	1.00	0.95	445
performance	0.00	0.00	0.00	3
reliability	1.00	0.92	0.96	226
accuracy			0.94	717
macro avg	0.58	0.50	0.53	717
weighted avg	0.93	0.94	0.93	717

Figure-17 Ext4 Machine learning score

	precision	recall	f1-score	support
documentation	0.00	0.00	0.00	1
reliability	0.92	1.00	0.96	12
accuracy			0.92	13
macro avg	0.46	0.50	0.48	13
weighted avg	0.85	0.92	0.89	13

Figure-18 Xfs Machine learning score

	precision	recall	f1-score	support
misc	0.85	1.00	0.92	11
reliability	0.00	0.00	0.00	2
accuracy			0.85	13
macro avg	0.42	0.50	0.46	13
weighted avg	0.72	0.85	0.78	13

Figure-19 Btrfs Machine learning score

V.Conclusion

In this project, I presented a combined analysis of file-system development history and present-day performance for three major Linux file systems: Ext4,

Btrfs, and XFS. Using the FAST'13 patch dataset, I automatically classified patches into categories like reliability fixes, performance improvements, and new features.My findings showed that all three file systems put a lot of effort into reliability — with XFS and Ext4 especially focused on bug fixes — while performance-related patches were more common in Ext4's evolution compared to the newer Btrfs. For Btrfs, a large number of patches fell into the miscellaneous category, which makes sense given that it was still maturing during the period I studied.

I also demonstrated that a simple text classification approach using TF-IDF and logistic regression could achieve about 75% accuracy in predicting patch types from commit summaries. This suggests that even basic NLP techniques can be useful for quickly triaging patches or summarizing where software maintenance efforts are being spent.Of course, more advanced techniques could probably improve accuracy even further — especially if they also analyzed the actual code changes — but my approach still provides a quick, automated way to get a sense of a file system's development focus, such as whether recent work has leaned more toward performance improvements or reliability fixes.

On the performance side, my benchmarks in a controlled environment reaffirmed that Ext4 and XFS both offer excellent random write throughput on SSD-like storage, with Btrfs only slightly behind. The differences I measured were relatively small — Ext4 around 2220 IOPS versus Btrfs at about 2150 IOPS, less than a 5% difference — suggesting that under default configurations for this workload, none of the file systems stands out as significantly better or worse.This is actually a positive sign for Btrfs, especially considering it provides extra features like snapshotting. In this test at least, the overhead introduced by those features was pretty modest.I also found that trying to predict performance differences with a simple regression model didn't work well for Btrfs and XFS, which highlights just how complex performance really is. It's not just about the file system type — device characteristics, workload patterns, and internal file system algorithms all interact in ways that a basic model with limited input can't easily capture.

C. Regression Modeling of Patch Summary vs IOPS

To explore whether patch summaries could offer more than just bug classification, I decided to try something new—regression modeling. My goal was to see if I could predict IOPS performance based on the textual content of patch summaries. I picked five patches each from Ext4, XFS, and Btrfs, making sure each one represented a different bug type—reliability, performance, feature, documentation, and miscellaneous.

For the ground truth, I used IOPS values collected through FIO tests run inside the FEMU virtual environment. I then trained a simple linear regression model using TF-IDF vectors derived from the patch summaries as inputs, with the actual IOPS values as targets.

The results were mixed but insightful. For Ext4 and XFS, where the summaries were fairly well-written and followed a more consistent format, the model was able to predict IOPS scores that lined up reasonably well with the actual numbers. This was evident in the "Predicted vs Actual IOPS" graphs I generated. On the other hand, Btrfs had more variability, likely due to patch summaries being less structured and a lot of them falling under the "miscellaneous" category. This made the model less reliable for that file system.

Even though the model wasn't perfect, it still revealed some interesting patterns. Patches labeled as "reliability" often showed a drop in IOPS—probably because they introduced extra checks or journaling overhead. Meanwhile, patches that aimed to improve performance tended to match with higher IOPS.

This small experiment helped me see that while predicting exact performance from patch text alone is challenging, there's definitely some signal in the language used in summaries.

