Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.Б11-мм

Контроль актуальности данных в RAID1 и RAID5 на основе SPDK

Кононова Юлия Алексеевна

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

С каждым годом объемы информации растут. С этой проблемой сталкивается в наши дни большинство компаний, ведь полученную информацию нужно как-то хранить и обрабатывать.

Рост объемов данных привел к появлению Системы хранения данных (СХД). СХД представляют собой сложно организованное блочное устройство, то есть такое устройство, которое позволяет работать с блоками определенного размера. Эти системы являются неотъемлемой частью инфраструктуры в современных компаниях.

Одной из основных технологий в этой отрасли является RAID — технология, которая объединяет несколько блочных устройств в единое пространство хранения. Смысл создания RAID-массива заключается в повышении надежности хранения данных на компьютере, а также в увеличении производительности. Возможности конкретного RAID-массива зависят от его уровня. Например, RAID1 представляет собой массив устройств, каждое из которых является копией друг друга. Широко используемым способом для создания RAID — массивов является программная реализация внутри ядра, но это влечет за собой последующие минусы. Во-первых, сложность разработки, во-вторых, снижение производительность самой СХД, так как первый шаг перехода из пространства пользователя в пространство ядра добавляет дополнительные издержки. Альтернативным решением является фреймворк от Intel — SPDK[16]. Он позволяет взаимодействовать с блочными устройствами из пространства пользователя. Исключение ядра Linux из процесса разработки не только упрощает разработку и отладку, но и увеличивает производительность, что подтверждается отчетом от Intel[15].

В ходе работы системы может возникнуть потребность в замене блочного устройства. Из-за этого пропадает согласованность каких-то блоков памяти. RAID1 и RAID5 в SPDK на данный момент не запоминают информацию о потерявших актуальность блоках, поэтому существует большая вероятность, что данные будут утеряны без возможности восстановления. Таким образом утрачивается главное преимущество со-

здание RAID массива — надежность. Одним из возможных решений для осуществления контроля актуальности хранимых данных является разбиение блочных устройств на непрерывные области памяти, так как хранить информацию о каждом блоке неэффективно. Следует отметить, что области памяти содержат внутри себя несколько блоков, в зависимости от объема блочного устройства. Поэтому в одной области может обрабатываться несколько запросов одновременно. Это ведет за собой необходимость параллельного обновления информации об актуальности хранимых данных в области памяти. В рамках данной работы планируется реализовать функциональность, которая будет отвечать за запись и хранение информации о состоянии блочного устройства. Если какой-то из блоков повредится, можно будет выявить его и в дальнейшем провести процесс восстановления данных.

1 Постановка задачи

Целью данной работы является осуществление контроля актуальности данных в RAID1 и RAID5. Для достижения данной цели были поставлены следующие задачи:

1. выполнить обзор существующих решений и изучить инструменты параллельного программирования, для параллельной обработки запросов ввода и вывода;
2. разработать структуру, содержащую в себе информацию об актуальности хранимых данных внутри RAID;
3. осуществить взаимодействие с реализованной структурой при обработке запросов на чтение и запись в RAID1 и RAID5;
4. провести интеграционное тестирование.

2 Обзор

В данном разделе будут рассмотрены ключевые термины и методы, фреймворк SPDK[1] , над которым будет идти работа, и аналоги процесса ребилда в других реализациях.

