Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.Б11-мм

Организация процесса восстановления данных в RAID1 и RAID5 на основе SPDK

Сичкар Георгий Константинович

Отчёт по учебной практике
в форме «Решение»

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

Санкт-Петербург
2024

Оглавление

Введение

Системы хранения данных (СХД) стали неотъемлемой частью современного мира. Многие компании из разных сфер используют их для своих нужд. Например, компании из финансовой сферы, используют СХД для сохранения информации о денежных транзакциях. Так одной из самых важных характеристик СХД является надежность и отказоустойчивость. Иными словами, способность системы хранения в случае частичной поломки иметь возможность восстановить потерянные данные и продолжить функционировать в штатном режиме.

Яркими примерами технологий с подобной функциональностью являются RAID-массивы разных уровней, в частности RAID1 и RAID5. Уровень RAID характеризует его внутреннюю архитектуру. RAID1 хранит дубликаты информации чтоб не допустить их потерю, в то время как RAID5 для достижения того же эффекта применяет контрольные суммы, в которые закодирована часть хранимой информации. Так, являясь логическими объединениями нескольких блочных устройств, эти системы хранения данных за счёт избыточности предоставляют конечному пользователю определенный уровень отказоустойчивости.

На данный момент большинство программных реализаций RAID-массивов работают в пространстве ядра. Однако одной из альтернатив является SPDK — фреймворк с открытым исходным кодом для работы с блочными устройствами, разработанный Intel Corporation. Особенностью SPDK является возможность управлять логикой работы блочных устройств из пространства пользователя. Более того, опираясь на отчеты о производительности компании Intel [5], можно утверждать об увеличении скорости работы СХД. Фреймворк содержит внутри себя реализацию массива RAID1. Однако эта реализация неполная и содержит ряд критических недочетов. Например, несмотря на наличие функциональности для непосредственного восстановления данных, контроль целосности данных не осуществляется. Это в свою очередь приводит к возможности некорректной обработки запросов на чтение или запись.

Некоторые RAID-массивы, обладают функциональностью, которая

отвечает за отказоустойчивость. Эта функциональность называется системой ребилда и ее особенности реализации зависят от уровня RAID-массива. Одной из подзадач этой системы является восстановление поврежденных областей — недоступных для чтения и записи. Информация об этих областях, сохраняется во время работы RAID-массива. Подобный процесс восстановления проводится в фоновом режиме. Иными словами, процесс ребилда не блокирует доступ пользователей к ресурсам RAID-массива, а проводится параллельно штатной работе. После успешного восстановления, области помечаются доступными на запись и чтение.

1 Постановка задачи

Целью данной учебной практики является реализация процесса восстановления поврежденных данных, внутри массива массивов RAID1 и RAID5 на базе фреймворка SPDK.

Для достижения этой цели были поставлены следующие задачи.

1.1 Осенний семестр

1. Изучить устройство и реализации RAID-массивов.
2. Разработать архитектуру процесса восстановления, с учетом принятых в SPDK соглашений.
3. Реализовать процесс восстановления для RAID1 без возможности параллельной обработки запросов на чтение и запись.
4. Провести ручное тестирование корректности.

1.2 Весенний семестр

1. Описать реализацию процесса восстановления данных RAID1 в SPDK.
2. Автоматизировать тестирование функциональности восстановления данных в RAID1.

1.3 Дальнейший план развития

1. Добавить реализацию процесса восстановления в RAID5.
2. Добавить в RAID1 возможность параллельной обработки запросов на запись и чтение.

2 Обзор

2.1 Технология RAID

RAID *(redundant array of independent disks)* — это технология хранения данных. Основная концепция технологии заключается: в предоставлении интерфейса взаимодействия с объединением нескольких устройств хранения данных; а также, в организации работы этих устройств, в качестве единой СХД. В основном RAID-массивы применяются либо для увеличения скорости доступа к данным, либо для повышения отказоустойчивости хранилища. Возможности RAID-массива зависят от его уровня. Так RAID1 имеет высокую надежность, однако проигрывает в доступном пространстве RAID5. RAID0, в свою очередь производит обработку запросов быстро, хотя и обладает самой низкой надежностью. Схематично описанные соотношения продемонстрированы на рис. 1.

