

Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.Б15-мм

Разработка утилиты для тестирования
производительности RAID-массивов
пространства ядра и пространства
пользователя

Павлушкин Дмитрий Владимирович

Отчёт по курсовой работе
в форме «Решение»

Научный руководитель:
Инженер-исследователь лаборатории технологий программирования
инфраструктурных решений СПбГУ А. И. Васенина

Санкт-Петербург
2024

Оглавление

Введение	3
1. Постановка задачи	5
1.1. Осенний семестр	5
1.2. Весенний семестр	5
2. Обзор	6
2.1. Утилиты для тестирования производительности блочных устройств	6
2.1.1. FIO	6
2.1.2. FIO-plot	6
2.1.3. FIO скрипт	6
2.2. Реализации RAID-массивов	7
2.2.1. Mdadm	7
2.2.2. SPDK RAID	7
3. Реализация	8
3.1. Архитектура	9
3.2. Детали реализации	10
3.2.1. Парсинг конфигурационного файла	10
3.2.2. Инициализация RAID	11
3.2.3. Работа с FIO	12
3.2.4. Остановка RAID	13
3.2.5. Генерация графиков	13
4. Эксперимент	15
4.1. Подготовка и тестирование	15
4.2. Анализ результатов	17
Заключение	18
Список литературы	19

Введение

Системы хранения данных — это сложно организованные блочные¹ устройства, которые повсеместно применяются в компьютерных системах и серверах для хранения и обработки больших объёмов данных. Тема производительности является важной частью любой системы хранения данных. Одной из ключевых технологий, обеспечивающей надёжность, доступность и производительность систем хранения, являются RAID²-массивы, использующие контрольно-восстановительные суммы.

Технологию RAID-массивов можно реализовать не только аппаратно, но и программно. Программные реализации очень удобны в обновлении и сопровождении. Также в программных реализациях можно использовать оптимизации, недоступные аппаратно.

Программные реализации могут работать из пространства ядра и из пространства пользователя. Реализации RAID-массивов работающих из пространства пользователя имеют ряд преимуществ над реализациями из пространства ядра. Перемещение необходимых драйверов работы с RAID-массивами в пространство пользователя позволяет избегать системных вызовов. Также не приходится выполнять операцию копирования информации в запросах из пространства ядра в пространство пользователя.

Компания Intel разработала фреймворк SPDK [12], позволяющий разрабатывать высокопроизводительные приложения для хранения данных в пространстве пользователя. Высокая производительность достигается за счёт избегания любых блокировок на пути ввода-вывода и опроса аппаратных средств для завершения работы вместо вызова прерываний, что снижает общую задержку запросов ввода-вывода.

Разработчики фреймворка провели множество тестов производительности, сравнивая между собой RAID-массивы, разработанные с помощью фреймворка SPDK, с их аналогами из пространства ядра. Резюмируя тесты производительности компании Intel, их реализации

¹Устройство хранения данных, работающее с данными по блокам фиксированного размера.

²Redundant Array of Independent Devices.

RAID-массивов, написанные с помощью SPDK, делают работу систем хранения данных быстрее.

Проведение тестов производительности блочных устройств долгая и трудоёмкая задача. Сравнение производительности двух и более реализаций RAID-массивов требует большого количества времени на правильную инициализацию и корректное освобождение дисков от неё. Наличие утилиты автоматизации, которая позволяла бы разработчикам с лёгкостью протестировать разные реализации RAID-массивов, облегчило бы этап тестирования разработчикам собственных RAID-массивов, а также всем пользователям, желающим найти для себя наиболее подходящую технологию и версию RAID-массива.

1 Постановка задачи

Целью данной работы является разработка расширяемой утилиты для поддержки тестирования производительности разных реализаций RAID-массивов из пространства ядра и из пространства пользователя.

Для достижения данной цели были поставлены следующие задачи:

1.1 Осенний семестр

1. Рассмотреть имеющиеся утилиты для запуска тестов производительности.
2. Рассмотреть имеющиеся реализации RAID-массивов из пространства ядра и из пространства пользователя для дальнейшего добавления их поддержки в утилиту.
3. Спроектировать архитектуру утилиты с поддержкой единого конфигурационного файла для удобной работы с любым видом реализации RAID-массивов как из пространства ядра, так и из пространства пользователя.
4. Представить прототип утилиты, которая смогла бы поддерживать реализации RAID-массивов из пространства ядра и из пространства пользователя.