2.1 RAID

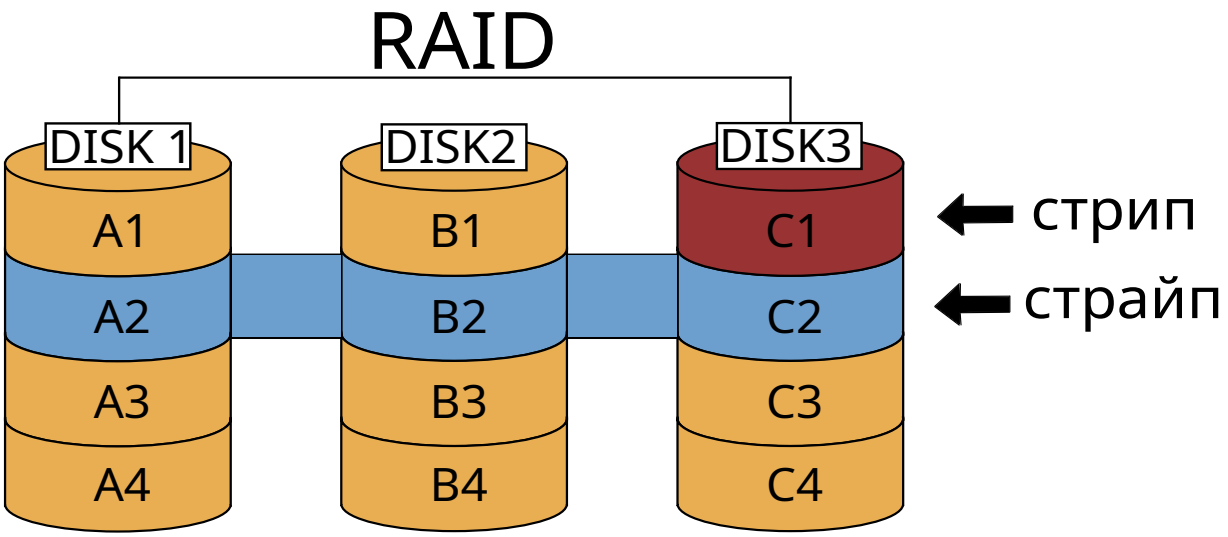

Рис. 1: Стрипы и страйпы в RAID-массиве

RAID (Redundant Array of Independent Devices)[11] — технология, которая объединяет несколько блочных устройств в единое пространство хранения — массив. В этом массиве часть физической емкости используется для хранения избыточной информации о пользовательских данных, которые хранятся на оставшейся части хранилища. Эта избыточная информация помогает, в зависимости от уровня RAID-массива, обеспечить отказоустойчивость и увеличить производительность. Важно отметить, что данные внутри RAID-массива разбиты на страйпы. Страйпы состоят из стрипов – непрерывная область памяти на одном устройстве одинакового размера. Каждое устройство поделено на одинаковое количество стрипов и каждому страйпу соответствуют стрипы с одним индексом. Схематично это разбиение представлено на Рис. 1

[1]Storage Performance Development Kit: https://spdk.io/

Уровни RAID-массивов бывают различными.