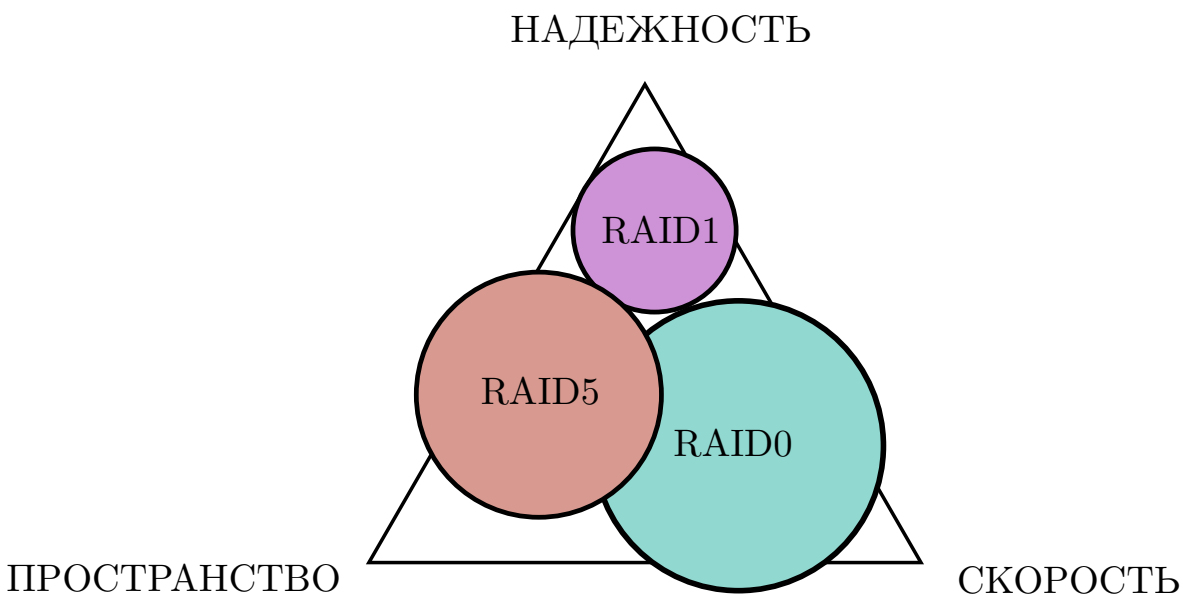

Рис. 1

Важной особенностью технологии RAID является использование страйпинга. Один страйп состоит из нескольких стрипов, расположенных на разных базовых устройствах. В свою очередь, стрип — это несколько подряд идущих блоков данных на базовом устройстве, входящем в состав RAID-массива. При этом количество стрипов внутри

одного устройства одинаково. В состав страйпа входят только стрипы, имеющие одинаковый порядковый номер. На рис. 2 страйп в массиве из трех базовых устройств состоит из стрипов: `A2`, `B2`, `C2`.

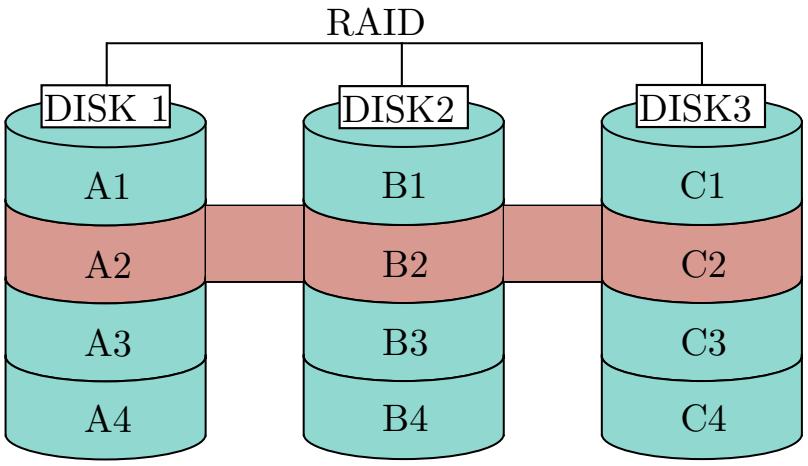

Рис. 2: Страйп внутри RAID-массива из трех базовых устройств.

2.1.1 RAID1

RAID1 подразумевает под собой наличие хотя бы двух базовых устройств. При этом данные на этих устройствах являются копиями друг друга. Иными словами, внутри одного страйпа одни и те же данные повторяются столько раз, сколько базовых устройств входит в конкретный RAID1. На рис. 3 `DISK2` и `DISK3` содержат в себе данные, идентичные тем, что хранятся на `DISK1`. Производительность этого уровня RAID при чтении выше, чем у одного диска, так как считывать данные можно параллельно со всех базовых устройств. Однако производительность при записи никак не меняется относительно одного устройства. Важной особенностью RAID1 является высокая отказоустойчивость. Этот массив будет корректно предоставлять доступ к данным, пока функционирует хотя бы одно базовое устройство. Слабой стороной RAID1 является небольшой объем хранимых данных, относительно всех базовых устройств.

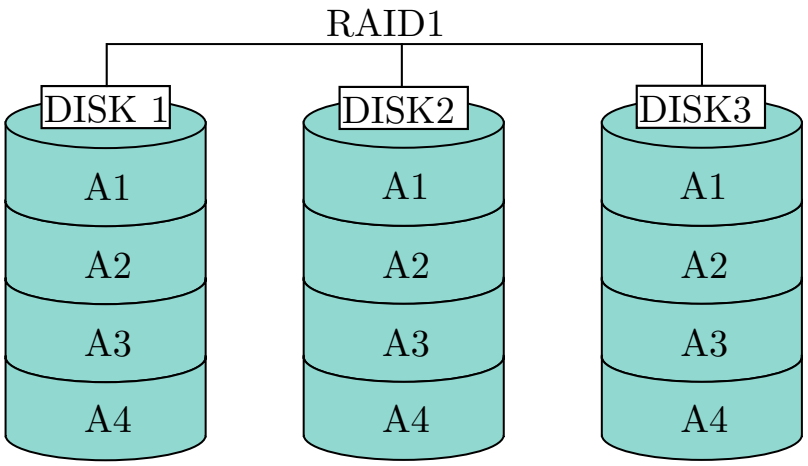

Рис. 3: RAID1 из трех базовых устройств.

