June 2023

OS-Performance Profiling & IO-Scheduling

CS350 Project

Ezgi Nur Alışan
Emir Mehmet Eryılmaz
Alkın Korkut

Introduction

I/O scheduling is a technique used by operating systems to manage the order in which input/output (I/O) requests are serviced by storage devices. It aims to improve the overall efficiency and performance of a computer system. Linux being an open-source operating system allows developers to directly make additions to the kernel. One of the challenges in accessing hard disk drives is the long access times, particularly for requests that are far from the current position of the disk head, known as seek operations. These seek operations negatively impact system performance by taking a significant amount of time. Efficient I/O scheduler algorithms are necessary to optimize and overcome these issues, which involve techniques like merging and sorting operations to reduce the number of seeks. However, different algorithms rearrange incoming block requests in varying ways, leading to variations in system performance. The performance of these schedulers also relies on the specific set of requests they receive.

Background Research

Many methods have been used in the optimization of I/O scheduling algorithms. Nou et al. (2012) introduce a tool named IOAnalyzer that gathers traces and statistics from the I/O stack. By using Dynamic Time Warping algorithm, traces are compared. [4] This model determines the most suitable I/O planner based on workload characteristics using clustering or regression algorithms. González-Férez et al. (2014) present a general Dynamic and Automatic Disk Scheduling framework (DADS) for hard and solid-state drives that can select the best scheduler. [5] In-kernel disk simulator which they developed is used for selecting the optimal I/O scheduler. It compares two schedulers (AS and CFQ) and chooses the best one. According to those studies, it has been revealed that there is a need for an application that chooses the optimal I/O scheduler, and this need can be met in different ways. In our study, an application will be developed by considering these examples.

I/O Schedulers

There are different approaches taken for various I/O schedulers. Each one has its own advantages and disadvantages. Also, I/O schedulers vary in terms of different concepts such as timing, merging, and sorting and I/O priority. The information about the I/O schedulers is thoroughly examined below.

Non-Multiqueue Schedulers

1) Complete Fair Queuing (CFQ)

The Complete Fair Queuing I/O scheduler algorithm is designed to provide fairness and reasonable performance. Since the main goal of the CFQ algorithm is to provide fairness, it ensures that no single process dominates all I/O resources. To do this, I/O time slices are allocated for each synchronous process in a round-robin fashion. In this way each process can perform I/O operations between their own allocated time slices. When dealing with asynchronous requests, the CFQ scheduler groups together I/O requests from various processes based on their respective I/O priorities. The CFQ scheduler divides processes into three distinct classes: real-time (highest priority), best effort, and idle (lowest priority). Higher priority is given to synchronous requests, which are typically linked to interactive processes, and they are processed ahead of asynchronous requests.

2) Deadline

The deadline scheduler aims to minimize the latency of I/O operations by striving to ensure a specific start time for each request. The deadline algorithm prevents I/O requests from being neglected by imposing specific time limits, or deadlines, on each operation. It manages read and write requests separately by using a FIFO queue for each type of operation, sorting them based on their respective deadlines. Whenever a new request arrives, the scheduler determines the appropriate queue to place it in. Read queues are prioritized over write queues since processes often wait for read operations to be completed.

3) Noop

The No Operation (NOOP) scheduler is a type of scheduler that takes incoming I/O requests and places them in a basic FIFO queue regardless of their operation type (read, write etc.). It also combines or merges similar requests together. This scheduler is beneficial when it is determined that the computer should not attempt to rearrange the requests based on the sector numbers they contain. Essentially, the scheduler assumes that the computer system doesn't know how to effectively reorganize the requests. The NOOP scheduler is particularly suitable for random access devices like flash drives and ramdisks. It is also useful for devices that handle I/O requests in a sorted manner, such as advanced storage controllers.