2.1.1 RAID1

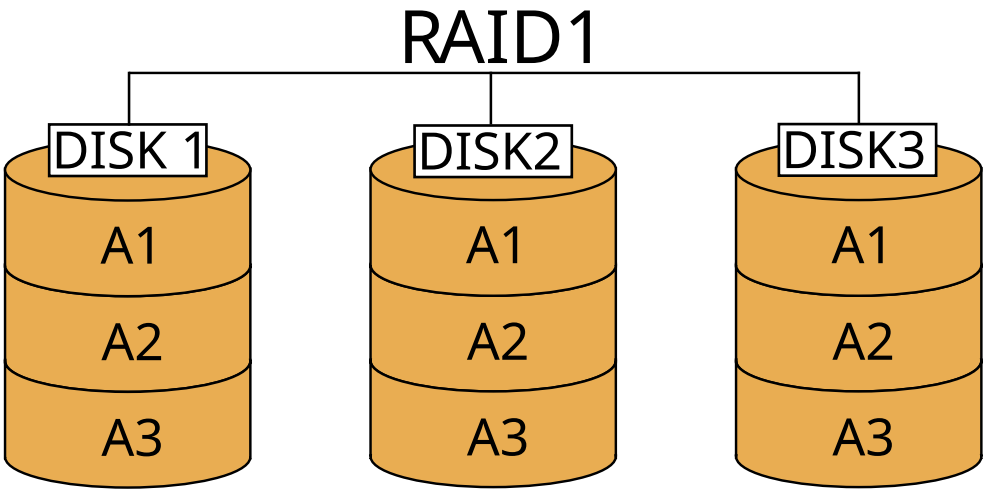

Рис. 2: RAID1

RAID1 использует дублирование или зеркалирование, то есть каждое устройство в таком массиве является полной копией других, как продемонстрировано на Рис. 2. Этот метод обеспечивает высокий уровень надежности хранения данных, поскольку в случае сбоя какого-либо устройства, система сможет продолжить свою работу без перебоев, данные сохранятся на другом устройстве, поэтому неисправное устройство можно будет заменить, а данные восстановить. Плюсом также является увеличение скорости чтения. Данные могут быть получены с любого устройства благодаря параллельному доступу к ним. Однако такая система имеет очевидный минус — малый объем хранения, ведь большая часть потенциальной емкости хранилища автоматически теряется. По сути максимально возможный объем равен минимуму из объемов имеющихся устройств. Поэтому RAID1 полезен для хранения существенно важных данных.