1.2 Весенний семестр

1. Добавить в утилиту поддержку автоматической генерации конфигурационного файла блочного устройства для SPDK.
2. Добавить поддержку графиков пропускной способности и задержки запросов.
3. Сравнить производительность RAID реализации SPDK и Mdadm, в том числе разработанного недавно SPDK RAID5.

2 Обзор

2.1 Утилиты для тестирования производительности блочных устройств

2.1.1 FIO

Реальную нагрузку на блочное устройство очень сложно смоделировать из-за многих факторов: может участвовать разное количество процессов и потоков, каждый из которых может использовать свой способ генерирования ввода-вывода, поэтому инструмент для теста производительности должен быть гибким. Для тестов производительности и моделирования рабочих нагрузок для Linux/Unix систем, удовлетворяющий гибкости и моделированию работы в несколько процессов и потоков, широко используется инструмент — FIO [5].

Однако, множество параметров в конфигурационном файле могут сбить с толку начинающих тестировщиков. Также вывод FIO в виде файла в формате `json` или в стандартный поток вывода в консоль может стать не очень наглядным для презентации производительности RAID-массива.

2.1.2 FIO-plot

Существуют инструменты для дополнительной визуализации. FIO-plot [3] является одним из таких инструментов, позволяющим не только тестировать производительность, но и на основе этих тестов строить графики. В основе FIO-plot встроена работа инструмента FIO.

FIO-plot принимает уже существующее блочное устройство, поэтому вся работа с инициализацией и запуском необходимых технологий переложена на пользователя.

2.1.3 FIO скрипт

В инструменте FIO есть собственный скрипт [4] для отображения в виде графиков производительности блочных устройств. Однако скрипт

больше не работоспособен и последний коммит был больше года назад.

2.2 Реализации RAID-массивов

2.2.1 Mdadm

Mdadm [7] — утилита командой строки для управления программными RAID-массивами в Linux. Повсеместно используется для создания различных версий RAID-массивов из пространства ядра. Обладает широкой функциональностью, например, можно легко сменить версию RAID без потери данных на дисках. Mdadm является стандартным инструментом в Linux и большинство дистрибутивов поставляются с ним.

Для создания RAID-массивов с помощью Mdadm используется консольная команда.

2.2.2 SPDK RAID

RAID массивы, работающие с помощью SPDK, являются сейчас единственными реализациями из пространства пользователя с открытым исходным кодом. На данный момент корректно работает только RAID-0 [11], однако ведётся активная разработка RAID-5, который уже доступен для тестирования.

Для создания RAID-массивов с помощью SPDK может использоваться конфигурационный файл в формате `json`.

3 Реализация

Далее будет описана общая концепция работы утилиты. Важно отметить, что утилита основывается на инструменте fio и без его установки, утилита работать не будет.

Реализация утилиты основывается на трёх основных этапах. (рис 1)

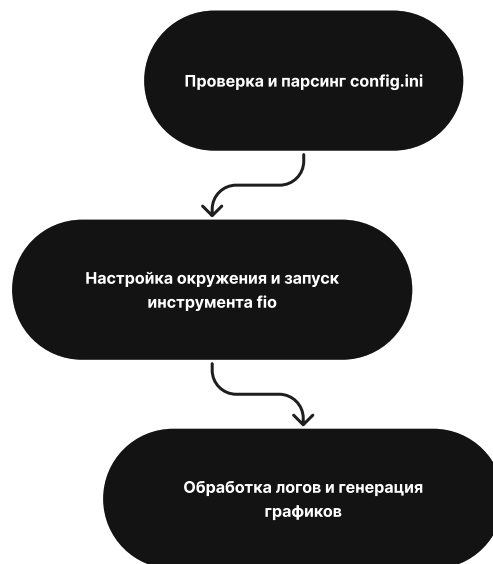


Рис. 1: Схема работы утилиты

В начале выполняются проверки для корректной работы утилиты: проверяется существования инструмента FIO и конфигурационного файла, а также выполняется проверка самих параметров в конфигурационном файле. Далее мы считываем параметры в уникальную структуру похожую на словарь. Следующим шагом будет запуск нужной реализации и версии RAID, указанной в конфигурационном файле. После инициализации RAID-массива генерируется FIO файл, который будет запускаться в инструменте FIO для генерации логов. После этого завершается работа инициализированного RAID-массива и по получившимся логам строятся графики. Схематичная логика работы утилиты представлена на рис. 1.