2.1.2 RAID5

Основная концепция RAID5 заключается в использование контрольно восстановительных сумм, называемых паритетами *(parity)*, в чередовании с обычными стрипами данных. Иными словами, внутри одного страйпа ровно один стрип хранит вспомогательную информацию, которая может использоваться для восстановления данных на вышедшем из строя базовом устройстве.

Для вычисления паритетов внутри RAID5 используются три свойства побитовой операции XOR:

- $x \oplus y = y \oplus x$ — коммутативность;
- $(x \oplus y) \oplus z = x \oplus (y \oplus z)$ — ассоциативность;
- $(x \oplus y) \oplus y = x$ — реверсивность;

Таким образом, если вычислять стрип паритета, как XOR всех стрипов внутри одного страйпа, тогда при выходе из строя одного базового устройства, данные из поврежденного стрипа могут быть вычислены следующим образом: попарное применение XOR ко всем оставшимся стрипам внутри страйпа, а также к паритету. Схематичное расположение стрипов паритетов и стрипов данных представлено на рис. 4 Важно заметить, что RAID5 защищает от выхода из строя только одного ба-

зового устройства. Если информция повреждена сразу на двух устройствах, то восстановлению она не подлежит.

Таким образом RAID5 эффективнее использует память базовых устройств, чем RAID1. Однако уровень отказоустойчивости у него ниже. Еще одним преимуществом RAID5 над RAID1 является увеличение производительности при записи. Так как запросы, могут распределятся по устройствам в составе RAID5, распараллеливая обработку данных.

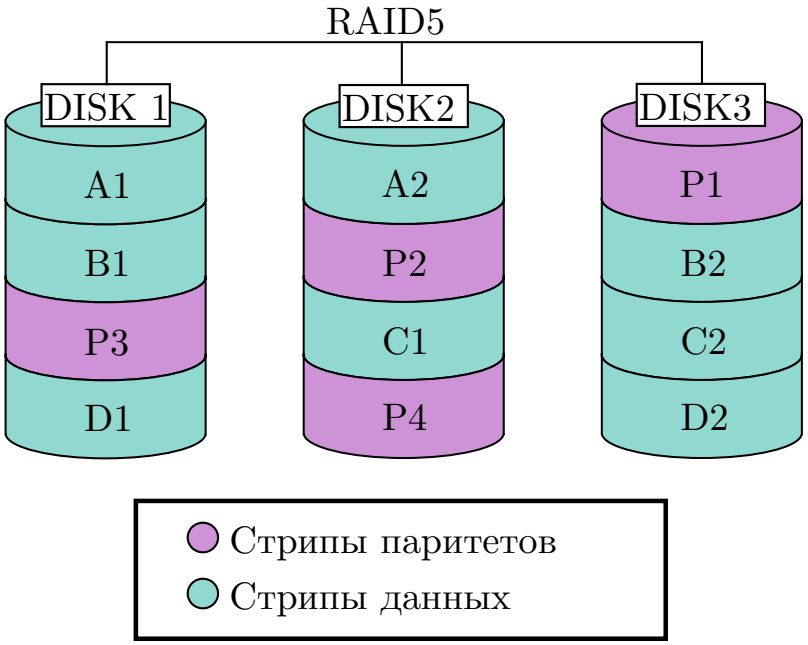

Рис. 4: RAID5 из трех базовых устройств.

2.2 Обзор существующих решений

В данный момент одними из самых популярных открытых программных реализаций RAID1 и RAID5 являются решения внутри Linux kernel *(mdraid)* и OpenZFS.

2.2.1 MDRAID

Реализация является частью ядра Linux, поэтому обработка IO-запросов к блочным устройствам происходит также в пространстве ядра. Это в свою очередь может снижать производительность из-за переключения пользовательского контекста в контекст ядра.

Еще одним недостатком этой реализации RAID5 является использование глобальных блокировок. Они используются во время вычисления паритетов, а также для избегания несогласованности паритетов и данных в страйпе после неожиданной перезагрузке RAID5 во время обработки запроса на запись. Такая проблема несогласованности называется «write hole» [9]. Конкретная реализация RAID5 контролирует этапы записи страйпов с помощью таблиц счетчиков. Счетчики обновляются по мере того, как обновляется информация в самом страйпе. Все таблица в фоновом режиме загружается на базовые устройства, и в случае неожиданной перезагрузки может быть использована для определения несогласованного страйпа. Для того чтобы избежать конкуренции в обновлении счетчиков RAID5 использует глобальные блокировки. Все вышеперечисленное приводит к снижению производительности [12].

2.2.2 RAIDZ

OpenZFS не содержит в себе реализацию RAID5 и RAID1, однако присутствуют прямые аналоги: RAIDZ и Mirrored Pool. Важной особенностью этой реализации является использования технологии «copy on write». Ее главная идея заключается в том, что при чтении используется одни и те же данные, в то время как при записи создается новая отдельная копия. Именно технология «copy on write» позволяет RAIDZ избегать проблемы «write hole». Еще одной особенностью является отсутствие страйпов фиксированного размера. Один страйп равняется логической единице файловой системы, поэтому его ширина может быть переменной. Иными словами, размер страйпа соответствует некоторой записи на файловую систему [8].Эта особенность позволяет отправлять запросы на запись на меньшее количество дисков. Так происходит, потому что RAIDZ используется вместе с ZFS [4]. Более того, ZFS позволяет также использовать ее возможности, как менеджера томов. Таким образом RAIDZ соединяет внутри себя RAID-контроллер, менеджер томов и файловую систему. Реализация RAID в OpenZFS, как и в случае MDRAID, запускается и взаимодействует с базовыми устройствами из пространства ядра.