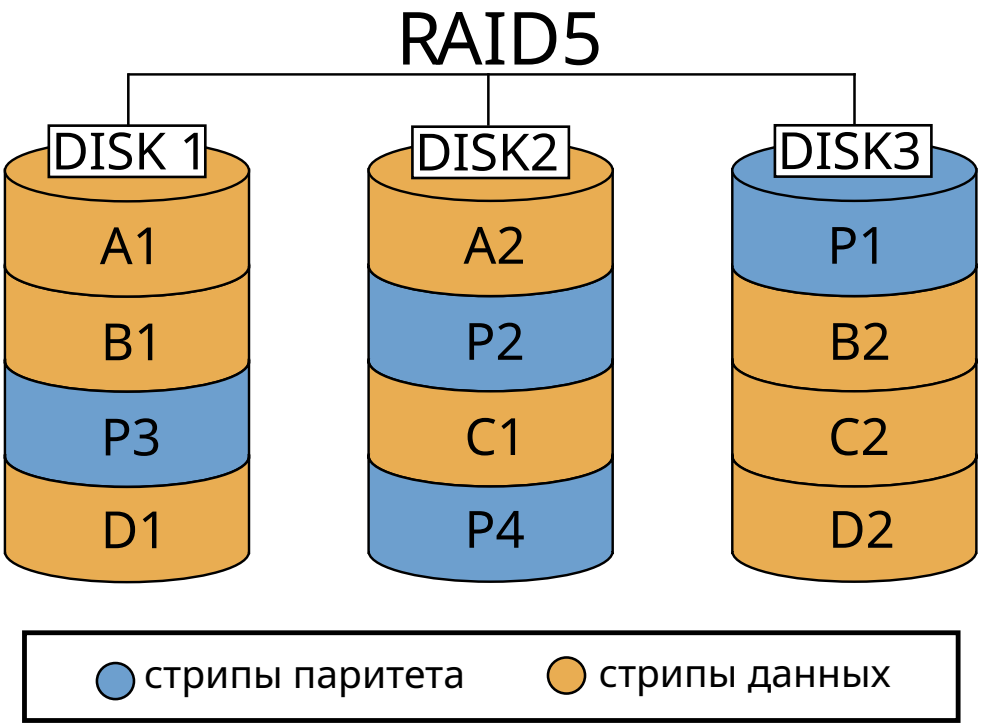

Рис. 3: RAID5

2.1.2 RAID5

RAID5, схематично изображенный на Рис. 3, использует вместо зеркалирования контрольно-восстановительные суммы, называемые паритетами (parity) для избыточности данных. Паритет представляет собой дополнительную информацию в виде контрольной суммы, записываемой на устройство. Эта информация используется в дальнейшем для восстановления потерянных или поврежденные блоков данных. Паритет в RAID5 высчитывается с помощью операции XOR — «исключающее или». Все стрипы данных входящих в один страйп связаны с помощью логики XOR, а результат записан в паритет. При этом данные и parity распределены равномерно по всем дискам, поэтому нагрузка на каждое устройство поделена более сбалансировано, чем в RAID-массивах таких как RAID2, RAID3, RAID4[14]. Одним из преимуществ является эффективное распределение объема всего массива по сравнению с зеркалированием, так как информация о контрольной сумме занимает меньше места. Операция чтения также может выполняться параллельно, что увеличивает скорость обработки этой операцию. При

выходе из строя одного устройства массив может продолжить работу, используя данные parity. Однако надежность в этот момент сильно падает, потому что существует большая вероятность потерять данные, если до окончания восстановления выйдет из строя еще хотя бы одно устройство. Более того для вычисления поврежденных данных требуется считать все стрипы из страйпа за исключением поврежденного и применить к ним операцию XOR. Предварительное считывание стрипов приводит к еще одной проблеме — снижение производительности.

2.2 SPDK

SPDK (Storage Performance Development Kit)[16] — фреймворк с открытым исходным кодом разработанный компанией Intel для улучшения производительности систем хранения данных. Основная цель SPDK — обеспечить разработчикам эффективные средства для создания высокопроизводительных и масштабируемых приложений для хранения данных в пространстве пользователя. Высокая производительность достигается благодаря следующим важным особенностям. Во-первых, драйвера блочных устройств находятся в пользовательском пространстве, что позволяет сократить затраты на переключение контекста ядра. Во-вторых, вместо прерываний в ходе выполнения запроса на чтение или запись используется механизм Poller, который выполняет работу опросника аппаратных средств о завершении работы, что позволяет избежать дополнительную нагрузку на систему.

2.3 Параллельное программирование

Закон Амдала[1] гласит, что при увеличении количества потоков, работающих параллельно, ускорение выполнения программы ограничено долей последовательной части программы. Из этого следует[9], что потоки, которые выполняются параллельно, и имеют блокировки взаимоисключающего доступа, таким образом в действительности выполняются последовательно, менее желательны, чем потоки без блокировок. Поэтому использование операций, выполняемых за один шаг от-

носительно остальных потоков, более желательны чем использование блокировок. Вследствии чего было решено в данной работе применять атомарные переменные.

Атомарной называется переменная, которая позволяет выполнять операции атомарно. Выполнение таких операций либо происходит полностью, либо не происходит вовсе. Это означает, что в многопоточной среде никакой другой поток не может видеть промежуточное состояние выполнения атомарной операции.

Основными функциями являются:

- set — установить переменную.
- inc — увеличить переменную на единицу.
- dec — уменьшить переменную на единицу.
- compare exchange — сначала провести сравнение и, если сравнение было удачным, сделать set.

2.4 MD/ZFS RAID — процесс ребилда

2.4.1 MDRAID

MDRAID является модулем ядра Linux. Он помогает создавать RAID-массивы различных уровней (таких как RAID0, RAID1, RAID5 и их комбинации), настраивать их и управлять ими.

Восстановление поврежденных данных осуществляет в mdraid процесс ребилда. Диск можно заменять без остановки работы массива. При этом mdraid автоматически начинает восстанавливать данные на новом диске из оставшихся в массиве, если уровень конкретного массива это позволяет. Процесс обычно занимает продолжительное время в зависимости от объема данных и производительности дисков, но пользователи могут продолжать работу с массивом в процессе его восстановления. Стоит заметить, что процесс ребилда повышает нагрузку на систему, потому что во время ребилда система выполняет интенсивные вычисли-

тельные операции (например, подсчет паритета) и обрабатывает большое количество операций чтения и записи.

