Санкт-Петербургский государственный университет

Кафедра системного программирование

Группа 22.Б15-мм

Добавление метаданных в SPDK RAID

Кузнецов Арсений Николаевич

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
Старший преподаватель каф. СП, Я. А. Кириленко

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

Санкт-Петербург
2023

Оглавление

Введение

Количество информации, хранимой в цифровом виде, увеличивается быстрее и быстрее с каждым годом. Появляются новые виды устройств хранения данных. Также растет и спрос на сохранность данных, так как все больше важной информации хранится в цифровом виде. Одним из видов накопителей являются блочные устройства, обмен данными с которыми осуществляется блоками данных. Объединение нескольких блочных устройств в одно может как повысить скорость доступа к данным, так и обеспечить высокую отказоустойчивость, при добавлении избыточной информации.

Одним из способов объединения блочных устройств является технология RAID (Redundant Array of Independent Disks) [7], которая может повысить скорость доступа к данным, а также обеспечить избыточность хранимой информации, что позволяет восстановить информацию в случае повреждения определенного количества блочных устройств. Максимальное количество блочных устройств, которые могут выйти из строя без потери информации, зависит от типа RAID.

Одна из реализаций технологии RAID находится в SPDK (Storage Performance Development Kit) [6]. SPDK — это программное обеспечение для Linux и FreeBSD, выпущенное компанией Intel. Главным преимуществом SPDK является высокая производительность, которая обеспечивается за счет расположения всех необходимых драйверов в пользовательском пространстве Linux, что помогает избежать частых прерываний процессора, а также отсутствия блокировок путей ввода/вывода и оптимизации для платформы Intel.

Один из минусов технологии RAID в SPDK — это отсутствие функциональности присутствующей в аналогах, в частности отсутствуют метаданные для RAID, следовательно невозможно восстановление массива с накопителей, из которых состоял RAID, что сильно сказывается на удобстве пользователя и отказоустойчивости, так как в случае перезагрузки нет возможности восстановить массив.

1 Постановка задачи

Целью работы является добавление в SPDK функциональности восстановления RAID с метаданных его накопителей после перезагрузки.

Для её выполнения были поставлены следующие задачи:

1. изучить реализацию метаданных RAID в ядре Linux;
2. реализовать в SPDK RAID метаданные, их запись при инициализации и восстановление с блочных устройств;
3. протестировать реализованную функциональность.

2 Обзор

2.1 Технология RAID

Как уже было сказано ранее, RAID представляет из себя технологию для объединение нескольких блочных устройств в одно, далее такие устройства будут еще называться базовыми. Это обеспечивает прирост производительности, а также отказоустойчивости в зависимости от того, какой тип RAID выбран. На данный момент самыми популярными видами RAID являются:

- RAID 0 — массив с чередованием. В нем данные делятся на части — стрипы (strip) [7] и равномерно распределяются по блочным устройствам. Это обеспечивает высокую производительность без избыточности.

- RAID 1 — массив с зеркалированием. Изображен на рис. 1. Стрипы дублируются на каждое базовое блочное устройства. Может обеспечить прирост в скорости, а также высокую отказоустойчивость за счет большого количество используемой памяти.

- RAID 5 — массив с чередованием и контролем четности. В нем стрипы данных вместе со стрипом контрольной суммы циклически записываются на все базовые блочный устройства. Контрольная сумма является результатом операции xor для массива стрипов, что позволяет за счет коммутативности и самообратимости xor восстановить данные в случае повреждения одного из стрипов. RAID 5 имеет приемлемую производительность и проигрывает по отказоустойчивости RAID 1, но имеет гораздо большую вместимость при том же количестве базовых устройств.

- RAID 6 — RAID с чередованием и контролем четности. Похож на RAID 5, но использует два стрипа контрольной суммы. Наихудший по производительности из представленных выше, однако позволяет восстановить массив при повреждении вдвое большого количества стрипов, по сравнению с RAID 5.