Multiqueue Schedulers

1) Budget Fair Queuing (BFQ)

Budget Fair Queuing algorithm aims to provide fairness by allocating separate budgets to different I/O requests. BFQ, the I/O scheduler, works differently than allocating a fixed time-slice to each process for disk access. Instead, it assigns a "budget" to each process, measured in sectors, using heuristics. It also associates weights with processes or groups, allowing them to receive a proportionate share of the available I/O bandwidth based on their weight. The primary objective of BFQ is to optimize system responsiveness and reduce latency for time-sensitive applications, ensuring better performance and responsiveness.

2) Kyber

Kyber is a modern scheduler inspired by network routing techniques. It uses tokens to control and limit I/O requests, preventing starvation. A queuing token is needed to allocate a request, while a dispatch token restricts operations based on priority. Kyber aims to reach a target read latency and adjusts itself accordingly. It is a straightforward and efficient algorithm, especially for fast devices. It has two request queues, accommodating both synchronous requests such as blocked reads, and asynchronous requests such as writes. Kyber can be used efficiently for high performance storage like SSDs and NVMe drives.

3) Mq-Deadline

Mq-deadline is a modified version of the deadline I/O scheduler specifically created for Multiqueue devices. Its main objective is to group pending I/O requests into batches, organizing them based on logical block addressing. This arrangement ensures that write operations are efficiently managed and stored according to their respective locations. Mq-deadline is a well-balanced scheduler that offers a good overall performance with minimal impact on CPU resources.

4) None

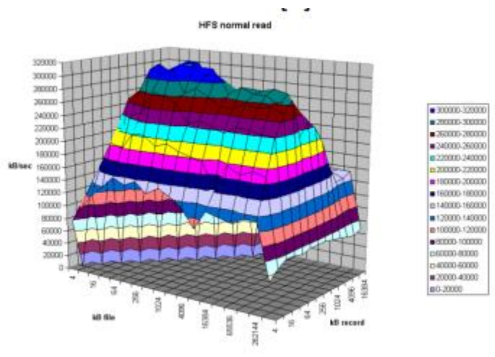
None is the multi-queue version of the no-op I/O scheduler, is a scheduler that follows a first-in first-out (FIFO) approach. It is particularly suitable for systems equipped with high-performance storage technologies like Solid State Drives (SSD) or Non-volatile Memory Express (NVMe) drives. In fact, the none scheduler is the recommended default for NVMe devices. It does not perform any reordering of I/O requests, resulting in minimal overhead. This scheduler is well-suited for fast random I/O devices like NVMe, ensuring optimal performance without additional scheduling complexities.

Experimental Analysis

We included 4 different I/O schedulers in the experiments. These are None, Bfq, Mq-Deadline and Kyber. We chose these schedulers because among the I/O schedulers mentioned above only these four were available in Ubuntu 22.04.

Tool

Iozone tool is used for the experiments. It generates an Excel compatible report to standard out. It facilitates the graphing of file system performance. It examines I/O performance in terms of many operations: Read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read, pread, mmap, aio_read, aio_write.