2.3 Фреймворк SPDK

SPDK*(Storage Performance Development Kit)* — это фреймворк на языке Си, разработанный компанией Intel [1], с открытым исходным кодом, используемый для взаимодействия с блочными устройствами. Основным преимуществом этого фреймворка являются перемещение драйверов блочных устройств из пространства ядра в пространство пользователя. Такой подход позволяет сократить количество системных вызовов, нужных для взаимодействия с устройствами. Более того, буфера для обработки IO-запросов могут напрямую выделяться и заполняться пользовательскими процессами без дополнительного копирования данных. Стоит также отметить, что SPDK использует механизм опроса блочных устройств вместо модели прерываний. В свою очередь модель прерываний способна дополнительно нагружать систему, снижая производительность [2]. Так происходит из-за потребности в обработке прерываний, генерируемых блочными устройствами. Другой важной особенностью фреймворка является избегание блокировок на пути обработки данных. Вместо этого используется концепция передачи сообщений. Смысл этой концепции заключается в том, что во многие функции, отвечающие за обработку IO-запросов, в качестве аргумента может быть передана функция завершения, что в свою очередь позволяет избежать использование запросов с ожиданием какого-то события.

2.3.1 Особенности выделения памяти

Из-за того, что драйвера SPDK выполняются в контексте обычного процесса пользовательского пространства, появляются некоторые трудности с выделением памяти для буферов, используемых NVMe [10] устройствами. Так происходит, потому что NVMe применяет технологию DMA*(Direct Memory Access)* [10] для передачи данных. Иными словами, корректное взаимодействие с базовыми устройствами подразумевает под собой работу со статическими физическими адресами. В то время как драйверам SPDK доступны только виртуальные. Решением этой проблемы в Linux является проверка `/proc/self/pagemap` [6]

изнутри процесса с драйверами. Этот файл содержит в себе информацию о сопоставление виртуальных страниц физическим. Однако ОС не дает гарантий на то, что в следующий момент времени после прочтения `/proc/self/pagemap` карта сопоставления не изменится из-за оптимизаций, осуществляемых операционной системой, или из-за подкачки физических страниц на диск [7]. Возможным решением является размещение буферов для DMA на «больших» страницах — 2 мб. Linux гарантирует, что физическое расположение этих страниц не изменится. Поэтому в SPDK для выделения буферов используется функция `spdk_dma_malloc()` [11] или ее аналоги, а не обычный `malloc()`.

3 Процесс востановления данных в SPDK

На данный момент RAID1 внутри SPDK содержит свою реализация процесса восстановления данных. Далее будет подробно описаны особенности этой реализации.

3.1 Интерфейс

Процесс рибилда организуется в рамках интерфейса `process`. Сам интерфейс является достаточно общим чтобы использоваться не только в процессе восстановления. Однако на данный момент другие применения этого интерфейса не реализованы.

Стоит также обратить внимание, что каждый запуск реализации этого интерфейса происходит в отдельном потоке (для каждого запуска выделяется новый поток). Это потенциально может сказаться на общей производительности и стать преградой в реализации эффективного процесса восстановления отдельных диапозонов адресов внутри конкретного базового устройства.

3.2 Процесс восстановления на уровне RAID

Запуск процесса восстановления происходит в момент добавления нового базового устройства в деградированный RAID-массив. Точечное восстановление поврежденных областей внутри функционирующего устройства в SPDK не реализовано. Причем критерием запуска является отсутствие на добавляемом устройстве блока метаданных.

На данный момент блок метаданных представляет из себя набор статических полей, содержащих в себе метаинформацию о параметрах RAID-массива. Например, размер стрипа, размер блока, уровень RAID-массива. Важно отметить, что блок метаданных не содержит в себе временных меток. Следовательно проверить актуальность данных, хранимых на базовом устройстве, которое было извлечено из RAID-массива, невозможно.

Таким образом, из-за особенностей метаданных, текущая реализация ребилда в SPDK не позволяет запускать процесс восстановления данных на базовых устройствах, ранее входящих в состав RAID-массива.

Еще одной немаловажной особенностью является блокировка диапазона адресов, подлежащих восстановлению. Этот диапазон задается статически макросом $WINDOW\_SIZE\_KB\_DEFAULT$ и никак не зависит от характеристик конкретного RAID-массива.

3.2.1 Вывод

Таким образом текущая реализация процесса восстановления данных внутри RAID-массива в SPDK не может обеспечивать требуемый уровень отказоустойчивости из-за ряда вышеперечисленных недостатков таких, как проверка временных меток в метаданных, использования глобальных блокировок статического диапазон адрессов внутри RAID-массива в момент восстановления и отсутствие возможности точечного ребилда.

4 Реализация

Далее будут описаны детали реализации процесса восстановления данных в RAID1.

