Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.Б11-мм

Реализация RAID5 в фреймворке SPDK

Байтенов Арсений Леонидович

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
Старший преподаватель кафедры системного программирования, Я. А. Кириленко

Консультант:
Ведущий инженер-программист ООО «Швачер», А. И. Васенина

Санкт-Петербург
2023

Оглавление

Введение

На уровне планирования процессорного времени, обработка запросов чтения/записи происходит асинхронно. Поток посылает запрос блочному устройству, происходит смена контекста, и вычислительные ресурсы передаются другому потоку. Когда блочное устройство обработает запрос, или во время обработки возникнет ошибка, вызовется прерывание, опять сменится контекст, и ресурсы будут переданы потоку, инициировавшему запрос. Между вызовом запроса и моментом, когда придёт ответ, проходит довольно много времени, то есть оптимизация работы с блочными устройствами может существенно улучшить скорость работы программы.

Операцию, которая обращается к длинной непрерывной области памяти, можно было бы обрабатывать параллельно, если бы эта область памяти на самом деле располагалась на нескольких блочных устройствах. Отсюда рождается идея RAID-массивов[1], массивов, объединяющих базовые блочные устройства в единое пространство хранения данных. Иными словами, с целью повышения производительности и отказоустойчивости диски объединяются в один логический модуль, который для внешний среды представляется как новое блочное устройство. RAID-массивы классфицируются по разным уровням, которые предоставляют различные преимущества за некоторую потерю в полезном объёме памяти. RAID0 представляет из себя простое объединение блочных устройств, не гарантирующие какую-либо отказоустойчивость, тем не менее позволяющее ускорить обработку запросов чтения/записи, за счёт того, что запрос к непрерывной снаружи области памяти, на самом деле выполняется сразу несколькими блочными устройствами одновременно. RAID1 дублирует данные на всех дисках, тем самым гарантируя возможность восстанавления информации, если есть хотя бы одна целая копия. RAID5 представляет из себя RAID0 с контрольно-восстановительными суммами, кроме ускорения обработки запросов предоставляет возможность восстановить данные в случае вы-

[1] Redundant Array of Independent Devices.

хода из строя одного диска. Пожертвовать приходится пространством памяти суммарно равным размеру одного базового блочного устройства. Именно эта память тратится на контрольно-восстановительные суммы, хранящиеся на разных базовых блочных устройствах. Реализации RAID-массивов бывают как аппаратные, так и программные. Аппаратная реализация имеет более высокую стоимость, однако контроллер выполняет вычисления самостоятельно, снижая нагрузку на процессор. Программный аналог может требовать конкретную операционную систему, но предоставляет возможность обновления и настройки без приобретения дополнительного оборудования, а также более широкие возможности оптимизации.

С развитием технологий скорость обработки запроса диском возросла, таким образом, задержка на переключение контекста процессора стала чувствительней. Аппаратное обеспечение больше не является узким местом, наоборот оно сместилось в сферу программных механизмов, так как, кроме прочего, из-за взаимодействие с блочными устройствами через пространство ядра приходится выполнять операции копирования данных запросов из пространства ядра в пространство пользователя. Решением этого вопроса в фреймворке SPDK[2] стала возможность работать с блочными устройствами через пространство пользователя. По performance reports компании Intel [3], это делает работу СХД быстрее.

В SPDK есть реализации RAID0 и RAID1, однако нет реализации RAID5. Таким образом, целью работы является добавление функциональности RAID5 в фреймворк SPDK и сравнение с аналогами.

[2]Storage Performance Development Kit.

1 Постановка задачи

Целью работы является реализация RAID5 в фреймворке SPDK. Для её выполнения были поставлены следующие задачи:

1.1 Осенний семестр

1. спроектировать систему функций и callback-функций для регистрации запросов чтения и записи в RAID5;
2. реализовать RAID5;
3. написать интеграционные тесты для проверки корректности полученной реализации.

1.2 Весенний семестр

1. реализовать обработку запросов с нулевой нагрузкой в RAID5;
2. проверить корректность полученной реализации;
3. сравнить производительность полученной реализации с производительностью аналогов.