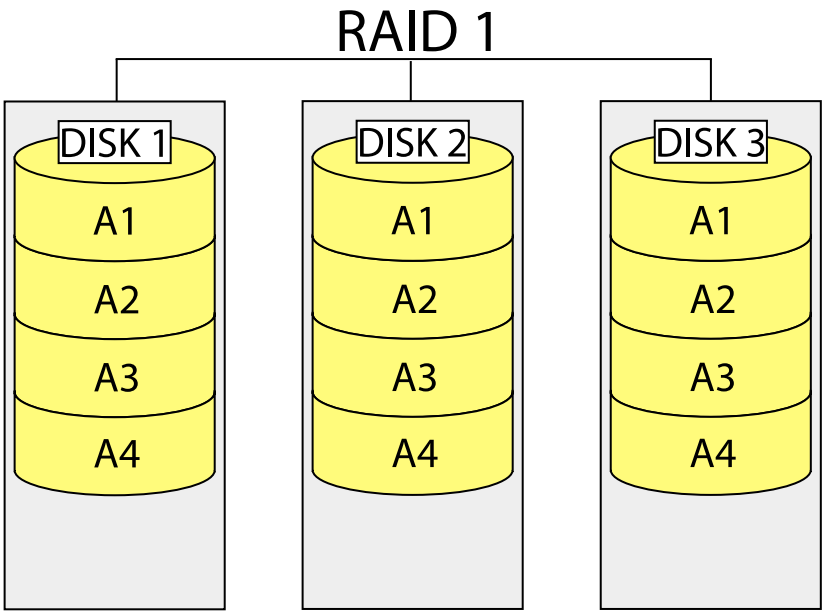

Рис. 1: Пример RAID типа RAID 1 из трех дисков

В случае некритичного повреждения данных на базовых блочных устройствах RAID переходит в degraded состояние [7]. В таком случае массив восстанавливает свою целостность с помощью процесса, именуемого ребилдом (rebuild) [7]. Стоит заметить, что ребилд отсутствует для RAID 0, так как в нем нет избыточности.

При работе с RAID пользователь может захотеть изменить тип массива или изменить количество базовых блочных устройств. Процесс позволяющий это сделать называется реконфигурацией (reshape) [3]. В ходе нее массив изменяет свою структуру, при этом не разрушаясь.

В RAID запись ведется сразу на базовые блочные устройства, что не является безопасным, так как в случае аварийного отключения данные могут не полностью записаться в стрип, что сделает данные, находящиеся в стрипе, некорректными. Если массив находился в degraded состоянии, то это может привести к повреждению всего RAID. С этой проблемой может помочь журнальное устройство, запросы на которое поступают перед тем как отправиться на RAID. Следовательно при аварийном отключении можно будет восстановить работоспособность массива по информации с журнального устройства.

2.2 Обзор существующих решений

Одним из аналогов SPDK RAID, с реализованными метаданными и восстановлением массива по ним, является технология RAID в драйверах ядра Linux (далее Linux RAID). На данный момент существует две версии метаданных: стандартная и расширенная 1.9.0. Будем рассматривать расширенную версию.

Метаданные Linux RAID создаются при инициализации RAID и записываются для всех базовых блочных устройств в суперблоки (superblock) [5]. Суперблок — это структура в Linux, в которой хранятся метаданные блочных устройств. При операциях, например, чтении/записи, с RAID метаданные обновляются.

После перезагрузки массив можно пересоздать, если метаданные Linux RAID есть хотя бы на одном базовом блочном устройстве. Если они присутствуют на нескольких устройствах, метаданные читаются с последнего обновленного блочного устройства, оно определяется по специальному счетчику в метаданных. На основе метаданных с этого базового устройства восстанавливается массив, и создаются метаданные для других базовых устройств.