Platform

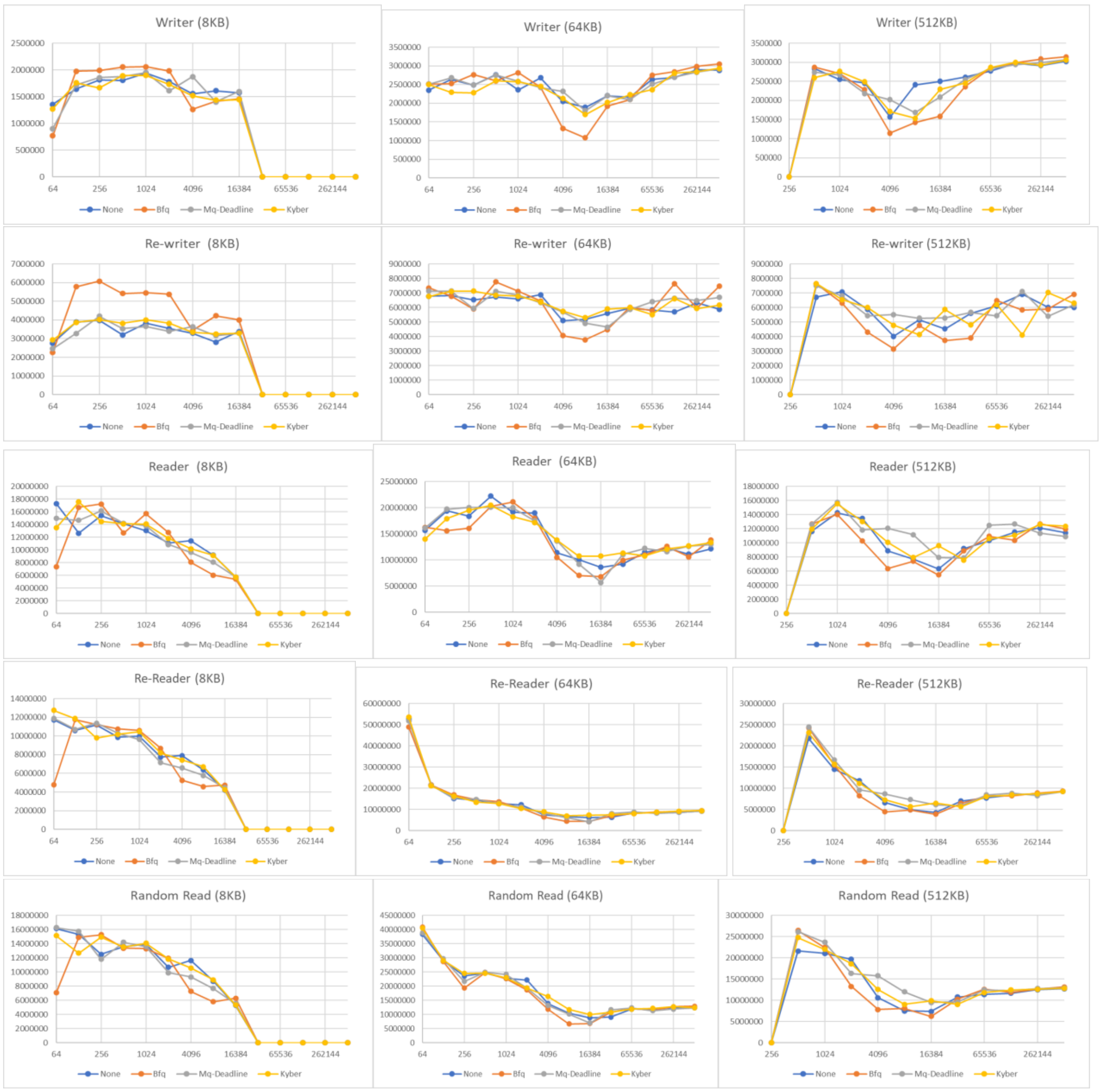
The experiments were carried out on Linux environment. The CPU that we will use has 6 cores and 2 threads per core, 2.60GHz Intel i7-10750H processor system, with total 15814 MB memory, 32kB L1 cache, 256kB L2 cache, 12MB L3 cache and 512GB SSD. The operating system and version that we will use is Ubuntu 22.04.2 LTS.

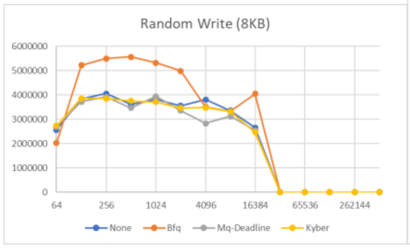
Experiment Results

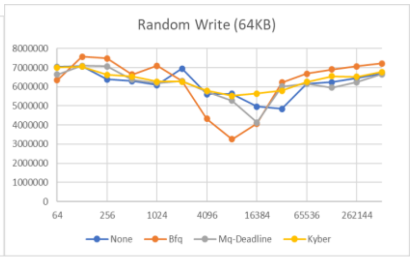
During the analysis of I/O schedulers, six primary operations are used: Read, write, re-read, re-write, random read and random write.