2 Обзор

2.1 RAID

RAID — Redundant Array of Independent Devices — технология хранения данных, основанная на объединении базовых устройств хранения в массив, единый логический модуль, представляющий из себя новое блочное устройство. Смысл такого объединения заключается в повышении скорости обработки запросов и(или) отказоустойчивости. Для понимания последующего рассказа будут введены некоторые термины. Для сопостовления адресов виртуального устройства с адресами базовых устройствам используются стрипы [4], области памяти на базовых устройствах. Стрипы адресуются последовательно на каждом базовом устройстве и имеют одинаковый объём памяти. Страйп [4] представляет из себя набор стрипов в соответствующих местах базовых устройств. Неформально его можно описать как один «уровень» стрипов в массиве базовых устройств. Отношение между понятиями стрип и страйп графически представлено на рис. 1. Для восстановления потерянной информации некоторые уровни RAID-массивов предполагают хранение контрольно-восстановительных сумм на базовых устройствах. Стрип, в котором эти суммы хранятся называется стрипом паритета. Расположение таких стрипов может быть разным, пример расположения стрипов паритетов RAID5 приведён на рис. 2.

2.1.1 RAID0

RAID0 — RAID-массив, в котором последовательные логические блоки данных равномерно распределяются по набору независимых базовых устройств. Такое распределение данных позволяет увеличить скорость обработки запросов чтения и записи за счёт распараллеливания запроса, так как подряд идущие данные находятся на разных устройствах. RAID0 не содержит какой-либо информации для восстановления потерянных данных, что может привести к потере данных при выходе из строя хотя бы одного базового устройства. Изображение RAID0 приве-

дено на рис. 1.

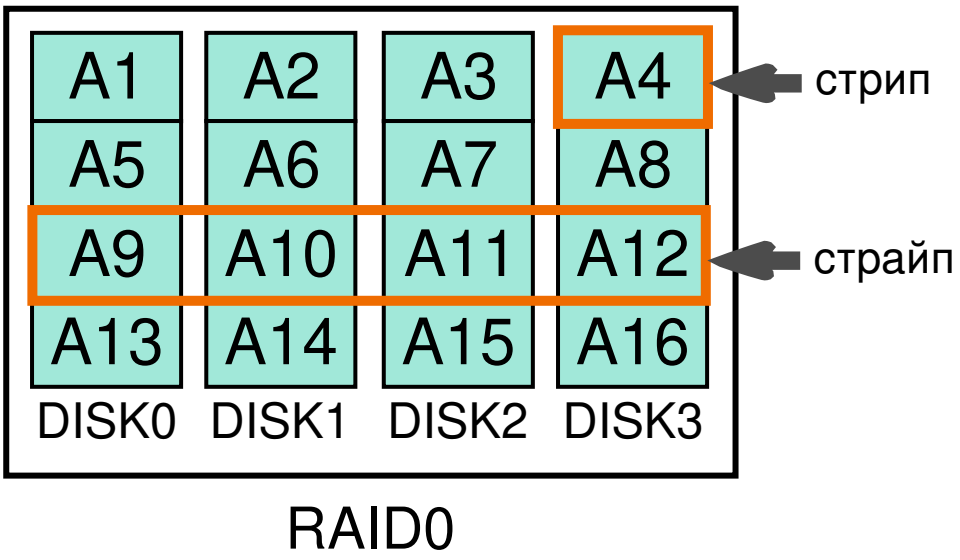

Рис. 1: RAID0 на четырёх базовых устройствах.

2.1.2 RAID5

RAID5 отличается от RAID0 тем, что в каждом страйпе один из стрипов отводится под данные паритета. В этом стрипе хранится результат операции XOR стрипов с данными. Таким образом, теряется объём данных, суммарно равный объёму одного базового устройства. RAID5 проигрывает RAID0 не только по объёму полезных данных, но и по скорости обработки запроса из-за необходимости обновления стрипов паритета. За счёт этих модификаций RAID5 предоставлят возможность работы при выходе из строя не более чем одного стрипа в страйпе. Изображение RAID5 приведено на рис. 2.