3.1 Архитектура

При реализации утилиты была создана следующая архитектура (рис 2):

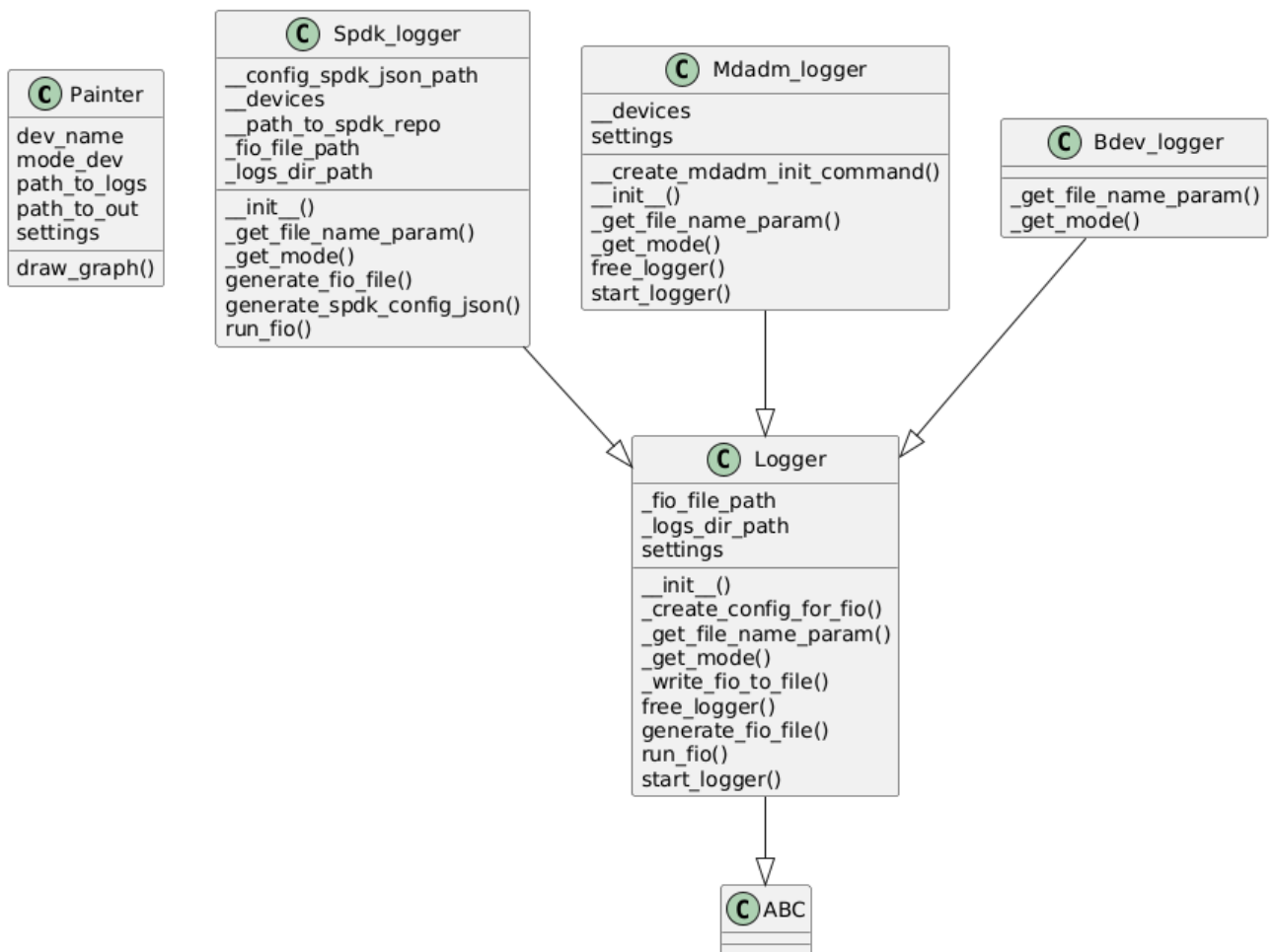


Рис. 2: Диаграмма всех классов утилиты

Как видно из диаграммы, есть абстрактный базовый [1] класс `logger` имплементирующий всю работу с дисковым блочным устройством. Все реализации RAID-массивов, поддерживаемые утилитой, должны наследоваться от него. В этих классах имплементируются только те методы, которые отличаются от методов абстрактного класса `logger`. Логика данных классов будет заключаться в генерации необходимого файла для утилиты `fio` и вывода логов в нужную папку, которую впоследствии будет обрабатывать класс `painter`.

Разработанная архитектура является расширяемой для разработчи-

ков, которые захотят добавить поддержку своей реализации RAID в утилиту. Встраивать свою реализацию RAID можно будет простым наследованием от класса `logger`, в котором нужно будет добавить только отличающиеся от абстрактного класса методы, отвечающие за инициализацию и освобождение ресурсов после работы.

На данный момент в утилите добавлена поддержка одной реализации RAID из пространства ядра — `Mdadm RAID` и одной реализации из пространства пользователя — `SPDK RAID`, также утилита поддерживает работу с любым блочным устройством по отдельности благодаря классу `bdev_logger`.

3.2 Детали реализации

В качестве языка реализации был выбран Python3, как удобный кроссплатформенный инструмент, имеющий множество встроенных библиотек, например для работы с интерпретатором Python3 использовалась библиотека `sys` [14], для взаимодействия с операционной системой библиотека `os` [8], а также библиотека `subprocess` [13] для запуска сторонних процессов в программе.

Для парсинга конфигурационного файла и дальнейшей генерации файла для работы с инструментом FIO была выбрана библиотека `configparser` [2], которая удобно работает со структурой очень похожей на структуру INI файла, использующегося в Microsoft Windows. Данная структура схожа с конфигурационным файлом FIO.