Метаданные Linux RAID можно поделить на три части: основные метаданные, метаданные для ребилда, метаданные для реконфигурации. Рассмотрим основные метаданные:

- `magic` — магическое число для распознавания метаданных, "DmRd" в ASCII кодировке;
- `compat_features` — версия метаданных;
- `num_devices` — количество базовых блочных устройств RAID;
- `array_position` — позиция базового блочного устройства в массиве, чтобы узнать его роль;
- `events` — количество событий произошедших с блочным устройством, необходимо чтобы узнать последний обновленный базовый накопитель;

- `level` — тип RAID;
- `layout` — подтип RAID;
- `stripe_sectors` — размер стрипа в секторах (512 Байт в Linux);
- `array_sectors` — размер RAID в секторах;
- `sectors` — размер базового блочного устройства в секторах.
- `incompat_features` — неподдерживаемые функции, на данный момент не используются.

Эти метаданные были взяты за основу для построения метаданных SPDK в последствии.

2.3 Устройство SPDK

Для удобства пользователя SPDK поделен на две части: клиент и сервер. Где клиент представляет из себя программу на python, в которой реализован CLI (command-line interface) и вызов соответствующей функциональности на стороне сервера, написанного на языке C. Общение происходит через JSON-RPC 2.0 [1] — это протокол RPC (remote procedure call) [2], который позволяет вызывать процедуры в другом адресном пространстве, что дает возможность запускать SPDK на удаленном узле или использовать стороннее программное обеспечение для управления SPDK.

На стороне сервера основная часть реализации делится на `lib` и `module`. `lib` — содержит базовые библиотеки, составляющие SPDK, а `module` содержит конкретные реализации соответствующих базовых библиотек из `lib`. Так RAID является частью модуля с реализацией для `spdk_bdev` библиотеки.

`spdk_bdev` предоставляет функциональность для создания блочных устройств, операций ввода/вывода с ними, а также множество другой полезной функциональности, связанное с блочными устройствами.

Рассмотрим структуру модуля RAID. Он включает:

- Заголовочный файл `bdev_raid.h`.
- `bdev_raid_rpc.c`, содержащий функции вызываемые через RPC и выполняющий первичную обработку поступающих данных.
- `bdev_raid.c` с реализацией общей для всех типов RAID функциональности.
- Файлы с реализацией специфичной для разных видов RAID функциональности. На данный момент реализованы только RAID 0 и RAID 1, также ведется работа над RAID 5.

3 Описание реализации

3.1 Серверная часть

3.1.1 Необходимые данные

Рассмотрим данные, которые будут сохраняться в SPDK RAID для обеспечения функциональности восстановления массива. Самое важное — метаданные, по информации с которых и восстанавливается RAID, представляют из себя следующее

- `magic` — магическое число, чтобы опознавать метаданные SPDK. ”SKRD” в ASCII.
- `version` — версия метаданных, для версионирования.
- `blocklen` — размер блока базового блочного устройства в Килобайтах. Эта информация нужна по причине того, что в SPDK для многих данных размер измеряется в блоках, поэтому полезно знать размер блока для определения абсолютного, а не относительного размера.
- `num_base_bdevs` — количество базовых блочных устройств. Нужно, чтобы при восстановлении определить не утеряно ли какое-то базовое блочное устройство.
- `level` — тип RAID.
- `array_position` — позиция базового блочного устройства в RAID. Требуется, чтобы узнать роль базового устройства в RAID.
- `strip_size` — размер стрипа в блоках.
- `blockcnt` — количество блоков, содержащихся на базовом блочном устройстве.
- `raid_blockcnt` — количество блоков, содержащихся во всем RAID.

- `timestamp` — время последнего обновления базового блочного устройства. Timestamp, а не счетчик событий как в Linux RAID, был выбран по причине того, что в SPDK не ведется счетчик событий произошедших с RAID.

- `uuid` — уникальный идентификатор RAID.

Также была модифицирована структура `raid_bdev`, которая содержит информацию о RAID, туда было добавлено поле `is_new` для определения того, нужно ли создавать метаданные или восстанавливаться по ним. Еще была модифицирована структура `raid_base_bdev_info`, описывающая базовое блочное устройство в составе RAID. Туда было добавлено поле `is_new`, для определения того были ли метаданные на базовом накопителе при инициализации RAID.