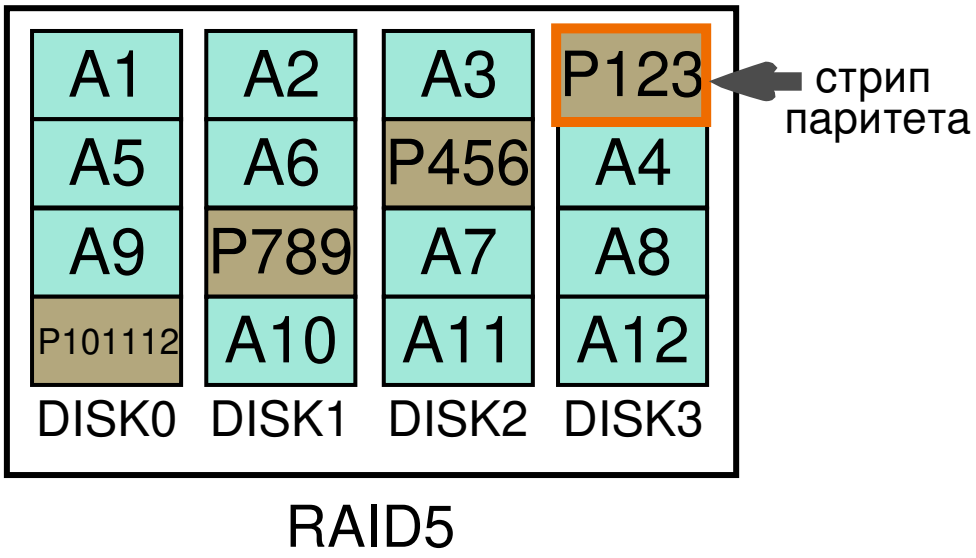

Рис. 2: RAID5 на четырёх базовых устройствах.

2.2 SPDK

SPDK — Storage Performance Development Kit — фреймворк с открытым исходным кодом для разработки высокопроизводительных пользовательских приложений хранения данных. Набор инструментов и библиотек разработан компанией Intel. Особенностью SPDK является взаимодействие с блочными устройствами через пространство пользователя. Одним из преимуществ фреймворка является возможность использования буфферов, выделяемых пользователем для отправки запросов на устройства, поддерживающие DMA, технологию прямого доступа к памяти без участия центрального процессора. То есть драйвер устройства, поддерживающего DMA, например SSD NVMe, работающий в пространстве пользователя, минимизирует количество операций копирования из kernel space в user space [5]. Кроме того, драйвер NVMe в SPDK, опрашивает устройство во время запроса чтения/записи. Современные SSD NVMe успевают обработать запрос до этого момента, то есть не приходится переключать ресурсы процессора на другие потоки

или процессы [7].

2.3 Аналоги

В SPDK реализован только нулевой уровень RAID-массивов, а именно RAID0. В то время как, непосредственными открытыми программными аналогами RAID5 являются решения, предоставляемые утилитой mdadm внутри ядра Linux и файловой системой OpenZFS.

2.3.1 RAID0 в SPDK

RAID0 в SPDK содержит обработку запросов чтения/записи к одному стрипу. Она реализована через делегирование запроса соответствующему базовому блочному устройству, посредством пересчёта логического адреса стрипа в физический. RAID0 не содержит каких-либо вычислений паритетных данных или выделений памяти, передавая данные запроса такими, какими получает. Таким образом, эта реализация не ограничивает поддержку DMA.

2.3.2 mdadm: RAID5

Утилита mdadm позволяет создать RAID5, однако драйвер такого RAID-массива работает в пространстве ядра, то есть обработка запросов происходит в kernel space. Это ведёт к необходимости копировать данные запросов из пространства ядра в пространство пользователя, а также к задержкам на переключение контекста процессора. Кроме того, недостатком этой реализации являются глобальные блокировки [6], например при пересчёте паритетов, что опять же снижает производительность.

2.3.3 OpenZFS: ZRAID

OpenZFS — платформа хранения данных, включающая в себя функциональность файловой системы. В OpenZFS нет реализации RAID5, однако есть решение сравнимое с ним — ZRAID [2]. Первое отличие —

это использование технологии “copy-on-write”, идея которой заключается в хранении копий запросов записи в незанятой памяти. В некоторый момент времени система принимает попытку атомарной записи части данных в соответсвующее место. Такая технология защищает данные от внезапного сбоя всего ZRAID. Вторым отличием от RAID5 является нефиксированный размер страйпа, ведь благодаря файловой системе появляется знание, какая память занята, в отличие от RAID5, вынужденного гарантировать сохранность всех данных на дисках. Тем не менее, драйвер ZRAID всё равно работает в kernel space, что ведёт к необходимости копировать данные запросов из пространства ядра в пространство пользователя.