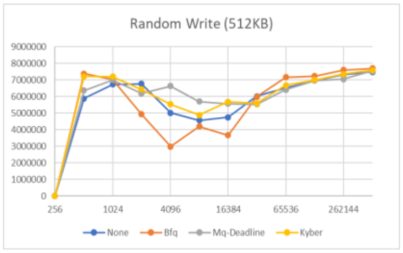
- Write operation occurs while storing data from the memory to storage device. Data is written in place of existing data or empty space.
- Read operation occurs while data is retrieved from storage device and transferred to the memory for processing purposes.
- Re-read occurs while accessing previously read data from the storage device multiple times. The data that is read has been read before.
- Re-write occurs while modifying existing data on the storage device. New content is overwritten in place of existing data.
- Random read occurs while accessing data from random locations (not sequentially) on the storage device. It retrieves data from different locations.
- Random write occurs while storing data to random locations on the storage device without any order (not sequentially). Data is written to different locations.

Each operation was conducted using each scheduler 10 times. Throughput results are obtained by averaging these 10 generated results. The results were grouped by the record-length and file size parameters. In the results, the record length was between 4KB and 16MB, while the file sizes were between 64KB and 512MB. Output (throughput) is in kBytes/sec. For the record lengths 8KB, 64KB and 512KB are chosen. Measurements were made for three different record lengths from the recording lengths. The ones we selected are 8KB, 64KB and 512KB.









The graphs given above show the average throughputs for each different I/O operation and record lengths according to file sizes. The average throughputs of 4 different I/O schedulers' are indicated in different colors in each graph. As can be seen from the graphs, although there are intervals in which some algorithms show better results in some cases, in general, it is observed that all I/O schedulers show similar results. Therefore, to understand the difference between performance of the I/O schedulers better, we divided graphs into two parts by file sizes. We calculated the average throughput of each algorithm for 2 different file sizes ranges. We chose the I/O scheduler that has maximum throughput for each range in different record lengths. Highlighted cells represent in which the maximum throughput is achieved. The calculations are shown below.

READER OPERATION						
Ranges	size <= 1MB	1MB < size <= 32MB	size <= 4MB	4MB < size <= 512MB	512 size <= 16MB	size <= 512MB
Record length	8KB	8KB	64KB	64KB	512	512
None	14503511.3	9356080.85	17881767.19	10696302.74	11187718.32	10159278.4
BFQ	13941275.62	8063292.7	16820698.64	10244587.87	10149306.06	10051256.62
MQ	14756119.02	8574253.7	18163622.87	10774615.84	12692983.38	10543751
KYBER	14739764.86	9217675.75	17298957.96	11680073.07	11701922.66	10626645.8

RE-READER OPERATION						
Ranges	size <= 1MB	1MB < size <= 32MB	size <= 4MB	4MB < size <= 512MB	512 size <= 16MB	size <= 512MB
Record length	8KB	8KB	64KB	64KB	512	512
None	10676770.3	6561341.05	19485714.03	7703936.7	11905709.48	7562687.5
BFQ	9829810.54	5818132	18943349.57	7274502.257	11459609.56	7475129.317
MQ	10784599.96	5995830.975	19404820.31	7612834.1	13357740.12	7793566.567
KYBER	11012014.4	6650273.125	19524041	8182791.257	12563084.92	7746899.567

WRITER OPERATION						
Ranges	size <= 1MB	1MB < size <= 32MB	size <= 4MB	4MB < size <= 512MB	512 size <= 16MB	size <= 512MB
Record length	8KB	8KB	64KB	64KB	512	512
None	1713129.04	1631693.2	2478171.943	2483768.129	2364079.08	2801717.7
BFQ	1770767.52	1529576.125	2426249.329	2390104.2	2087649.4	2669166.65
MQ	1660784.24	1622015.075	2538078.029	2440989.986	2260120.92	2746840.217
KYBER	1698892.86	1533101.325	2405312.571	2414374.129	2218805.58	2766276.35

