Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.М05-мм

Петров Иван Васильевич

Разработка и исследование гибридной системы хранения на основе Intel CAS технологии

Отчёт по преддипломной практике
в форме «Производственное задание»

Научный руководитель:
доцент кафедры системного программирования, к.ф.-м.н., Д.В. Луцив

Санкт-Петербург
2023

Оглавление

1. Введение

В современном мире объемы информации с каждым годом растут а также увеличивается количество пользователей запрашивающих данную информацию что повышает нагрузки и требования к системам хранения данных (СХД). Организации и компании разрабатывающие различные продукты СХД стремятся получить наиболее производительную систему с учетом растущего объема высокопроизводительных нагрузок.

Одим из самых известных вариантов организации СХД является семейство технологий RAID- избыточный массив независимых дисков. Производительность RAID обеспечивается за счет чередования (striping) и реже за счет зеркалирования, а отказоустойчивость за счет информационной избыточности которая подразумевает использование дополнительных синдромных дисков предназначенных для восстановления данных при частичном повреждении или утрате.

Большая часть хранимых данных имеет небольшое количество повторных обращений, такие данные принято называть холодными(cold). Они составляют значительную часть как в крупных серверных СХД так и на накопителях обычных настольных компьютеров. Если же к данным осуществляются повторные обращения они будут называться горячими(hot).

Кэширование- это технология в которой используется буфер для хранения часто запрашиваемых данных. Система должна определять горячие данные и перемещать их в буфер. В качестве буфера часто используются устройства хранения с более высокой производительностью, например твердотельные накопители SSD или NVME или кэш-память в RAID контроллере или RAM-диски (технология, позволяющая хранить данные в быстродействующей оперативной памяти). При запросах на чтение и запись часто запрашиваемых данных операции будут выполняться с большей скоростью и с меньшей задержкой.

Системы, которые используют кэширование вместе с HDD-дисками, принято называть гибридными. Они популярны на рынке СХД, так как

значительно доступнее по цене, чем массивы на основе флэш-памяти, и эффективны для работаты с достаточно широким спектром задач и нагрузок. Это делает гибридные СХД подходящими для приложений требующих большой объем хранилища.

Одним из известных продуктов предоставлящих инструмент кэширования является Open Cache Acceleration Software (Open CAS). Open CAS- это проект с открытым исходным кодом основная задача которого предоставить инструмент кэширования за счет устройств с более высокой производительностью. В основе Open CAS лежит Open CAS Framework (OCF). OCF- это высокопроизводительная метабиблиотека кэширования блочного хранилища, написанная на C. Она полностью независима от платформы и системы и тесно интегрируется с остальным программным стеком, обеспечивая высокопроизводительную утилиту кэширования с малой задержкой. В первую очередь OCF он был разработан для кэширования данных с жестких дисков на твердотельных накопителях, но его также можно использовать для кэширования данных с твердотельных накопителей QLC на твердотельные накопители TLC, накопители Optane, оперативную память или любую комбинацию вышеперечисленного, включая все виды многоуровневых конфигураций. На текущий момент OCF— это проект, поддерживаемый сообществом. Первоначально он был разработан Intel. Open CAS постоянно обновляется, появляются новые версиии, новые функции, при этом активно работает отслеживание проблем сообществом призванное быстро исправлять обнаруженные ошибки реализации или объяснять неожиданное поведение кэша.

2. Постановка задачи

Целью работы является повышение производительности DCR Raid с помощью инструмента кэширования Open CAS.

Для достижения этой цели были сформулированы следующие задачи:

1. Сделать обзор применения Open CAS для повышения производительности на различных системах;
2. Проанализировать влияние разных настроек и параметров Open CAS на производительность;
3. Произвести тесты производительности Open CAS и DCR Raid;
4. Выработать рекомендации о использовании CAS для различных паттернов нагрузок по результатам тестирования;
5. Создать анализатор нагрузки для установки необходимых параметров;

3. Обзор

Open CAS довольно распространенное решение кэширования и используется в множестве систем и продуктов, например в Supermicro SuperServer Solutions [12], в технологии Swift [7], в Storage Performance Development Kit (SPDK) [9]. Несколько способов использования Open CAS перечислены в обзорной части работы.

Исследование различных конфигураций хранилищ и использования Open CAS для повышения общей производительности системы с использованием решения Intel SSD data center представлены в 2013 г. [1] Ying-ping Zhang. В тестах статьи используются многопользовательские сценарии сравнительного анализа(MAWD бенчмарки) для имитации использования реальных объемов данных и структур типичного пользователя SAS. Задания запускались с определенными временными задержками друг относительно друга с помощью скрипта для имитации запланированных заданий и запуска интерактивных пользователей в разное время. Типы входных данных теста: текст, набор данных SAS и транспортные файлы SAS. Хотя и используется смешанная нагрузка: чтение, запись, случайная и последовательная, авторы утверждают что в момент написания статьи не стоит использовать Open CAS в сценариях только записи или смешанных чтения и записи основываясь на результатах тестирования и потому как еще не добавлен режим обратной записи(Write-Back) в Open CAS.