3 Реализация

Реализация была написана на языке C, поскольку в фреймворке SPDK интерфейс для встраивания новых уровней RAID-массивов реализован на языке C. Чтобы встроить новый модуль, необходимо инициализировать структуру, ассоцированную с данным модулем. Кроме уровня RAID-массива и минимального количества базовых устройств, необходимо указать, по крайней мере, две функции: функцию старта устройства(`raid5_start`) и функцию регистрации запроса чтения/записи(`raid5_submit_rw_request`). В функции старта текущей реализации взводится флаг, указывающий на требование разбивать запросы для модуля на запросы размером не больше страйпа. Функция регистрация запроса чтения/записи принимает на вход указатель на структуру `raid_bdev_io`, ассоцированную с запросом к RAID-массиву. Однако эта структура не предполагает наличие каких-либо временных буферов для хранения данных, которые будут читаться из или записываться в несколько отдельных базовых устройств.

3.1 Структуры запроса

Буферы с пространством памяти выделенным под данные делегированных запросов к базовым устройствам хранятся в структуре `raid5_stripe_request`. Память для неё необходимо выделять в начале регистрации каждого запроса и освобождать после выполнения. Связи структур, нужных для обработки запроса можно посмореть на рис. 3.

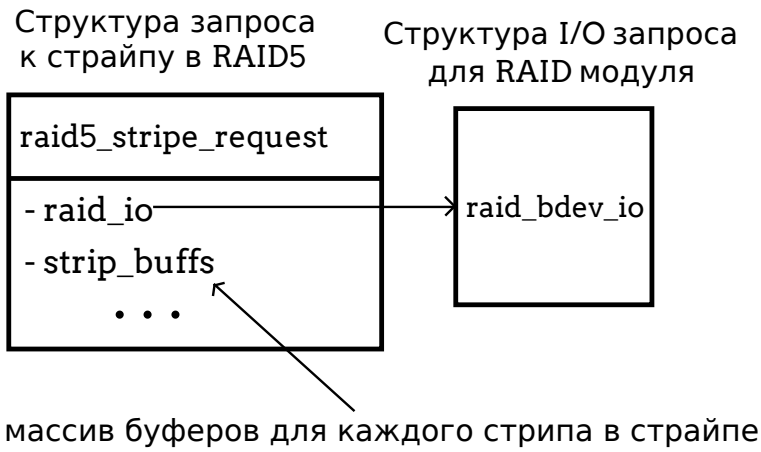

Рис. 3: Структура запроса чтения/записи к страйпу.

3.2 Обработка запросов чтения

Для обработки запросов чтения из функции `raid5_submit_rw_request` вызывается функция `raid5_submit_read_request`, которая нужна, чтобы определить сценарий и выделить память для буферов. В силу того, что RAID5 гарантирует корректность работы случае выхода из строя не более одного стрипа в каждом страйпе, возникают два сценария при обработке запросов чтения. Схема вызова функций для обоих вариантов изображена на рис. 5.