2.4.2 RAID-Z

В файловой системе ZFS[13] есть своя реализация RAID-массивов — RAID-Z[12]. RAID-Z является вариацией RAID5, но имеет свои особенности. Во-первых, RAID-Z использует семантику Copy-on-Write (CoW)[8]. Она никогда не изменяет имеющиеся данные, а полученную при запросе на запись информацию сохраняет на новый блок. Это помогает решить проблему несогласованности данных внутри страйпа после сбоя. Такая проблема называется Write hole[6]. Во-вторых, была увеличена производительность. Так как отсутствуют частичные записи страйпа, потому что каждый блок записи в файловую систему является одним страйпом RAID-Z, независимо от размера блока (страйп имеет динамический размер[5]), то запись новых данных может происходить без необходимости выполнять чтение старых данных и пересчет паритета, то есть без read-modify-write[2]. Из-за добавления этих особенностей структура и устройство RAID-Z были усложнены.

Процесс восстановления RAID в ZFS называется resilver. Resilver — операция, продолжительность которой зависит от объема данных, подлежащих восстановлению, а также от местоположения данных и parity. Если один или несколько дисков выходят из строя во время восстановления, данные теряются, а RAID-массив становится непригодным для использования. Сам процесс происходит следующим образом[3]: ZFS обращается к дереву указателей блоков, чтобы увидеть, как данные размещаются на устройствах. Это замедляет работу из-за фрагментации данных, которая появляется при удалении файлов, однако несет в себе преимущество — система перестраивает только ту часть RAID-Z, которая используется. Затем восстанавливает потерянные данные используя паритет. RAID-Z для высчитывания паритета применяет умножения полей Галуа[17]. После чего записывает восстановленные данные и паритет на устройство.

3 Реализация

3.1 Атомарные переменные

В SPDK уже были внедрены атомарные переменные, однако их использолвание осуществляется через прослойку `env_ocf`, обеспечивающей взаимодействие с внешней библиотекой Open Cache Acceleration Software(OCF)[10]. В случае изменения API указанной библиотеки, потенциально возникает необходимость изменения реализации атомарных переменных, что может повлечь за собой нежелательную зависимость от внешней библиотеки.

Поэтому решено было создать собственную библиотеку атомарных переменных — `atomic_raid`. За основу были взяты встроенные функции предоставляемые компилятором[7]. Эти функции являются расширением кода C и позволяют использовать синтаксис вызовов функций и переменных для доступа к набору команд процессора компилирующей машины. Для реализации атомарных операций были использованы функции атомарного доступа к памяти. Такие функции гарантируют атомарность манипуляций с данными независимо от количества потоков, работающих в системе. Например, `__sync_fetch_and_add` атомарно добавляет указанное значение к переменной, на которую ссылается указатель.

Используя эти функции, были реализованы атомарные операции, основными из которых являются `read`, `set`, `inc`, `dec`, `compare_exchange`. А также функции `set_bit` и `remove_bit`, которые, используя макросы и функцию `compare_exchange`, могут атомарно менять бит с проверкой для достижения синхронизации данных. Эта функция сравнивает содержимое ячейки памяти со значением над которыми происходили манипуляции и, только если они совпадают, изменяет содержимое этой ячейки памяти на новое получившиеся значение, а если значение было обновлено другим потоком, то функция попытается записать значение снова и так пока не сохранит новое значение на основе актуальной информации.