3.1.2 Реализация

При инициализации RAID возможно два случая либо массив восстанавливается с базовых блочных устройств, если там есть метаданные, либо он создается на основе каких-то параметров и перезаписывает метаданные, хранящиеся на базовых устройствах. И в первом и во втором варианте было решено встраиваться в функцию создания RAID — `rpc_bdev_raid_create` в файле `bdev_raid_rpc.c`. Перезаписывание метаданных взять как поведение по умолчанию, а восстанавливать массив в том случае, если указан специальный флаг под названием `retrieve`.

Решение по объединению инициализации с нуля и восстановления было принято по причине того, что в случае разделения на две функции, реализации бы практически не различались, поэтому в случае восстановления легче создать RAID с неиспользуемыми параметрами и потом их переписать.

В `rpc_bdev_raid_create`, а также в CLI на стороне клиента, уже был реализован параметр отвечающий за включение метаданных (суперблоков), поэтому оставалось добавить только `retrieve` параметр.

Передем непосредственно к реализации. Так как данные на сервер приходят в виде JSON файла, возникает потребность их декоди-

ровать. Декодирование параметра `retrieve` производится за счет декодера `spdk_json_decode_bool`, так как параметр является логическим.

После декодирования параметров в `rpc_bdev_raid_create` идет вызов функции `raid_bdev_create`, которая делает первоначальную настройку RAID, на основе переданных параметров. В частности, там определяется структура модуля RAID. Модуль содержит типоспецифичные данные, такие как ссылки на функции чтения/записи, минимальное количество базовых устройств, при котором RAID может существовать, и так далее. Функциональность по определению модуля и его валидации было решено вынести из `raid_bdev_create` в отдельную функцию `raid_bdev_module_init`, чтобы потом переиспользовать.

Далее, если флаг `retrieve` стоит, то тогда полю `is_new` в структуре `raid_bdev` проставляется значение false. Далее оно будет сигнализировать, если мы находимся в режиме восстановления.

Следующим шагом в `rpc_bdev_raid_create` запускается конфигурация для всех базовых блочных устройств, и в самом конце вызывается функция `raid_bdev_configure`, которая завершает конфигурацию RAID. В эту функцию было решено вставить основную функциональность.

Рассмотрим функциональность по анализу метаданных в `raid_bdev_configure`.

- Часть с определением и валидацией параметров RAID
- Вызов для каждого базового устройства `raid_bdev_base_bdev_super_load`, которая загружает метаданные с диска и определяет последнее обновленное базовое устройство (freshest).
- Потом происходит запуск функции `raid_bdev_base_bdev_super_validate` для последнего обновленного базового устройства. Она в зависимости от переданного параметра либо меняет меняет конфигурацию RAID на ту, которая описывалась в метаданных, либо перезаписывает метаданные на основе характеристик RAID.

- Если идет восстановление, вызывается функция по инициализации модуля `raid_bdev_module_init`, чтобы в случае, когда тип RAID изменился, его модуль тоже поменялся.
- Блок с окончательной инициализацией RAID.
- После него `raid_bdev_base_super_validate` запускается для всех базовых устройств и перезаписывает их метаданные.
- До этого момента загруженные метаданные хранились в буфере, теперь чтобы загрузить их на накопители, для каждого базового накопителя вызывается функция `raid_bdev_base_bdev_write_sb`.

Как видно, в функциональности метаданных был вынужденный разрыв на блок кода, в котором проводится окончательная инициализация RAID. Причиной этому послужило то, что при `retrieve` для восстановившегося массива требуется впоследствии вызов функциональности из этого блока, а для массива, создающегося с нуля, требуется uuid, вычисляемый в этом блоке, чтобы записать его в метаданные.

3.2 Клиентская часть

На стороне клиентской части CLI организован с помощью библиотеки argparse в файле `rpc.py`. Определение вызываемых функций происходит с помощью сабпарсеров (subparsers) [4]. Сабпарсер — это объект в библиотеке argparse, который позволяет разделить функциональность парсинга на подкоманды со своими аргументами. К каждому сабпарсеру привязана своя соответствующая функция в `bdev.py`, которая вызывает реализацию на стороне сервера через RPC.