Для отображения графиков был использован набор функций из библиотеки `pyplot` [9].

3.2.1 Парсинг конфигурационного файла

Путь до конфигурационного файла указывается при запуске утилиты и является единственным аргументом для. Конфигурационный файл схож по структуре на формат INI [6], который используется для настроек Microsoft Windows. Сам файл делится на две секции: `[global]` и специфичную для реализации RAID.

В секции под названием `[global]` перечислены все параметры, характерные для запуска любого теста производительности блочного устройства, там перечислены такие параметры как размер блока, время тестирования и другие. Подробнее со всем списком можно ознакомиться в документации к утилите [10].

После секции `[global]`, которая является общей для любой реализации RAID, добавляется вспомогательная секция с информацией для поддерживаемых утилитой реализаций RAID. Сейчас присутствуют секции `[raid]` и `[spdk]` для указания параметров `mdadm` и `spdk` реализаций соответственно (рис 3).

Bdev	Mdadm	SPDK
<pre>[global] ioengine=libaio invalidate=1 ramp_time=5 size=4K iodepth = 32 numjobs = 1 runtime = 120 bs = 4K rw = write, read dev = /dev/sdb</pre>	<pre>[global] ioengine=libaio iodepth = 1 numjobs = 1 rw = write, read, randread, randwrite [raid] dev = /dev/sdb, /dev/sdc, /dev/sdd number_realization = 0</pre>	<pre>[global] path_to_spdk_repo=/root/spdk ioengine=spdk_bdev ramp_time=5 rw = write [spdk] dev = /dev/sdb, /dev/sdc, /dev/sdd number_realization = 0</pre>

Рис. 3: Примеры возможных конфигурационных файлов

Важно отметить, что пользователю необязательно указывать все параметры в конфигурационном файле. Недостающие параметры будут взяты из шаблонных файлов в репозитории проекта, которые существуют для каждой поддерживаемой реализации RAID-массивов.