В тестах работы [1] кэширование используется по прямому назначению для хранения горячих данных(hot), то есть данных к которым периодически запрашивается доступ.

Исследование повышения производительности кластера Ceph на основе HDD описаны в 2020г. Тинцзе Чен , Вивиан Чжу [4]. Ceph - это широко используемое решение для распределенного хранения. Авторы статьи тестируют Ceph на основе жестких дисков. Используется Open CAS и твердотельный накопитель для устройства кэша как показано на рисунке 1.

CAS в тестах производительности настроен так что последователь-

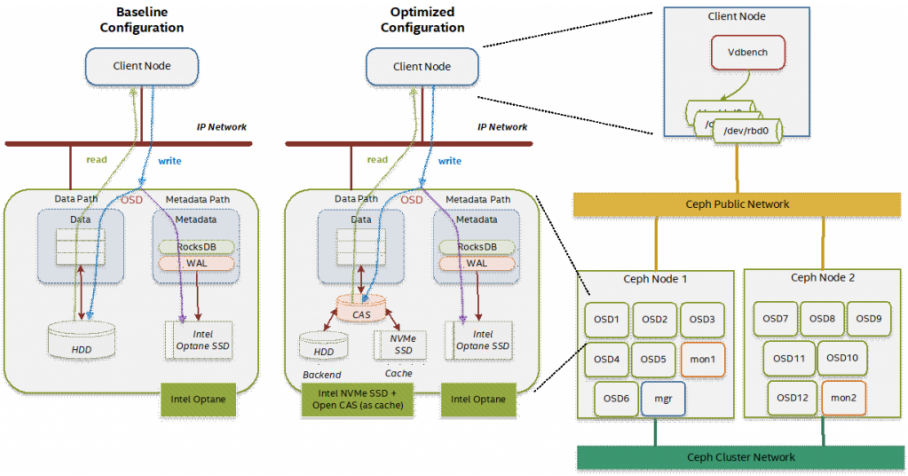


Рис. 1: Схема работы CAS из документации

ная нагрузка идет мимо кэша. Также с использованием классификации ввода вывода случайные запись и чтение направляются в обход кэша если размер запроса более 128K. В такой конфигурации кэшируются в основном случайные чтение и запись. В конце тестов авторы упоминают что в процессе тестирования производительности попадания в кэш достигли 100 процентов и кэш не был заполнен полностью, максимальные значения зполнения кэша фиксируются как 30 процентов(Dirty). Open CAS увеличил производительность случайной записи и чтения в 86 и 119 раз соответственно. Увеличить производительность последовательных операций чтения и записи авторам статьи не удалось. В то же время сообщается что чем больше жестких дисков в ноде, то есть чем больше параллелизм, тем лучше должны быть результаты последовательных нагрузок с использованием Open CAS. Также отмечено неудовлетворительные результаты работы cleaning policy ACP, так как она слишом агрессивно очищает кэш и горячие данные не могут накопиться из-за чего снижается производительность. Очевидно выводы сделаны при условии цели внедрения Open CAS хранить в кэше горячие(hot) данные.

Технология Open CAS в статье о исследовании повышения производительности кластера Ceph на основе HDD [4] используется по основному назначению то есть для кэширования горячих данных(hot) случайных запросов с размером запроса менее 128K. Результаты работы

Open CAS с последовательными операциями признаны неудовлетворительными.

В работе Accelerating Swift with Intel Cache Acceleration Software [7] в которой содержится основная информация для целевой аудитории о проекте повышения производительности Swift с помощью Open CAS описывается примение технологии Open CAS для кэширования только метаданных файловой системы. Так же как и в статьях рассмотренных выше используется SSD для устройства кэширования, то есть для хранения горячих данных. Так как кэшируются только метаданные файловой системы и из-за особенности конфигурации системы в общем(для каждого узла системы хранения используется один жесткий диск и один твердотельный накопитель для устройства кэша) в результате получается увеличение производительности с использованием Open CAS в три раза. К сожалению нет полной информации о конфигурации CAS: статистики попаданий в кэш, заполненности кэша, настроек политики очистки и отсечения последовательных операций.

Во всех рассмотренных источниках Open CAS используется для кэширования горячих данных, в основном случайных запросов. Соответственно большинство настроек выставлены в значения по умолчанию, так как это основная задача которую призван решать кэш вообще и Open CAS в частности.

По результатам рассмотренного применения CAS, кэш используется для кэширования горячих данных чаще случайных запросов записи и чтения. Мы же хотим применить Open CAS для использования последовательного потока, для высокоинтенсивной HPC нагрузки. С помощью Open CAS хотим увеличить производительность для рабочей нагрузки с малой глубиной.

4. Анализ Intel CAS

4.1. Основные обозначения

Raid- рейд массив, избыточный массив независимых дисков, технология виртуализации данных для объединения нескольких физических дисковых устройств в логический модуль для повышения отказоустойчивости и(или) производительности [11].