3.2 Структура

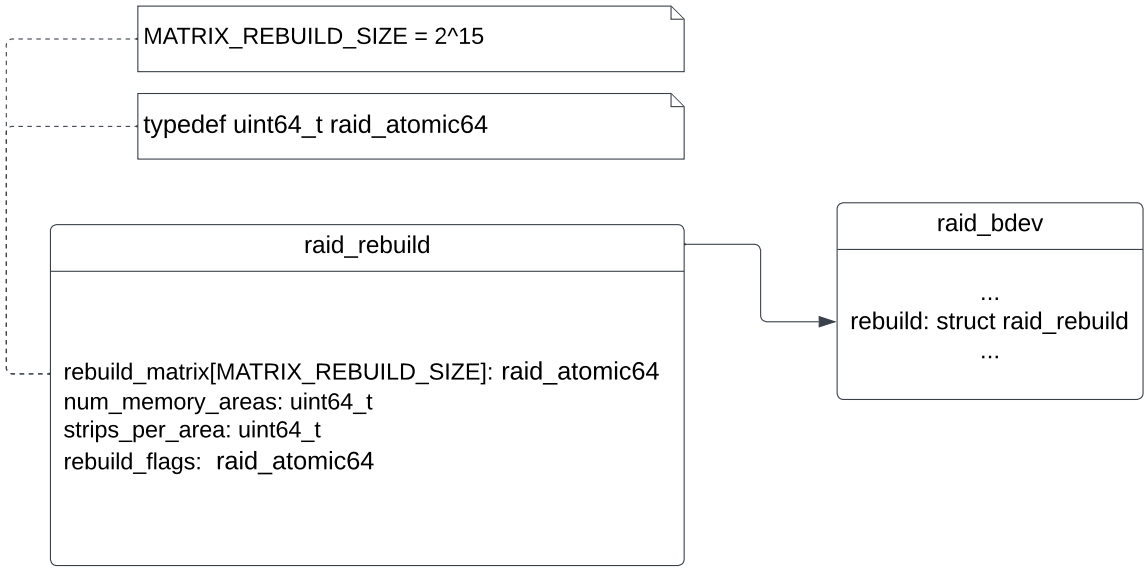

Рис. 4: Структура ребилда

На основе полученной информации о процессе ребилда была разработана структура, описывающая этот процесс. Впоследствии она была внедрена в уже существующую структуру `raid_bdev`, которая содержит информацию о RAID-массиве, поэтому было принято решение вложить сюда структуру, отвечающую за состояние данных внутри всего RAID-массива.

Основным полем в структуре ребилда является матрица. Каждая ячейка матрицы хранит информацию о целостности данных в одной области памяти внутри блочного устройства, входящего в состав RAID массива. Важно отметить, что области памяти содержат внутри себя несколько стрипов, количество которых зависит от объема блочного устройства. Связано это с тем, что стрипов может быть много и хранить информацию об актуальности данных внутри каждого стрипа неэффективно. Из-за того что в области содержится несколько блоков, к одной области может поступать несколько запросов одновременно. Это влечет за собой необходимость параллельного обновления информации о целостности данных в конкретной области памяти. Приемлемым для

производительности решением этой проблемы является использование атомарных переменных. Поэтому матрица состоит из реализованных в библиотеке `atomic_raid` атомарных 64-битных переменных.

Доступ к ячейке матрицы осуществляется посредством атомарных функций на основе битовых операций, что повышает производительность. Битовые операции, такие как `set_bit`, `remove_bit` и `tes_bit`, были внедрены в качестве макросов в файл `util.h`, который содержит внутри себя функции общего назначения.

В созданной структуре матрица была реализована с помощью одномерного массива, каждый элемент из которого описывает страйп областей. А j-ый бит этого элемента характеризует область на j-ом устройстве, входящем в состав RAID-массива. Так, если бит равен единице, область считается недоступной для записи и чтения. Такой подход позволяет экономить пространство.

Помимо матрицы в структуре есть такие поля как флаг — `rebuild_flags`, количество стрипов на область — `strips_per_area` и количество областей в RAID-массиве — `num_memory_areas`. Флаг отвечает за состояние матрицы. Например, `REBUILD FLAG NEED REBUILD` говорит о том, что одно из устройств было повреждено или произошел сбой и требуется процесс восстановления данных, а `REBUILD FLAG INIT CONFIGURATION` показывает, что информация в памяти актуальна. Поле количество стрипов на область нужно для работы с матрицей, в нем записано целое число стрипов, помещающихся на одну область. Более подробно структура ребилда представлена на Рис. 4.