3.2.2 Инициализация RAID

Если в конфигурационном файле были секции `[spdk]` или `[raid]`, то отработывает функция для запуска необходимой реализации RAID.

Для секции `[raid]` сначала происходит проверка на то, что данный инструмент установлен на устройстве, а затем генерируется команда для создания RAID-массива. Для порождения новых процессов используется библиотека `subprocess` [13], которая инициализирует RAID-массив `mdadm` нужной версии и на указанных дисках.

В случае секции `[spdk]` нужно самостоятельно собрать проект `spdk` и нужную для работы `spdk` версию `fio`. Инструкция по установке находится в репозитории проекта. После того, как все инструкции по сборке будут выполнены, необходимо указать путь до репозитория `spdk` в конфигурационном файле. После запуска утилиты, будет сгенерирован специфичный для работы `spdk` конфигурационный файл для инициализации RAID в json формате и выставлена переменная окружения `LD_PRELOAD` нужная для запуска нужной для `spdk` версии `fio`. Далее процесс тестирования ничем не будет отличаться от тестирования простого диска.

3.2.3 Работа с FIO

Из введённого пользователем конфигурационного файла генерируется специфичный для работы инструмента FIO файл, в котором утилита сама поставит необходимые для лучшего тестирования параметры.

В итоговый конфигурационный файл для FIO добавляются параметры:

1. `invalidate=1` — флаг, для аннулирования кэширования. Нужен, чтобы корректно замерить производительность блочного устройства и всегда обращаться к нему для обработки новых запросов.
2. `ramp_time=30` — время в секундах, во время которого FIO работает с блочным устройством без сбора результатов. Нужно, чтобы результаты производительности были точнее.
3. `time_based` — параметр без значения, который указывает FIO работать в течение всего указанного времени.
4. `group_reporting` - параметр, отвечающий за вывод общей сводки по всем заданиям (jobs).
5. `direct=1` — флаг, отвечающий за небуферизированный ввод-вывод.

После запуска `fio` будет генерировать логи для каждого вида нагрузки, который был указан пользователем в конфигурационном файле,

например для только для чтения. Для того, чтобы fio сгенерировал необходимые для построения графиков логи, используются следующие параметры:

- Для количества запросов ввода/вывода в секунду: `write_iops_log`.
- Для задержек запросов перед исполнением: `write_lat_log`.
- Для подсчёта общей пропускной способности в секунду: `write_bw_log`.

Пользователю об этих нюансах знать необязательно, поэтому эти параметры от него скрыты и он не может их изменить.

3.2.4 Остановка RAID

Для Mdaadm RAID нужно остановить работу запущенного RAID-массива и обнулить блок, содержащий метаданные для настройки RAID-массива, у каждого использованного в работе RAID блочного устройства. Обнуление этого блока удаляет метаданные RAID.

Для SPDK RAID ничего останавливать не нужно так как мы работаем с версией fio от SPDK, которая делает запуск и остановку RAID-массива за нас.

3.2.5 Генерация графиков

Стоит отметить, что в конфигурационном файле пользователя может присутствовать параметр `numjobs`, который отвечает за создание нескольких копий задания для тестирования. Каждая копия будет генерировать свои файлы логов, поэтому для построения графика утилита собирает данные со всех копий и строит график по сумме всех значений.

Утилита генерирует в папку `out` графики вида:

- `Iops` — график количества операций ввода-вывода в секунду.
- `Bandwidth` — график пропускной способности в MiB/s
- `Lat` — график общей задержки `Clat` (отправки) + `Slat` (завершения) операции. Также отдельно генерируются графики `Clat` и `Slat`.