RE-WRITER OPERATION						
Ranges	size <= 1MB	1MB < size <= 32MB	size <= 4MB	4MB < size <= 512MB	512 size <= 16MB	size <= 512MB
Record length	8KB	8KB	64KB	64KB	512	512
None	3527889.86	3257546.175	6497182.1	5788075.829	5761009.3	5865597.383
BFQ	5002097.14	4262003.725	6487983	5877804.629	5236163.74	5460297.65
MQ	3426236.8	3375853.225	6599763.329	5951465.157	6102126.22	5847431.783
KYBER	3729806.28	3427843.5	6681507.829	5914932.3	5820938.48	5719326.017

RANDOM READ OPERATION						
Ranges	size <= 1MB	1MB < size <= 32MB	size <= 4MB	4MB < size <= 512MB	512 size <= 16MB	size <= 512MB
Record length	8KB	8KB	64KB	64KB	512	512
None	14295949.38	9064159.875	24871649.69	10977917.39	16087510.22	11070983.77
BFQ	12774443.18	7823985.175	23844176.61	10498730.56	15618807.18	11126697.77
MQ	14326233.28	8091183.075	24540870.6	10951372.06	18780363.44	11594404.3
KYBER	14065322.68	9148857.075	25319294.4	11676036.34	17394758.4	11435155.65

RANDOM WRITE OPERATION						
Ranges	size <= 1MB	1MB < size <= 32MB	size <= 4MB	4MB < size <= 512MB	512 size <= 16MB	size <= 512MB
Record length	8KB	8KB	64KB	64KB	512	512
None	3574343.38	3344708.375	6492738.929	5851098.743	5789776.1	6512775.6
BFQ	4727801.26	3963536.2	6537288.529	5917486.329	5306057.38	6558533.35
MQ	3550345.4	2966500.7	6487882.057	5775710.443	6372576.4	6513852.233
KYBER	3581854.64	3184457.8	6508314.229	6150650.886	6252914.64	6644164.15

Implementation

As we observed from the test results, different schedulers perform differently depending on file sizes and record-length parameters. Therefore, by selecting only one I/O scheduler algorithm, the performance that can be achieved is limited. Based on this, we decided to focus on writing a program that changes the I/O schedulers dynamically depending on file sizes, record-lengths and operation types.

The program that we implemented decides which I/O scheduler it should choose based on the intervals that are shown in previous tables and select the one that achieves maximum throughput.

```python
def select_scheduler(operation, file_size, rec_length):
    scheduler = ""
    if (operation == "write"):
        if (rec_length == 8):
            if (file_size <= 1024*1024):
                scheduler = BFQ
            else:
                scheduler = NONE
```

For instance, in the code above when the operation type is write, rec-length is 8 KB and file size is smaller than and equal to 1 MB the scheduler is determined as BFQ. Otherwise when the operation type and rec-length are the same, and file size is greater than 1MB, the I/O scheduler is determined as none. We have implemented if-else blocks like this for each highlighted cell in the tables above.

```python
def set_scheduler(scheduler_):
    system('echo %s | sudo tee /sys/block/nvme0n1/queue/scheduler' %
scheduler_)
    system('cat /sys/block/nvme0n1/queue/scheduler')

    print("I have choosen " + scheduler_)
```

The way we change schedulers automatically can be seen in the above code. The system function is coming from the os module in Python. This function allows executing commands in the underlying operating system's shell. First shell command is used for switching the I/O schedulers that are available in the operating system. The second command is used for printing the selected I/O scheduler.