3.3 Взаимодействие с запросами RAID1

Инициализация структуры ребилда происходит при создании RAID1. После чего массив может начать принимать и обрабатывать запросы на запись и чтение. Если при запросе на запись произошел сбой. То происходит процесс записи в матрицу ребилда информации о поврежденных областях памяти. Схематично процесс обработки запроса на запись можно видеть на Рис. 5.

Для этого была реализована функция, которая позволяет определить области, задействованные в текущем запросе, а также функция, которая, используя атомарные операции, меняет флаг структуры на `REBUILD FLAG NEED REBUILD` и выставляет единицы в ячейку матрицы ребилда, отвечающую за область, на которые пришел запрос, но по некоторым причинам не смог обработаться и записаться.

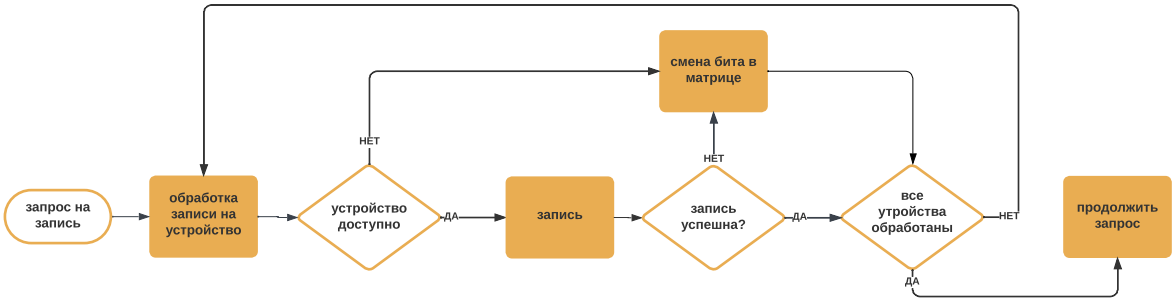

Рис. 5: Процесс обработки запроса на запись

Важно отметить, что запросы на чтение никак не влияют на матрицу ребилда. Однако, если произойдет попытка прочесть области с неактуальными данными (такие области, при записи на которые произошел сбой) система сообщит об ошибке (Рис. 6).

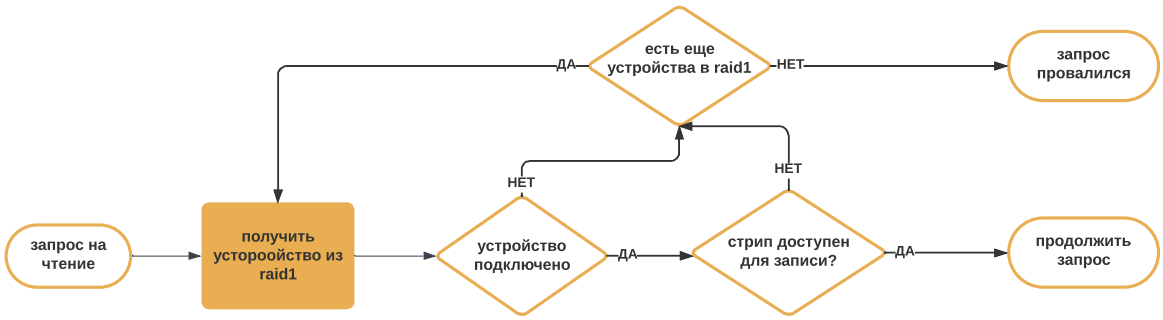

Рис. 6: Процесс обработки запроса на чтение

Подробнее с реализацией можно ознакомиться по ссылке[2].