Важно отметить, что реализация опирается на уже существующую функциональность, отвечающую за контроль актуальности хранимых данных внутри RAID-массива. Эта функциональность сохраняет информацию о поврежденных областях памяти. Область памяти представляет собой объеденение нескольких подряд идущих стрипов на одном базовом устройстве. Страйпом областей памяти называется множество соответствующих областей памяти, расположенных на каждом базовом устройстве. Иными словами все области памяти с индексом i представляют из себя один подобный страйп.

Процесс востановления можно разделить на две основные части. Во-первых, общая функциональность, которая не зависит от уровня RAID. Эта часть, запускаясь с определенным временным интервалом, следит за изменениями в актуальности хранимых данных, а также запускает и контролирует сам процесс восстановления. Во-вторых, функциональность, специфичная для конкретного уровня RAID. Именно она выступает за непосредственную обработку отдельных поврежденных областей памяти.

4.1 Используемая функциональность

Нужная для процесса восстановления информация содержится в структуре `raid_rebuild`, в часности используются поля `rebuild_matrix` и `rebuild_flag` (см. рис. 5).

Первое поле содержит в себе массив, состоящий из атомарных типов [10], и применяется для хранения информации об актуальности данных. Каждый элемент из `rebuild_matrix` отвечает за один страйп областей памяти. Так если хотя бы один стрип, принадлежащий области памяти, является поврежденным, то вся эта область помечается, как недоступная для записи и чтения. При этом для контроля используется побитовый доступ: каждый бит в элементе `rebuild_matrix` отвечает

за область памяти на соответствующем базовом устройстве. Если бит равен единице, то область считается поврежденной.

Поле `rebuild_flag` также является атомарным. В нем сохраняется информация о флагах, используемых для контроля актуальности хранимых в RAID-массиве данных.

4.2 Архитектура

Архитектура решения опирается на поллинговый механизм, котрый предоставляет фреймворк SPDK. Основная идея этого механизма заключается в повтороном исполнении одной и той же передаваемой функции с определенным временным промежутком. Однако поллер сам по себе не способен сохранять состояния, поэтому для организации процесса восстановления были разработаны две вспомогательные структуры `rebuild_progress` и `rebuild_cycle_iteration` (см. рис. 5).

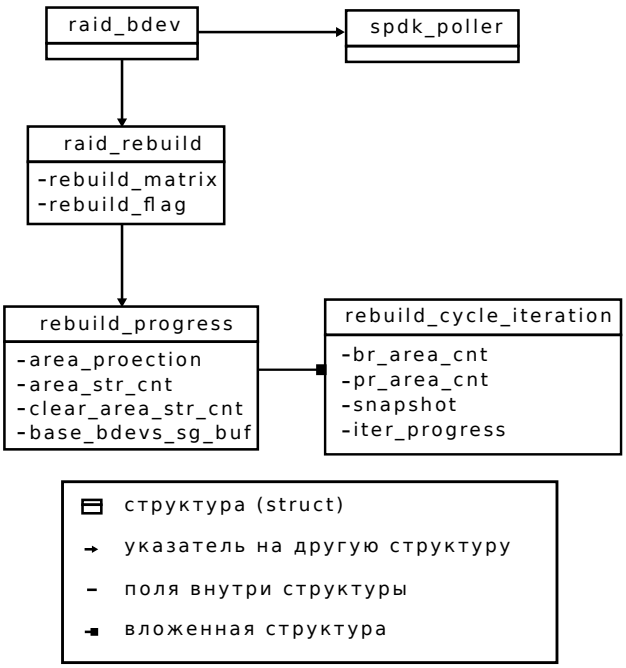

Рис. 5: Диаграмма отношения структур, используемых в процессе восстановления.

4.2.1 `rebuild_progress`

Так как поломки областей памяти случаются недетерминированно, процесс восстановления, циклически проходясь по элементам массива

`rebuild_matrix` может создавать постоянную нагрузку на систему. Механизм поллинга, позволяет избежать этой проблемы, однако возникает необходимость определять начало и конец процесса восстановления, при условии недетерминированного появления новых поврежденных областей. Для этого используется система снэпшотов.

При инициализации очередного цикла восстановления в битовый массив `area_proection` записывается проекция `rebuild_matrix`. При этом единица на i-ом бите внутри `area_proection` обозначает наличие хоть одной единицы в i-ом элементе `rebuild_matrix`. Таким образом происходит фиксация тех страйпов областей памяти, которые будут подвержены восстовлению в текущем цикле. Поврежденные области памяти, не попавшие в зафиксированные, будут обработаны в следующем цикле восстановления. В качестве оптимизации количество единиц в `area_proection` записывается в `area_str_cnt`. А число обработанных страйпов областей сохраняется в поле `clear_area_str_cnt`. Изначально это поле инициализируется нулем. Цикл восстановления считается завершенным в тот момент, когда `clear_area_str_cnt` становится равным `area_str_cnt`. Поле `base_bdevs_sg_buf` содержит указатель на выделяемый при инициализации цикла восстановления вспомогательный буфер. Он в свою очередь выделяется через функцию `spdk_dma_zmalloc()` (см. п. 2.3.1).