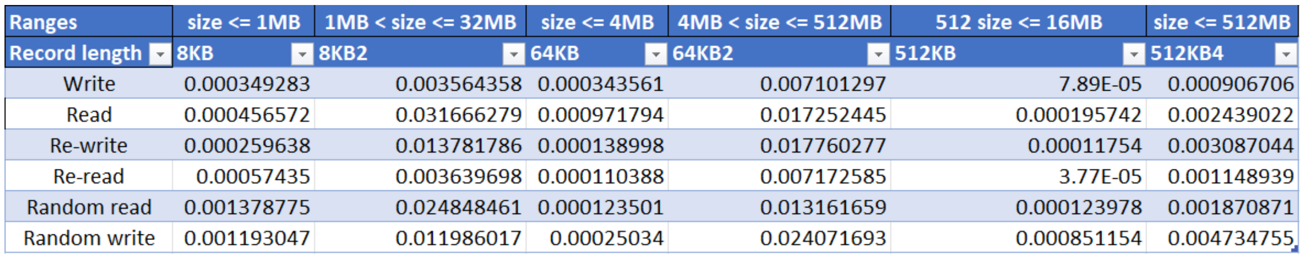
```
set_scheduler(scheduler)
```

Set_scheduler function is being called at the end of the select_scheduler function with the paramater which is the scheduler that has been determined in the select_scheduler function.

To form some test cases we designed and implemented a file creation function by filtering with the parameters, rec-length and file sizes. We created some functions that simulates the specified six operations (read, write, re-read, re-write, random read, random write). These functions returns latency and throughput of the related operation.

Evaluation of our program results

We have a total of 36 different cases to test. Each test case generates a result which involves the selected scheduler for the operation and latency of the operation. While we got good results with low file sizes, we got some inconsistent results with high file sizes. The test results are shown in below.



Ranges	size <= 1MB	1MB < size <= 32MB	size <= 4MB	4MB < size <= 512MB	512 size <= 16MB	size <= 512MB
Record length	8KB	8KB2	64KB	64KB2	512KB	512KB4
Write	0.000349283	0.003564358	0.000343561	0.007101297	7.89E-05	0.000906706
Read	0.000456572	0.031666279	0.000971794	0.017252445	0.000195742	0.002439022
Re-write	0.000259638	0.013781786	0.000138998	0.017760277	0.00011754	0.003087044
Re-read	0.00057435	0.003639698	0.000110388	0.007172585	3.77E-05	0.001148939
Random read	0.001378775	0.024848461	0.000123501	0.013161659	0.000123978	0.001870871
Random write	0.001193047	0.011986017	0.00025034	0.024071693	0.000851154	0.004734755

Use Cases of Different File Sizes

There are various applications with different file sizes that perform file operations. In this study, since we determined the file size ranges according to certain record lengths, we selected the smallest file size bound 1MB and the largest bound 512MB from 1MB, 4MB,16MB, 32MB and 512MB file sizes to show the use cases of those and investigated which applications the operations of file sizes are used. Each one is described below.

Less than 1MB:
Text editors and Note-taking Apps: Those applications are similar applications that allow users to create, edit and save text-based files and notes. Write operation is performed when the user creates a new note or text-file and saves this text. Read operations includes opening and displaying existing files. Re-read and re-write operations occur when user modify the file and saves it multiple times. Random read and random write can be performed when reading and writing specific positions.

Database Systems: Database applications perform all those file operations. It writes new records into database file and reads data from the database. When updating the existing records in database re-read and re-write occurs. For specified records of fields in database, random read and random write can happen.

File Compression/Archiving Tools: These applications are used to compress or archive the file. Writing occurs when creating a compressed file, while reading occurs when extracting and reading files. Other operations can also be used.

Less than 512MB:

Document Editors: Applications such as Word and Google Docs can be used for file sizes less than 512MB. Users can create, edit, and save documents on those platforms. Write operations involves entering and saving texts or other contents like images, read operations involves opening or viewing the contents of existing documents. When the user makes changes to the documents and saves them multiple times re-read and re-write operations occur. For the accessing and operating specific sections random read and random write can occur.