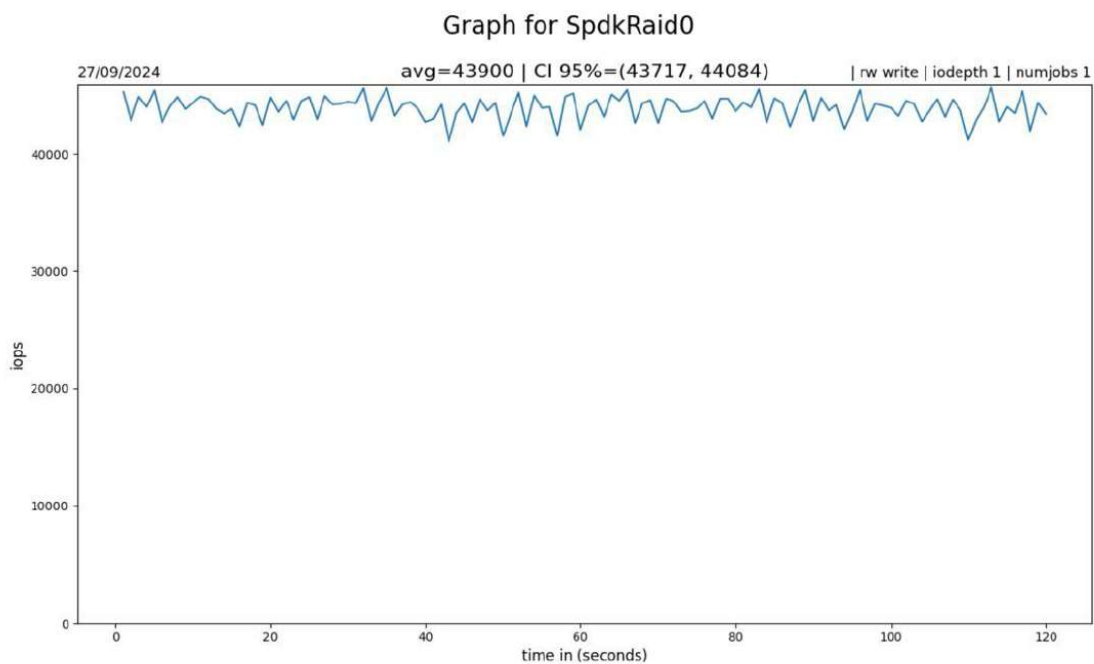


Рис. 4: Пример сгенерированного графика

Из примера видно, что на графике показаны основные параметры, которые были заданы при тестировании и дата создания. Также в заголовке графика отображается среднее по всем значениям графика и доверительный интервал в 95%.

4 Эксперимент

4.1 Подготовка и тестирование

Тестирование проводилось на выделенном сервере со следующими характеристиками:

- Процессор Intel core i7-4970 с частотой 3.6 ГГц;
- Оперативная память Kingsman DIMM DDR3 Synchronous 1600 MHz общим объёмом 16 Гб;
- 5 SSD накопителей Netac SSD 256GB NT01SA500-256-S3X;

В рамках тестирования утилиты было решено провести исследование, которое наглядно бы показало выигрыш в производительности RAID-массивов пространства пользователя перед RAID-массивами пространства ядра на примере сравнение SPDK RAID и MDADM RAID. Тестирование производилось с помощью представленной утилиты, после этого выведенные средние значения были добавлены на общую гистограмму для большей наглядности.

Так выглядели конфигурационные файлы для тестирования MDADM RAID и SPDK RAID (см рис 5).

```
Used parametrs

[global]
time_based
group_reporting
direct=1
ioengine=libaio
invalidate=1
ramp_time=30
size=128G
runtime=60
bs=4K
iodepth=8
numjobs=8
```

Рис. 5: Параметры конфигурационного файла для тестирования

Получившиеся гистограммы представлены на рисунках 6 и 7.