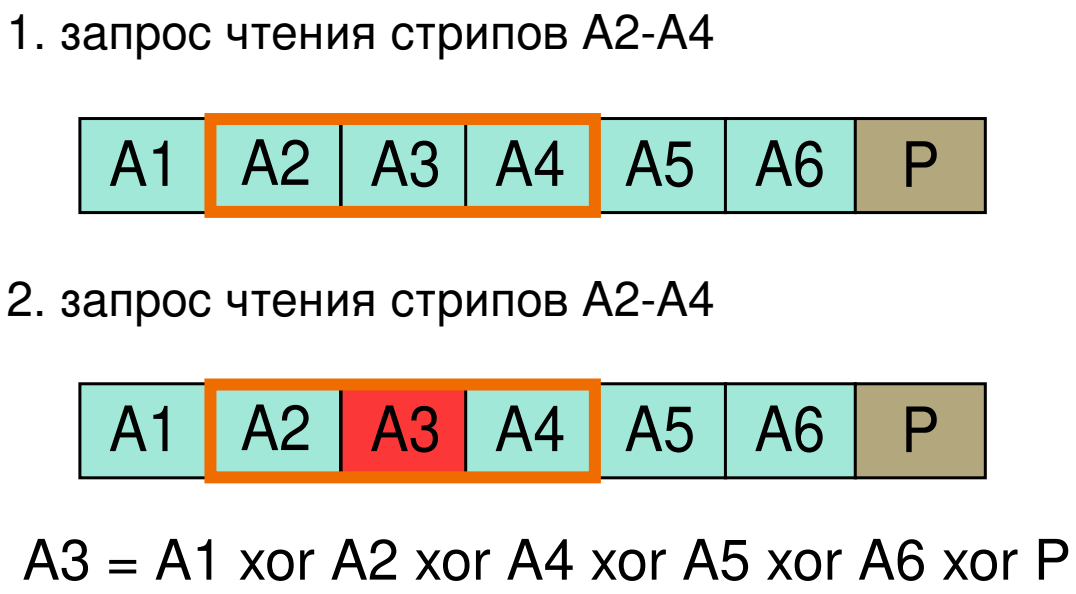

Рис. 4: Два сценария запроса чтения.

3.2.1 Без битых стрипов

В этом сценарии базовые блочные устройства, которые напрямую задействованы в запросе не содержат битых стрипов. Необходимо разделить память, выделенную под чтение на отдельные буферы, после этого вызывается функция `raid5_read_req_strips`, которая делегирует запросы чтения нужным устройствам. Обработка запроса закончится после того, как базовые устройства обработают запросы, делегированные им. Сценарий изображён на рис. 4 под номером один.

3.2.2 Один битый стрип

Во втором сценарии один из стрипов запроса находится в нерабочем состоянии, значит надо воспользоваться данными со всего страйпа, чтобы восстановить информацию. Кроме разделения данных запроса, необходимо выделить память для чтения данных со стрипов, принадлежащих тому же страйпу, но не учавствовающих в запросе напрямую. После этого вызывается функция `raid5_read_except_one_req_strip`,

которая отправляет запросы чтения всем базовым устройствам, кроме того которое содержит битый стрип. И после завершения всех остаётся посчитать результат операции XOR на полученных данных, а также очистить память. Сценарий представлен на рис. 4 под номером два.

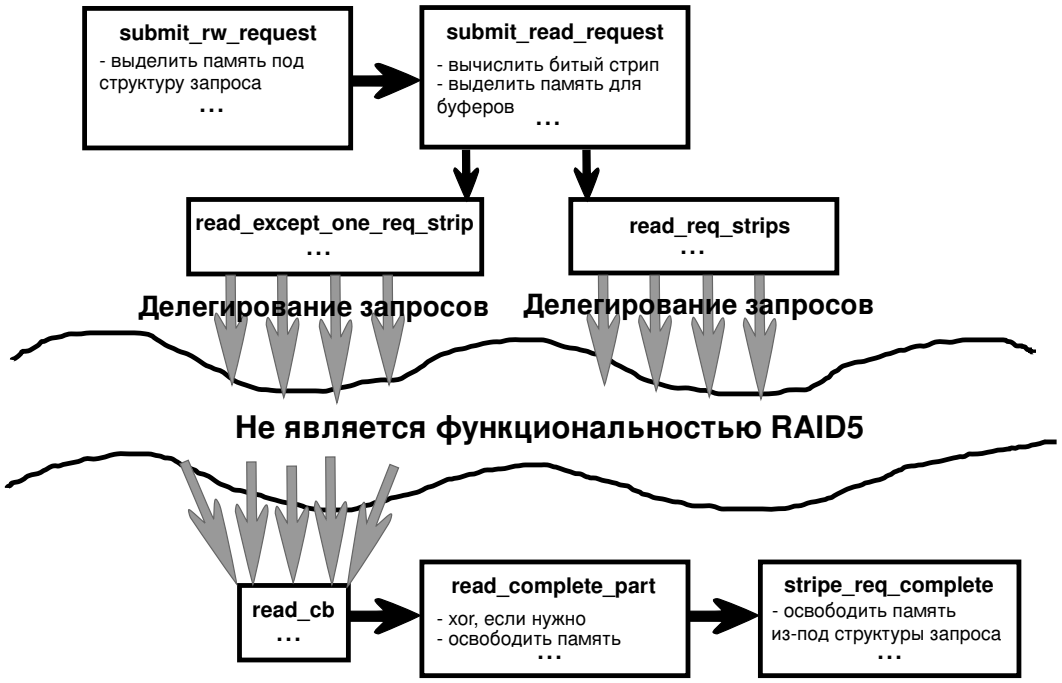

Рис. 5: Схема выполнения запроса чтения к страйпу.