Media Players: Media player applications use audio and video files. Write operations may be used for creating media contents, such as playlists, read operations may be used for playing media content. When modifying the metadata, re-read and re-write operations may occur. Navigating specific sections of a media file, random read and random write operations can be used.

Photo Editing Software: In applications such as Adobe Photoshop, users can edit and save images. File operations can occur when making changes to images and saving them or opening and displaying images, modifying, and saving images multiple times and accessing specific image regions.

Virtual Machines: The software that users can use to run virtual operating systems is virtual machines. File operations can happen in different situations. Writing occurs when virtual disk images, configuration files are created, reading occurs when the virtual machine is booted or interacted. Re-read and re-write happen when making changes to the virtual machine state.

Conclusion

In conclusion, I/O scheduling plays a crucial role in optimizing the performance and efficiency of computer systems by managing the order of I/O requests. Different I/O scheduler algorithms, such as CFQ, Deadline, Noop, BFQ, Kyber, Mq-Deadline, and None, offer various approaches to address seek time and improve system responsiveness. Through experimental analysis using the Iozone tool on a Linux platform, we observed that different schedulers perform differently based on file sizes and record lengths, but overall, their performances are relatively similar. Instead of using a single I/O scheduler, we created a program that can dynamically choose the optimal I/O scheduler depending on the type of operation, file size, and record length. The program automatically switches schedulers using system commands and provides flexibility in adapting to different scenarios. With this approach, we think that performance of different applications can be improved. By dynamically selecting the most suitable I/O scheduler, efficiency of I/O operations can be enhanced, resulting in improved overall performance.

References

[1] D. Dai, Y. Chen, D. Kimpe and R. Ross, "Two-Choice Randomized Dynamic I/O Scheduler for Object Storage Systems," SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, New Orleans, LA, USA, 2014, pp. 635-646, doi: 10.1109/SC.2014.57.

[2] Pilar González-Férez, Juan Piernas, Toni Cortes, A general framework for dynamic and automatic I/O scheduling in hard and solid-state drives, Journal of Parallel and Distributed Computing, Volume 74, Issue 5, 2014, Pages 2380-2391, ISSN 0743-7315, https://doi.org/10.1016/j.jpdc.2014.02.002.

[3] R. Nou, J. Giralt and T. Cortes, "Automatic I/O Scheduler Selection through Online Workload Analysis," 2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing, Fukuoka, Japan, 2012, pp. 431-438, doi: 10.1109/UIC-ATC.2012.12.

[4] Seelam, S., Romero, R., Teller, P., & Buros, B. (2005, July). Enhancements to linux i/o scheduling. In Proc. of the Linux symposium (Vol. 2, pp. 175-192).

[5] Yunus Ozen, Abdullah Yildirim; (2019), Performance comparison and analysis of Linux block I/O schedulers on SSD. Sakarya University Journal of Science, 23(1), 106-112, DOI: 10.16984/saufenbilder.477446

[6] IOSchedulers. *Ubuntu wiki*. 2023 https://wiki.ubuntu.com/Kernel/Reference/IOSchedulers

[7] *IOzone Filesystem Benchmark.* 2023 https://www.iozone.org/

[8] *Iometer*. 2023 http://www.iometer.org/

[9] I/O Scheduling. *Wikipedia.* 2023 https://en.wikipedia.org/wiki/I/O_scheduling
[10] I/O scheduling in Operating Systems. *GeeksforGeeks*. 2023 https://www.geeksforgeeks.org/deadline-scheduler-in-operating-system/
[11] Red Hat Enterprise Linux: Disk Schedulers. *Dell Technologies*. 2023 https://infohub.delltechnologies.com/l/day-three-best-practices-7/red-hat-enterprise-linux-disk-schedulers-3#:~:text=Mq%2Ddeadline%3A%20This%20scheduler%20groups,latency%20rather%20than%20maximum%20throughput.