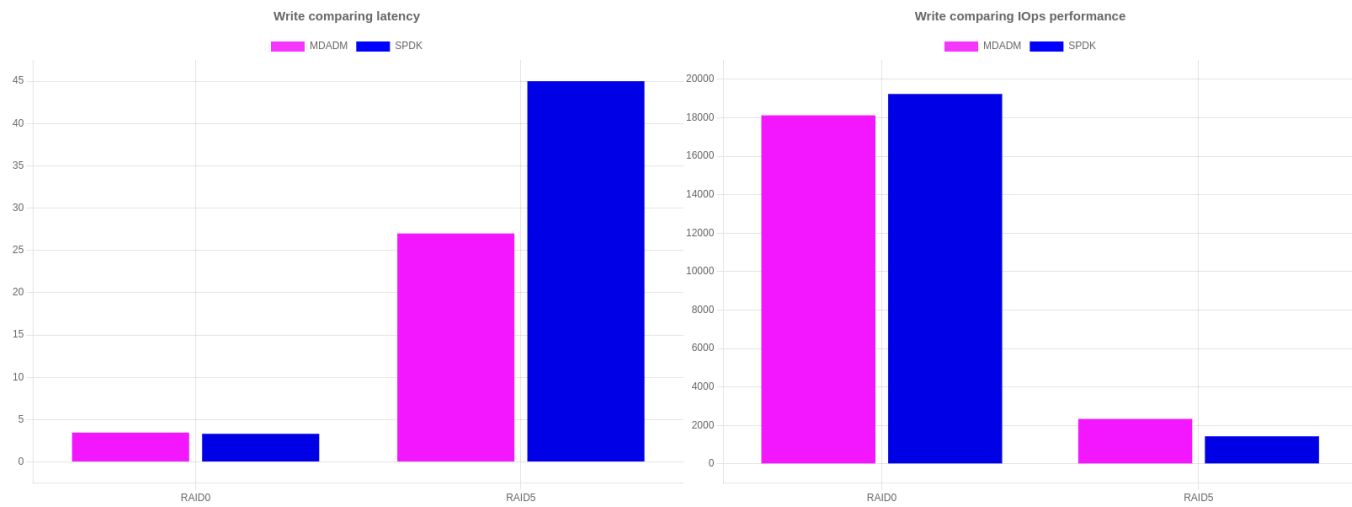


Рис. 6: Гистограмма сравнения запросов на запись

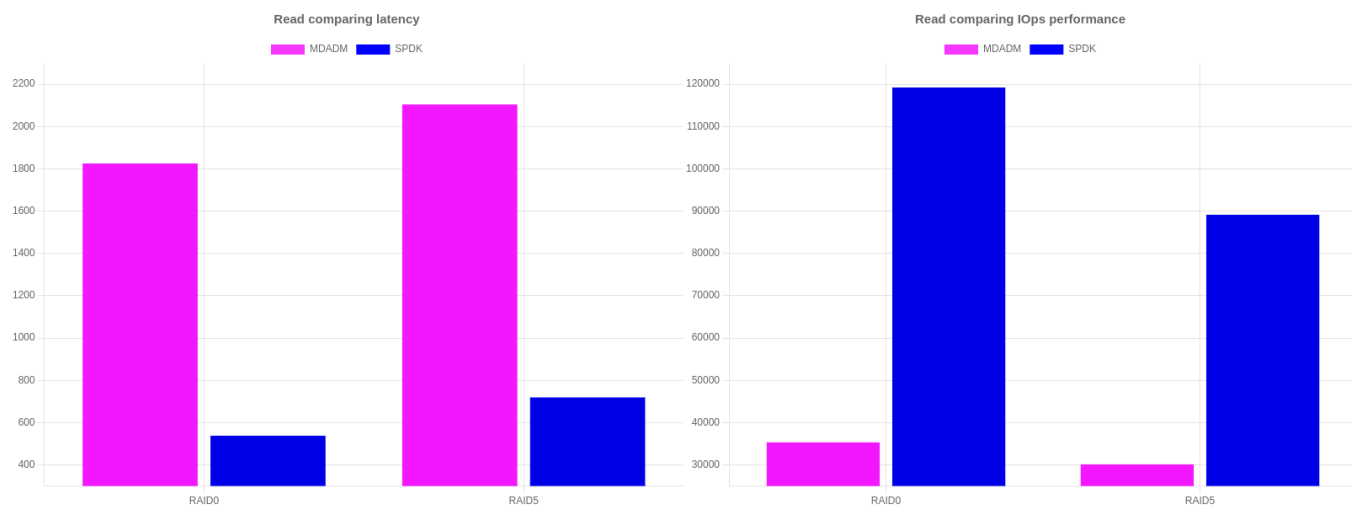


Рис. 7: Гистограмма сравнения запросов на чтение

4.2 Анализ результатов

Важно отметить, что в эксперименте принимал участие SPDK RAID-5, написанный моим коллегой.