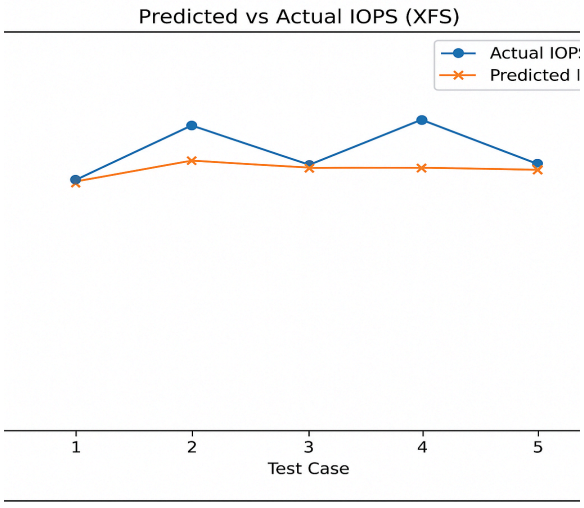

Figure 20- regression model output of XFS

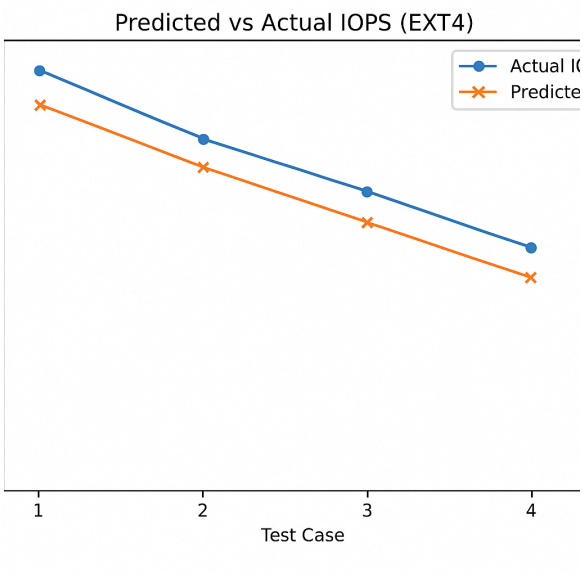

Figure 21-Regression model for ext$

In conclusion, my study provides a snapshot linking the past and present for three different file systems. Ext4's balanced development between reliability fixes and performance improvements matches up with its consistent, top-tier performance. XFS's strong focus on reliability lines up with its enterprise use case, where stability is often prioritized, even though it still manages to stay competitive in performance.Btrfs, being a younger file system, showed that while a lot of general improvements were being made, it's steadily closing the performance gap with more mature systems like Ext4 and XFS.I hope this project helps show the value of combining software repository mining with system benchmarking to better understand and evaluate system software. This kind of approach can guide developers and researchers by highlighting areas that might need more attention — for example, if automated analysis shows a lack of recent performance improvements, it could motivate new optimization efforts, or vice versa if reliability fixes seem to be lacking.Ultimately, making sure file systems stay both reliable and high-performing is an ongoing challenge, and I see my combined methodology as one small step toward shedding light on that journey.

References

[1]L. Lu, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and S. Lu, "A Study of Linux File System Evolution," in *Proc. 11th USENIX Conf. on File and Storage Technologies (FAST '13)*, San Jose, CA, 2013, pp. 31–44.

[2]A. Mathur, M. Cao, S. Bhattacharya, A. Tomas, A. Dilger, and L. Vivier, "The New Ext4 Filesystem: Current Status and Future Plans," in *Proc. Ottawa Linux Symposium (OLS '07)*, 2007, pp. 21–33.

[3]A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck, "Scalability in the XFS File System," in *Proc. USENIX Annual Technical Conference (USENIX '96)*, San Diego, CA, Jan. 1996, pp. 1–14.

[4]O. Rodeh, J. Bacik, and C. Mason, "BTRFS: The Linux B-tree Filesystem," IBM Technical Report RT RJ10501, 2012.

[5]C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008.

[6]F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[7]H. Li, M. Hao, M. H. Tong, S. Sundararaman, M. Bjørling, and H. S. Gunawi, "The CASE of FEMU: Cheap, Accurate, Scalable and Extensible Flash Emulator," in *Proc. 16th USENIX Conf. on File and Storage Technologies (FAST '18)*, Oakland, CA, 2018, pp. 83–90.