3.3 Обработка запросов записи

Для обработки запросов записи из функции `raid5_submit_rw_request` вызывается функция `raid5_submit_write_request`, в которой определятся сценарий записи и выделяется память для буферов. Все сценарии обобщаются до четырёх общих случаев(рис. 6). Два из этих четырёх случаев зависят от согласованности данных, то есть то есть является ли стрип паритета результатом операции XOR остальных стрипах страйпа.

При старте RAID5 запускается функция, которая пробует инициализировать нулями все базовые устройства. Это происходит посредством попытки отправить write-zeroes запрос на каждое из них. Если все запросы на всех базовых устройствах прошли успешно, то контрольно-

восстановительные суммы в каждом страйпе валидны и можно использовать для записи стратегию “read-modify-write”. Идея этой стратегии заключается в том, что для пересчёта стрипа паритета необязательно считывать весь страйп, достаточно лишь прочитать старые данные, хранящиеся стрипах из запроса, и старых данных из стрипа паритета:

$$P \oplus A1 \oplus A1_{new} = (A1 \oplus A2 \oplus ... \oplus A7) \oplus A1 \oplus A1_{new} = A1_{new} \oplus A2 \oplus ... \oplus A7 = P_{new}$$

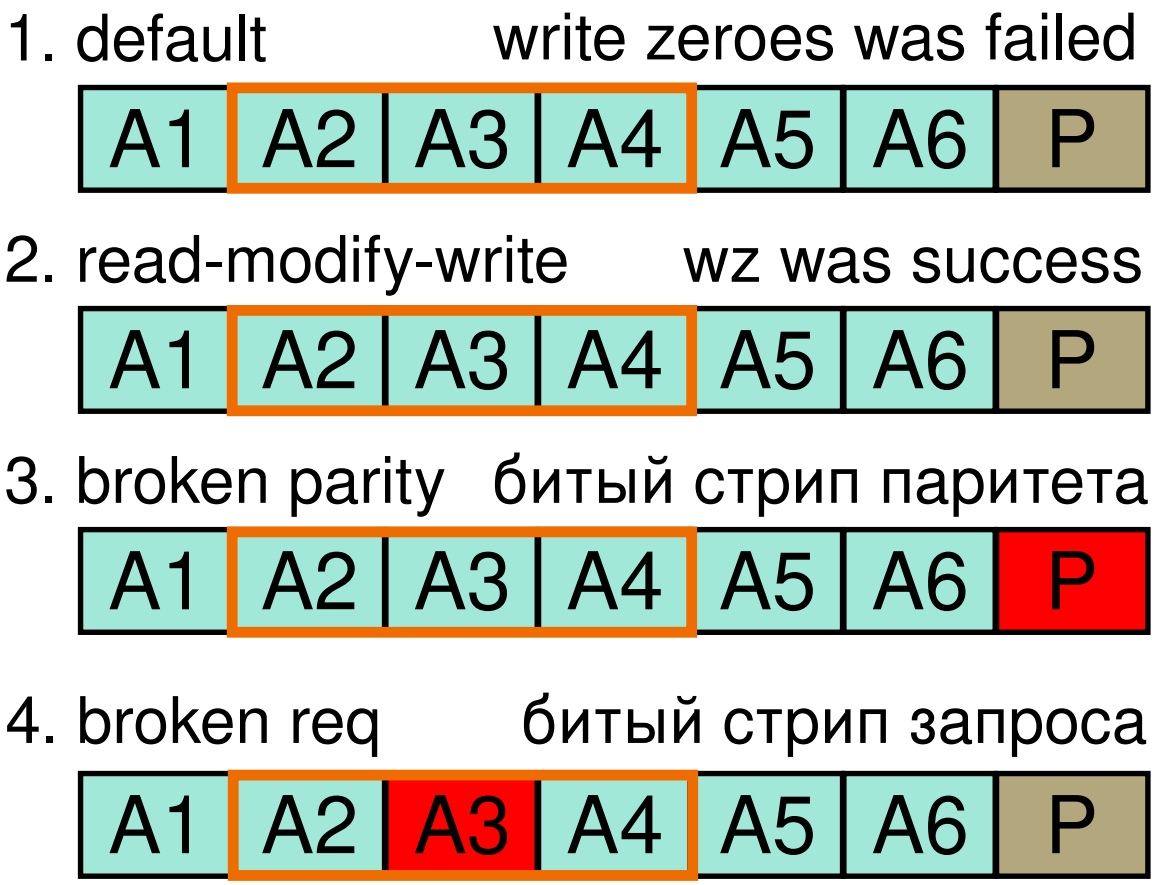

Рис. 6: Четыре сценария запроса записи.