4.2.2 `rebuild_cycle_iteration`

Чтобы снизить нагрузку на систему во время процесса восстановления, было принято решение использовать константный буфер, не превышающий размер одного страйпа областей. Из-за этого в единицу времени процесс восстановления может обслуживать только один подобный страйп — итерация процесса восстановления. Структура, описывающая такую итерацию называется `rebuild_cycle_iteration` и располагается внутри `rebuild_progress`. Также структура инициализируется каждый раз, когда происходит переход на следующую итерация. При этом в поле `snapshot` создается копия элемента `rebuild_matrix`, соответствующего страйпу областей памяти, который будет восстанавливаться во

время итерации. Индекс этого элемента сохраняется в `iter_progress`. Количество поврежденных областей, определяется по сделанному снэпшоту и записывается в поле `br_area_cnt`. Итерация считается завершенной в тот момент, когда было совершена попытка восстановить все поврежденные области, описываемые полем `snapshot`. Попытка восстановления вне зависимости от результата считаются в атомарном поле `pr_area_cnt`. Итерация заканчивается, когда `pr_area_cnt` становится равным `br_area_cnt`. Более подробно расположение полей схематично изображено на рис. 5.

Поле `pr_area_cnt` является разделяемым ресурсом, так как обработка областей в одном страйпе может происходить одновременно.

4.3 Общей функциональность процесса восстановления

Общая функциональность, вынесенная в файл `service.c`, основывается на двух функциях: `continue_rebuild()` и `run_rebuild_poller()`.

`run_rebuild_poller()` — функция, которая запускается внутри поллера при инициализации конкретного RAID-массива. Именно она отвечает за запуск нового цикла восстановления, а также за его завершение вместе с освобождением ресурсов. Так, при прохождении всех используемых элементов из `rebuild_matrix`, заполняется `area_proection`. В случае, если встречается хоть одна единица, запускается новый цикл восстановления с инициализацией структуры `rebuild_cycle_iteration` и выделением буфера. Более подробная блок-схема представлена на рис. 6

Важно заметить, что функция — `rebuild_request()`, осуществляющая процесс восстановления для конкретного уровня RAID вызывается также внутри `run_rebuild_poller()`. Поэтому подобная функция должна быть полиморфной, так как реализация процесса восстановления отличается в зависимости от уровня RAID. В SPDK такой полиморфизм достигается через использование константной структуры - `raid_bdev_module`, содержащей внутри себя указатели на функции.

Эта структура заполняется уникальным образом для каждого уровня RAID.