Для того, чтобы добавить опцию ––`retrieve` в CLI, надо добавить ее как аргумент сабпарсеру, с указанием описания и поведения. В данном случае поведением будет сохранение значения true при указании опции. Далее в файл `bdev.py` надо только добавить `retrieve` в словарь аргументов, а кодирование уже произойдет автоматически.

Так как при указании опции `retrieve` в `bdev_raid_create` большинство параметров впоследствии фактически не используется, при

том что они обязательны для указания, было решено создать обертку (wrapper) под названием `bdev_raid_retrieve` для функции `bdev_raid_create`. Обертка требует только новое имя для восстановленного RAID и блочные устройства, из которых он состоял. Остальные обязательные параметры заполняются валидными значениями, а именно RAID 1 для типа RAID и true для опций `superblock` и `retrieve`. Потом, с этими данными в качестве аргументов, вызывается `bdev_raid_create`.

4 Тестирование

На данный момент из-за нехватки времени проводилось только ручное тестирование, в ходе которого создавался RAID, читались его параметры, самым показательным из которых являлся uuid, так как он уникален для каждого массива. Потом массив разрушался и восстанавливался с помощью `raid_bdev_retrieve`, потом читались параметры восстановленного. По итогу параметры RAID до разрушения и после восстановления совпадали. Протестированные следующие случаи, когда массив должен создаваться корректно:

- Создание RAID 1, с количеством базовых устройств > 1, его удаление и восстановление через retrieve.
- Создание RAID 0, его удаление и восстановление через retrieve.
- Создание RAID 1, с количеством базовых устройств > 1, его удаление и восстановление через create с опцией ––`retrieve` и RAID 0 в качестве типа RAID.

И не корректные случаи:

- Вызов функции `bdev_raid_create` с указанием флага ––`retrieve`, но без флага ––`superblock`
- Вызовом `retrieve` не со всеми базовыми устройствами из RAID в качестве параметров.
- Вызов `retrieve` с блочными устройствами не содержащими метаданные.
- Вызовом `retrieve` с блочными устройствами с метаданными из разных RAID.

Заключение

В ходе проделанной работы на данный момент были достигнуты следующие результаты.

- Изучен код аналога SPDK RAID с реализованными метаданными и восстановлением массива по ним, а именно код Linux RAID.
- Реализованы метаданные для SPDK RAID, восстановление массива по ним, а так же вызов этой функциональности через CLI.
- Добавленная функциональность была частично протестирована.

С реализованной функциональностью можно ознакомиться на GitHub

В рамках весеннего семестра планируется продолжить реализовывать функциональность метаданных для SPDK RAID, а именно.

- Провести полноценное тестирование уже написанной функциональности и написать Unit тесты.
- Реализовать обновление метаданных при операциях над массивом.
- Возможно добавить в метаданные информацию о процессе ребилда, находящемся сейчас в стадии разработки.

Список литературы

[1] Morley Matt. — Спецификация JSON-RPC, 2013. — URL: https://www.jsonrpc.org/specification (дата обращения: 17 декабря 2023 г.).

[2] Thurlow Robert. — Спецификация RPC, 2009. — URL: https://datatracker.ietf.org/doc/html/rfc5531#section-4 (дата обращения: 17 декабря 2023 г.).

[3] гайд по утилите mdadm. — URL: https://raid.wiki.kernel.org/index.php/A_guide_to_mdadm (дата обращения: 18 декабря 2023 г.).

[4] документация argparse. — URL: https://docs.python.org/3/contents.html (дата обращения: 21 декабря 2023 г.).

[5] документация ядра Linux. — URL: https://www.kernel.org/doc/html/latest/index.html (дата обращения: 15 декабря 2023 г.).

[6] официальный сайт SPDK. — URL: https://spdk.io/ (дата обращения: 12 декабря 2023 г.).

[7] словарь терминов SNIA. — URL: https://www.snia.org/education/online-dictionary (дата обращения: 12 декабря 2023 г.).