3.3.1 Обычный(Default)

Обычная запись означает, что при каждом запросе записи, читаются данные со стрипов, неучавствующих в запросе, и результат XOR стрипов по всему страйпу, записывается как новый паритет. Выделяются буффера, вызывается функкция `raid5_write_default_reading` (рис. 7), которая посылает запросы чтения на базовые устройства, которые не учавствуют в запросе к страйпу. В случае успешного завершения

всех запросов чтения, вычисляется новая контрольно-восстановительная сумма, вызывается функция `raid5_write_default_writing`, в которой отправляются запросы записи стрипов изначального запроса и стрипа паритета.

3.3.2 Read-modify-write

В соответсвиии со стратегией "read-modify-write" выделяются буферы для хранения результатов запросов чтения, после вызывается функция `raid5_write_r_modify_w_reading` (рис. 7), которая отправляет запросы на чтение старых данных, после успешного завершения всех запросов чтения, вычисляется новый паритет, заменяются буферы, вызывается функция `raid5_write_r_modify_w_writing`. Отправляются запросы записи нужных стрипов из запроса к страйпу и стрипа паритета.

3.3.3 Битый стрип паритета

В этом случае невозможно перезаписать стрип паритета, а ,значит, и считать его не нужно. Необходимо только записать данные в запрашиваемые стрипы. Инициализируются буферы для запрашиваемых стрипов. После вызывается функция `raid5_write_broken_parity_strip`, в которой отправляются запросы на запись запрашиваемых стрипов.

3.3.4 Битый стрип запроса

Информация, которую надо было бы записать в битый стрип, сохранится в стрипе паритета вместе с данными остальных стрипов, то есть необходимо пересчитать и перезаписать контрольно-восстановительную сумму а также записать остальные стрипы запроса. Выделяются буферы, вызывается функция `raid5_write_broken_req_reading` (рис. 7), которая посылает запросы чтения,чтобы получить стрипы, которых не хватает для вычисления нового значения паритета. Если все запросы отработали успешно, вычисляется новая контрольно-восстановительная сумма, изменяются буферы. После чего вызывается функция `raid5_wri-`

te_broken_req_writing, которая отправляет запросы записи изначально запрашиваемых стрипов и стрипа паритета.