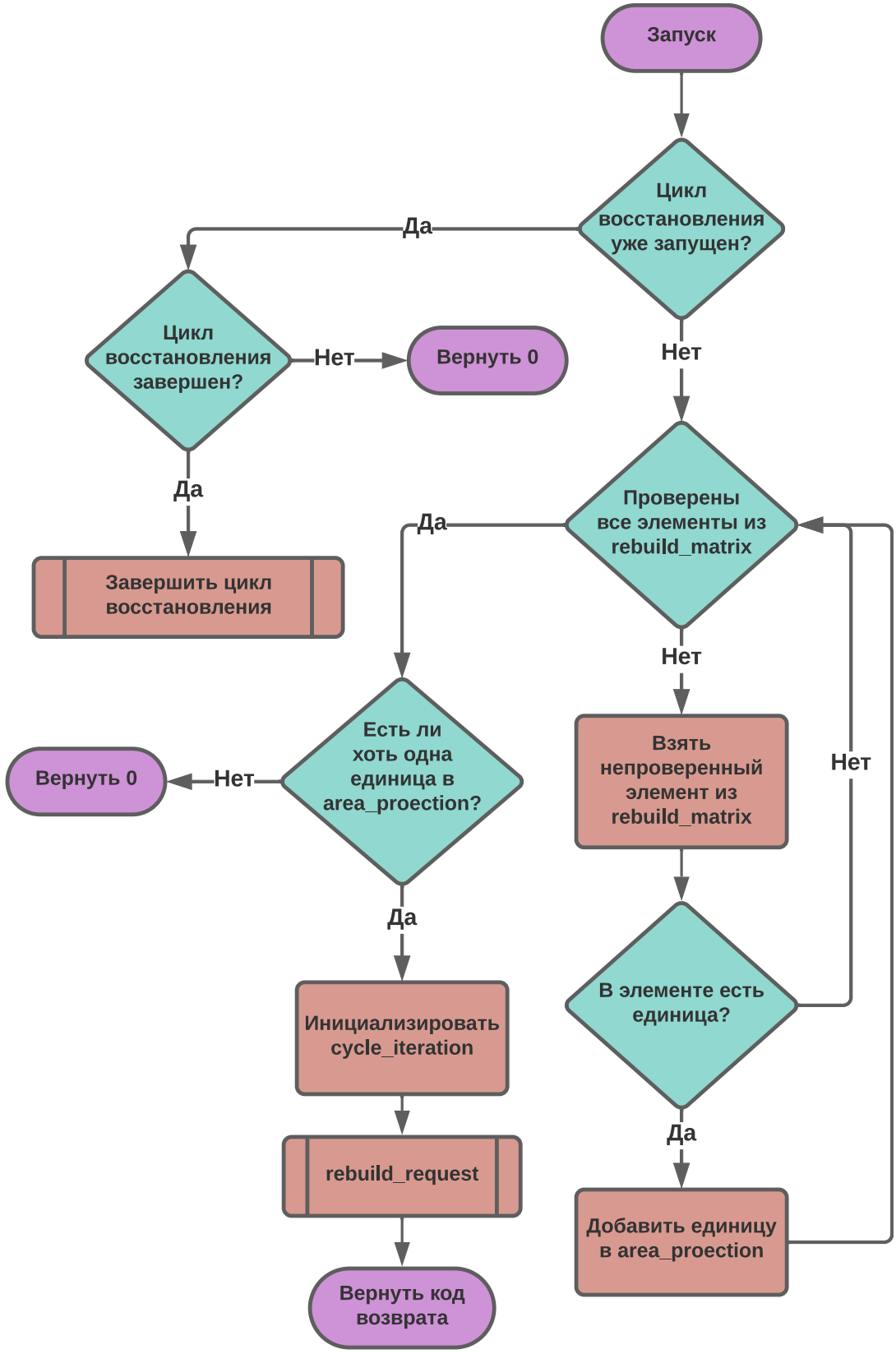

Рис. 6: Блок схема `run_rebuild_poller()`.

`continue_rebuild()` — выступает в роле функции «callback». Иными словами, это функция, которая запускается после завершения каждой итерации и инициализирует новую. Если завершилась последняя итерация, то `continue_rebuild()` выставляет флаг завершения процесса `REBUILD_FLAG_FINISH`, обрабатываемый в дальнейшем внутри `run_rebuild_poller()`. Более подробно см. рис. 7.

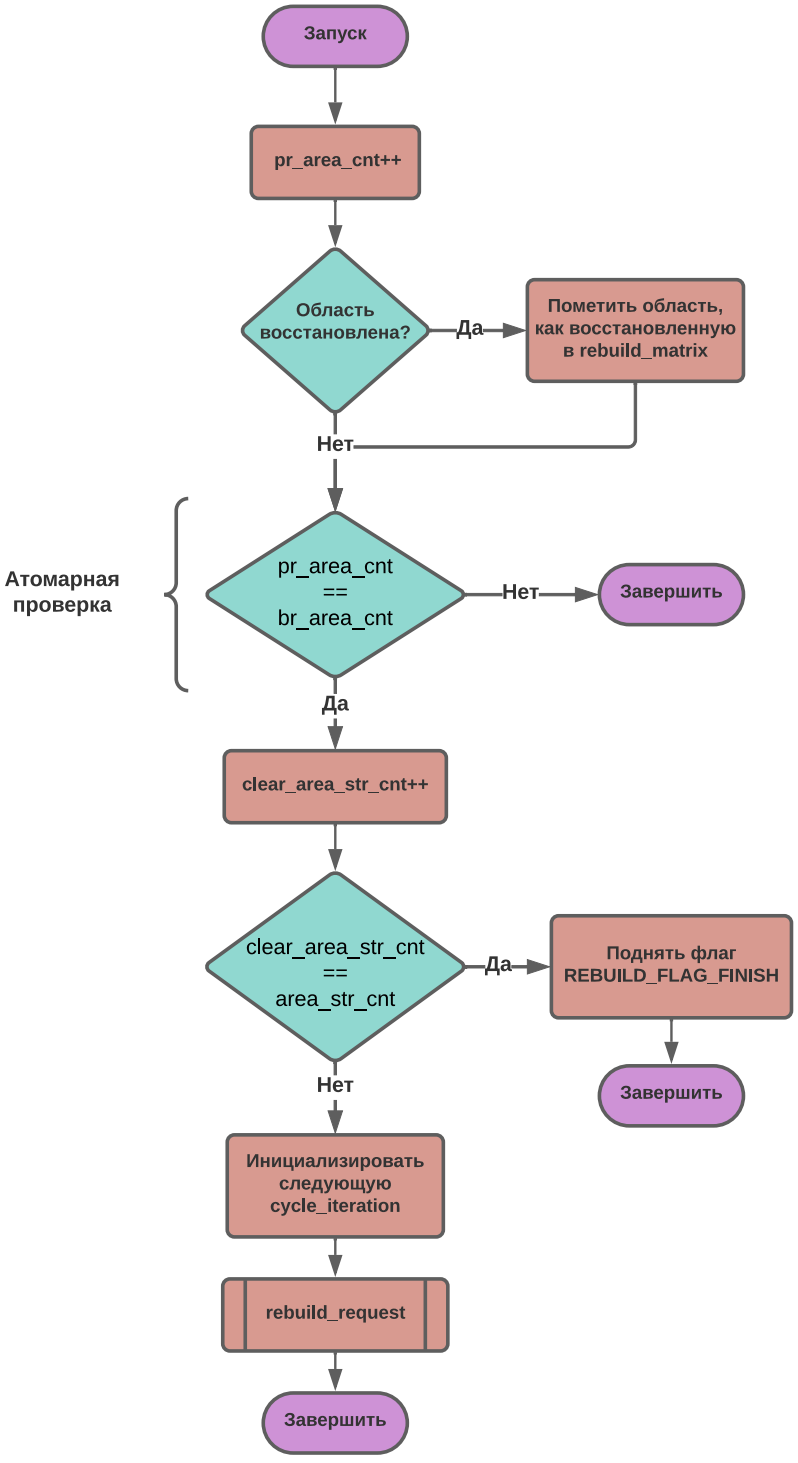

Рис. 7: Блок схема `continue_rebuild()`.