На рисунке 6 заметно, что количество запросов ввода-вывода в секунду на запись почти не отличается у MDADM RAID и SPDK RAID. Высокая задержка в RAID-5 разных реализаций связана с его спецификой, а именно с тем, что перезаписывание контрольно-восстановительных сумм требует какого-то времени.

Однако на рисунке 7, на котором изображено количество запросов ввода-вывода в секунду на чтение, результаты отличаются на порядок. SPDK реализации RAID-массивов из пространства пользователя почти в три раза быстрее обрабатывают запросы на чтение с более низкой задержкой, чем у MDADM реализаций RAID-массивов.

Вывод: SPDK реализация RAID-массивов действительно работает быстрее, как и заявляли разработчики фреймворка. Разница сильно заметна на запросах чтения. Утилита облегчила процесс тестирования и ускорила ход эксперимента в разы.

Заключение

В данной работе были достигнуты следующие результаты:

- Рассмотрены реализации RAID-массивов из пространства ядра, созданные с помощью MDADM и из пространства пользователя, созданные с помощью SPDK
- Рассмотрена утилита FIO, которая была взята за основу в утилите для тестирования производительности разных RAID реализаций
- Спроектирована архитектура утилиты, которая позволяет в едином синтаксисе описать, какую реализацию RAID (из пространства ядра или из пространства пользователя) и какие параметры для тестирования нужно будет использовать.
- Представлен прототип утилиты³, в котором поддерживается реализации RAID-массивов не только из пространства ядра, но и из пространства пользователя.
- Проведено исследование, которое позволило сравнить производительность RAID реализации SPDK и Mdadm. Была замерена производительность SPDK RAID5, который разрабатывался с качестве курсовой работы моего коллеги.

В дальнейшем планируется добавить в утилиту поддержку OpenZFS и реализовать возможность построения гистограмм с логами от разных RAID реализаций для более наглядного сравнения их производительности.

³Готовый прототип

Список литературы

- [1] Abstract Base Classes documentation. — URL: <https://docs.python.org/3/library/abc.html> (дата обращения: 26 сентября 2024 г.).
- [2] Configparser module documentation. — URL: <https://docs.python.org/3/library/configparser.html> (дата обращения: 19 декабря 2023 г.).
- [3] FIO-plot. — <https://github.com/louwrentius/fio-plot>.
- [4] Fio script. — https://github.com/axboe/fio/blob/master/tools/fio_generate_plots.
- [5] Flexible I/O documentation. — URL: https://fio.readthedocs.io/en/latest/fio_doc.html (дата обращения: 19 декабря 2023 г.).
- [6] INI file format. — URL: <https://ru.wikipedia.org/wiki/.ini> (дата обращения: 26 сентября 2024 г.).
- [7] Mdadm wiki. — URL: <https://en.wikipedia.org/wiki/Mdadm> (дата обращения: 19 декабря 2023 г.).
- [8] Os module documentation. — URL: <https://docs.python.org/3/library/os.html> (дата обращения: 19 декабря 2023 г.).
- [9] Pyplot documentation. — URL: <https://matplotlib.org/stable/tutorials/pyplot.html> (дата обращения: 19 декабря 2023 г.).
- [10] Repository of the resulting utility. — URL: <https://github.com/PavlushaSource/Rlot> (дата обращения: 26 сентября 2024 г.).
- [11] Storage Performance Development Kit RAID. — URL: <https://spdk.io/doc/bdev.html> (дата обращения: 19 декабря 2023 г.).
- [12] Storage Performance Development Kit documentation. — URL: <https://spdk.io/doc/about.html> (дата обращения: 19 декабря 2023 г.).

- [13] Subprocess module documentation. — URL: <https://docs.python.org/3/library/subprocess.html> (дата обращения: 19 декабря 2023 г.).
- [14] Sys module documentation. — URL: <https://docs.python.org/3/library/sys.html> (дата обращения: 19 декабря 2023 г.).