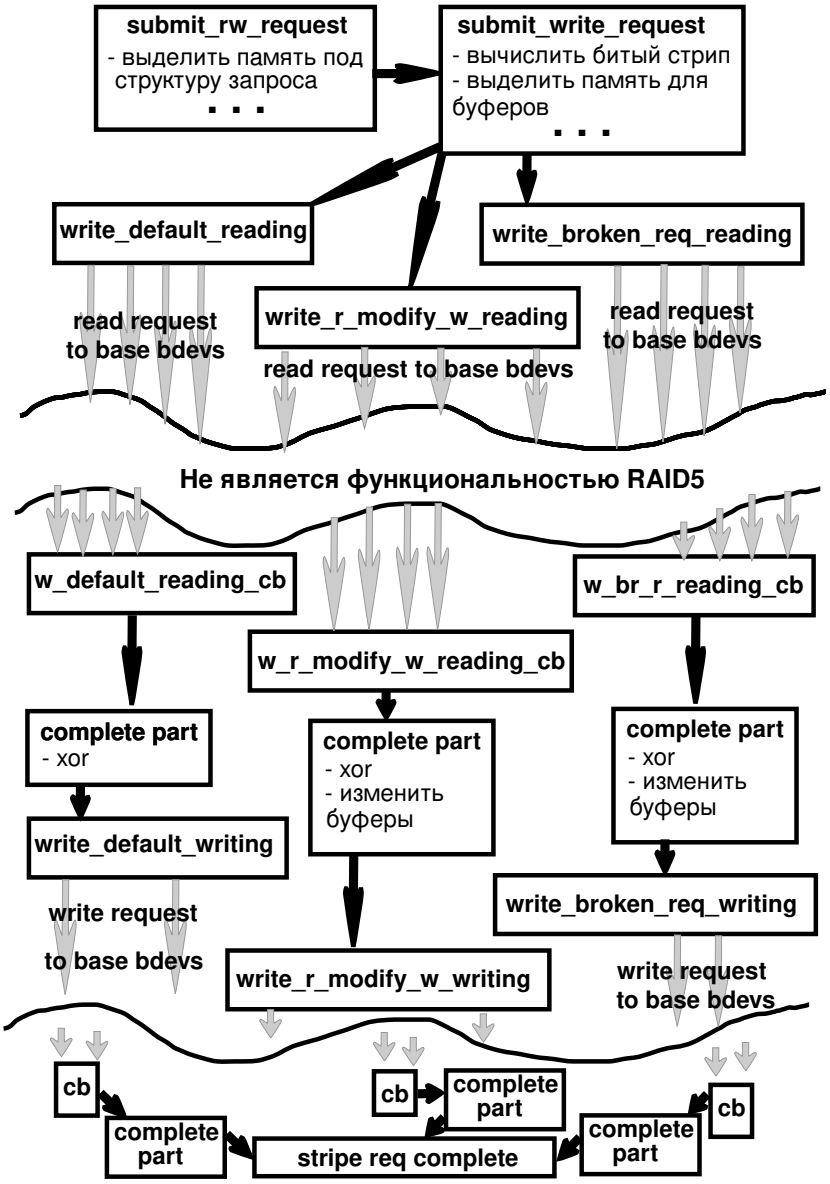

Рис. 7: Схема выполнения запроса записи к страйпу с чтением для пересчёта контрольно-восстановительных сумм.

4 Тестирование

Тестирование полученной реализации проводилось через Userspace block device driver. Bash-скрипт, создавал блочные устройства, аллоцированные в пространстве пользователя и RAID5, с параметрами из конфигурационных файлов. Тесты проводились с помощью утилиты fio [1], которая позволяет генерировать запросы чтения и записи на блочное устройство без использования файловой системы. В ходе тестирования используются нагрузки двух видов: запись и запись в случайное место. Первый из посылает запросы, адресуя непрерывные блоки по последовательным адресам на RAID5, второй же подразумевает выбор адреса случайным образом.

Проверка каждой конфигурации проводилась по следующему сценарию. Создавались базовые устройства и RAID5, с помощью утилиты fio на них генерировалась нагрузка каждого вида, с верификацией md5-сумм. После удалялось одно из блочных устройств и снова генерировалась нагрузка, тем самым проверялась не только корректность работы в обычном режиме, но и с вышедшим из строя базовым устройством.

Заключение

В течение семестра получены следующие результаты:

- спроектирована система функций и callback-функций для регистрации запросов чтения и записи в RAID5;
- реализован RAID5 с обработкой запросов к страйпам[3];
- написаны интеграционные тесты для проверки корректности полученной реализации.

В течение весеннего семестра планируется добавить обработку запросов с нулевой нагрузкой и сравнить производительность полученной реализации с производительностью аналагов.

[3] Полученная реализация

Список литературы

[1] Fio documentation. — URL: https://fio.readthedocs.io/en/latest/index.html (дата обращения: 2023-12-21).

[2] Hickmann Brian, Shook Kynan. Zfs and raid-z: The über-fs? // University of Wisconsin–Madison. — 2007.

[3] Karol Latecki, Jaroslaw Chachulski. SPDK NVMe BDEV Performance Report Release 23.09. — 2023. — URL: https://ci.spdk.io/download/performance-reports/SPDK_nvme_bdev_perf_report_2309.pdf (дата обращения: 2023-12-21).

[4] The SNIA Dictionary. — URL: https://www.snia.org/education/online-dictionary (дата обращения: 2023-12-21).

[5] The SPDK Documentation, NVMe Driver. — URL: https://spdk.io/doc/nvme.html (дата обращения: 21 декабря 2023 г.).

[6] ScalaRAID: Optimizing Linux Software RAID System for Next-Generation Storage / Shushu Yi, Yanning Yang, Yunxiao Tang et al. // Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems. — 2022. — P. 119–125.

[7] Steven Briscoe, Thai Le. Accelerating your NVMe drives with SPDK. — 2017. — URL: https://www.intel.com/content/www/us/en/developer/articles/technical/accelerating-your-nvme-drives-with-spdk.html (дата обращения: 2023-12-21).