[2] https://github.com/sumr-proj/spdk/pull/9

4 Тестирование

Были разработаны и организованы интеграционные тесты, предназначенные для проверки взаимодействия созданной функциональности и RAID1. Массив создавался благодаря json файлу `raid1`, а функциональность тестировалась на одном потоке и нескольких одновременно для проверки работы атомарных переменных на операции рандомной записи и чтение благодаря Flexible I/O Tester(FIO).

Утилита FIO[4] использовалась для создания потоков ввода и вывода. Она разработана с целью предоставить гибкую и эффективную среду для проведения различных тестовых сценариев, которые позволяют пользователям оценить производительность и устойчивость I/O под нагрузкой. Одной из ключевых особенностей FIO является его способность настраивать параметры тестирования для воспроизведения реалистичных рабочих нагрузок. Например, можно контролировать количество потоков, размер блоков и операции (чтения или записи).

Сценарий тестирования был следующим: создавался массив RAID1, проводилась проверка матрицы ребилда при инициализации, затем одно из устройств помечалось вышедшим из строя и проходила очередная проверка корректности массива, то есть верно ли в биты, отвечающие за области вышедшего из строя устройства, выставлены единицы.

Заключение

В рамках данной работы были получены следующие результаты

- Исследованы документация и литература, содержащие обзор существующих методов и приемов, а также изучены инструменты, применяемые в задаче параллельной обработки запросов ввода и вывода.
- Разработана структура ребилда, которая включает в себя информацию об актуальности хранимых данных внутри RAID-массива.
- Реализовано взаимодействие с матрицей ребилда в процессе обработки запросов на операции чтения и записи в RAID1.
- Проведены интеграционные тесты.

В весеннем семестре планируются следующие задачи: внедрить структуру ребилда и взаимодействие с ней в RAID5, а также автоматизировать тесты, проверяющие функциональность.

Список литературы

[1] Amdahl Gene M. Validity of the single processor approach to achieving large scale computing capabilities // Proceedings of the April 18-20, 1967, spring joint computer conference. — 1967. — P. 483–485.

[2] Behrmann Niklas. Support for external data transformation in ZFS : Ph. D. thesis / Niklas Behrmann ; PhD thesis, Master's Thesis. Universität Hamburg. — 2017.

[3] Characterizing and modeling reliability of declustered raid for HPC storage systems / Zhi Qiao, Shuwen Liang, Song Fu et al. // 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks–Industry Track / IEEE. — 2019. — P. 17–20.

[4] Flexible I/O Tester. — 2023. — URL: https://fio.readthedocs.io/en/latest/fio_doc.html.

[5] Henson Val, Ahrens Matt, Bonwick Jeff. Automatic performance tuning in the Zettabyte File System // File and Storage Technologies (FAST), work in progress report. — 2003.

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

[7] IBM Documentation. — 2023. — URL: https://www.ibm.com/docs/.

[8] Kasampalis Sakis. Copy on write based file systems performance analysis and implementation // Technical University of Denmark, Department of Informatics. — 2010.

[9] M. Herlihy N. Shavit. The Art Of Multiprocessor Programming. — 2021.

[10] Open Cache Acceleration Software. — 2023. — URL: https://open-cas.github.io/index.html.

[11] 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.

[12] RAID-Z. — URL: https://openzfs.github.io/openzfs-docs/.

[13] Rodeh Ohad, Teperman Avi. zFS-a scalable distributed file system using object disks // 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings. / IEEE. — 2003. — P. 207–218.

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

[15] SPDK NVMe BDEV Performance Report Release 23.09. — URL: https://ci.spdk.io/download/performance-reports/SPDK_nvme_bdev_perf_report_2309.pdf.

[16] What is SPDK. — URL: https://spdk.io/doc/about.html.

[17] zur Erlangung des Doktorgrades der Naturwissenschaften. Erasure and Corruption Resilience Methods for High Performance Parallel File Systems. — 2018.