4.4 Процесс восстановления RAID1

Реализация функциональности, отвечающая за процесс восстановления для RAID1 может быть разбита на две основные стадии. Во-первых, `raid1_submit_rebuild_request()`, которая в `raid_bdev_module` соответствует вызову `rebuild_request()`. Эта функция запускает чтение актуальной области памяти из текущего страйпа областей, если такая область памяти не существует, то выставляется флаг ошибки процесса `REBUILD_FLAG_FATAL_ERROR`. В дальнейшем полученные данные используются для восстановления поврежденных областей в том же страйпе (см. рис. 2.1.1). Если запрос на чтение завершается успешно, то вызывается функция `raid1_submit_rebuild_second_stage()`. Она в свою очередь создает запросы на запись полученных данных в поврежденные области. После их обработки вызывается `continue_rebuild()`. Важно заметить, что запросы на запись могут обрабатываться в том числе параллельно, поэтому `continue_rebuild()` внутри обновляет атомарные счетчики.

5 Тестирование

Тестировани внутри фреймворка SPDK основывается на bash скриптах, поэтому тесты функциональности процесса восстановления RAID1 также используют этот инструмент. Важно обратить внимание, что проводилось именно интеграционное, а не unit тестирование. Такой тип был выбран из-за тесной связи процесса восстановления с работой RAID1. Тесты основывались на утилите fio [3], которая позволяет генерировать запросы чтения и записи на блочное устройство без использования файловой системы. Целью тестирования являлась проверка корректности восстанавливаемых данных.

5.1 Сценарий тестирования

1. Создавался RAID1-массив на основе виртуальных блочных устройств.
2. Из массива убиралось нулевое базовое устройство (устройство, с нулевым индексом, входящее в состав RAID1).
3. С помощью утилиты fio производилась запись на деградированный RAID1.
4. На место убранного устройства добавлялось новое.
5. Из RAID1 извлекались все остальные базовые устройства.
6. Через утилиту fio проводилась верификация данных, хранящихся в RAID1.

При таком сценарии тестирования изначально записанные в RAID1 данные с помощью процесса восстановления оказывались на базовом устройстве, которое было добавленно в массив уже после записи этих данных в RAID.

Стоит добавить, что для реализации подобного сценария тестирования требовалось наличие функциональности по добавлению и извлечению базового устройства из запущенного RAID-массива. Однако в

текущей версии SPDK, на основе которой реализовывался процесс восстановления, отсутствовала функциональность, добавляющая устройство. Из-за этого было принято решение провести ее интеграцию. За основу было взято решение[1], реализованное для той же версии SPDK.

[1]Реализация доступа по ссылке.

Заключение

В ходе выполнения данной работы были достигнуты следующие результаты.

- Изучена реализация RAID-массивов внутри фреймворка SPDK.
- Описан обзор процесс восстановления данных в RAID1 на основе SPDK.
- Спроектирована архитектура процесса восстановления, соответствующая принятым в SPDK соглашениям.
- Реализована функциональность, отвечающая за процесс восстановления данных в RAID1 без возможности параллельной обработки запросов на чтение и запись[2].
- Проведено интеграционное тестирование корректности RAID1 с учетом добавленной функциональности.
- Реализованы автоматизированные тесты для проверки корректности восстанавливаемых данных.

В дальнейшем планируется добавить в процесс восстановления для RAID1 возможность параллельной обработки IO-запросов; а также реализовать процесс восстановления данных для RAID5.

[2]Реализация доступа по ссылке.

Список литературы

[1] Accelerating your NVMe drives with SPDK. — URL: https://spdk.io/doc/about.html (дата обращения: 2023-12-14).

[2] Accelerating your NVMe drives with SPDK. — 2016. — URL: https://habr.com/ru/companies/intel/articles/315906/ (дата обращения: 2023-12-14).

[3] Fio documentation. — 2023. — URL: https://fio.readthedocs.io/en/latest/index.html (дата обращения: 2023-12-14).

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

[5] 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: 2023-12-14).

[6] The Linux Kernel documentation, Examining Process Page Tables. — URL: https://www.kernel.org/doc/html/v4.18/admin-guide/mm/pagemap.html (дата обращения: 2023-12-14).

[7] The Linux Kernel documentation, Swap Management. — URL: https://www.kernel.org/doc/gorman/html/understand/understand014.html (дата обращения: 2023-12-14).

[8] The OpenZFS documentation, RAIDZ. — 2023. — URL: https://openzfs.github.io/openzfs-docs/ (дата обращения: 2023-12-14).

[9] Patterson David A, Gibson Garth, Katz Randy H. A case for redundant arrays of inexpensive disks (RAID) // Proceedings of the 1988 ACM SIGMOD international conference on Management of data. — 1988. — P. 109–116.

[10] The SNIA Dictionary. — 2023. — URL: https://www.snia.org/education/dictionary/about-dictionary (дата обращения: 2023-12-14).

[11] The SPDK documentation, Direct Memory Access (DMA) From User Space. — 2023. — URL: https://spdk.io/doc/memory.html (дата обращения: 2023-12-14).

[12] 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.