DCR- это декластерный рейд компании Xinnor, программный рейд, представленный в виде модуля ядра Linux.

ss- strip size, параметр рейда, определяющий объем данных записываемых на рейд за одну операцию ввода/вывода. Размер страйпа задается в момент конфигурации рейд массива и не может быть изменен позднее без переинициализации всего массива [14].

Merge- функция DCR рейда обеспечивает накопление последовательных данных в один страйп и последующую запись этого страйпа на рейд. У функции merge есть два подпараметра- merge wait и merge max [14].

Read modify write- операции которые одновременно считывают ячейку памяти и записывают в нее новое значение. Негативно влияют на производительность DCR.

FIO- Flexible I/O tester, Linux утилита, предназначена для проведения тестов записи чтения. Порождает ряд потоков или процессов, выполняющих определенный тип операций ввода-вывода, указанных пользователем [2].

numjobs/iodepth - параметры FIO теста, обозначение количества заданий и глубины очереди через косую черту. Numjobs- это указанное количество клонов задания. Каждый клон задания создается как независимый поток или процесс. Iodepth- количество единиц ввода-вывода, которые необходимо поддерживать по отношению к файлу, иными словами глубина очереди, т.е. количество одновременных запросов на чтение или запись [2]. В тестах всегда используется numjobs = 1.

Blocksize- параметр FIO теста, размер блока данных в килобайтах

операции ввода-вывода [2].

Offset- параметр FIO теста, смещение, заданное как фиксированный размер в байтах, зонах или процентах, с которого начнется тест FIO. Данные перед заданным смещением не будут затронуты [2].

Blktrace — это специальная утилита, осуществляющая трассировку операций ввода-вывода и предоставляющую подробную информацию о всех операциях [3]. В результате работы утилиты пользователь получает трассировку представленную в виде таблицы, колонки которой отображают такую информацию: мажорный и минорный номера устройства; ядро, задействованное при выполнении операции; порядковый номер операции; время выполнения операции (в миллисекундах); идентификатор процесса (PID); событие (blktrace отслеживает события жизненного цикла всех операций ввода-вывода, в том числе и свои собственные); RWBS (R —чтение, W — запись, B — барьерная операция, S — синхронная операция); блок, с которого началось выполнение операции+число блоков; имя процесса, выполнившего операцию (указывается в квадратных скобках) [3] [15]. Основные операции имеют следующее обозначение: A — операция ввода-вывода была передана другому устройству; C — операция завершена; F — операция объединена со смежной операцией в очереди; I — запрос на выполнение операции поставлен в очередь; M — операция объединена со смежной операцией в очереди; Q — операция поставлена в очередь; T — отключено по причине таймаута; X — операция разбита на несколько операций [15] [3].

Iops- количество операций ввода-вывода в секунду.

CAS- под CAS(cash acceleration software) в тестах будем понимать запущенный, инициализированный кэш и core устройство.

Cache device- устройство кэша, кэш хранилище.

Core device- основное хранилище, блочное устройство которое кеширует Intel Open CAS с помощью устройства кэша. В тестах в качестве core устройства чаще всего будет выступать рейд массив.

Сброс кэша- очистка кеш хранилища.

Hits- попадание в кэш, попытка записать и(или) прочитать блок который присутствует в кэше.

Dirty- грязные данные, данные, которые изменены в кэше, но не изменяются в основном хранилище(core устройстве). В тестах процент dirty будет указывать на заполненонсть кеша данными которые не успел сбросить CAS на core device.

WB, WO, PT- краткое обозначение режимов cash mode: Write-Back, Write-Only, Pass-Through.

ACP (very) agressive politic- специальная настройка политики очистки ACP, которая максимально быстро очищает кеш. Параметры cleaning policy ACP настроены следующим образом: минимальное значением параметра задержки(wake up time) и максимальным значением параметра flush max buffers кратным 2 в степени, то есть 8192.

ALRU (very) aggressive politic- специальная настройка политики очистки ALRU, которая максимально быстро очищает кеш. Параметры cleaning policy ACP настроены следующим образом: минимально возможные значения параметров задержек и максимальное значением параметра flush max buffers кратным 2 в степени, то есть 8192.

RAM диск- программная технология, позволяющая хранить данные в быстродействующей оперативной памяти как на блочном устройстве [10]. Как правило, является составной частью операционной системы [10].

Null device- специальный файл в системах класса UNIX, представляющий собой так называемое «пустое устройство». Запись в него происходит успешно, независимо от объёма «записанной» информации [5].

4.2. Основные настройки Open CAS

Кэш(1 на рисунке 2) — это компонент, который обеспечивает обмен данными между приложением и внутренним хранилищем (2 на рисунке

2), выборочно сохраняя наиболее часто используемые данные в относительно меньшем и более быстром кеше-хранилище (3 на рисунке 2). Каждый раз, когда приложение обращается к кэшированным данным, в хранилище кэша выполняется операция ввода-вывода, что сокращает время доступа к данным и повышает производительность всей системы.

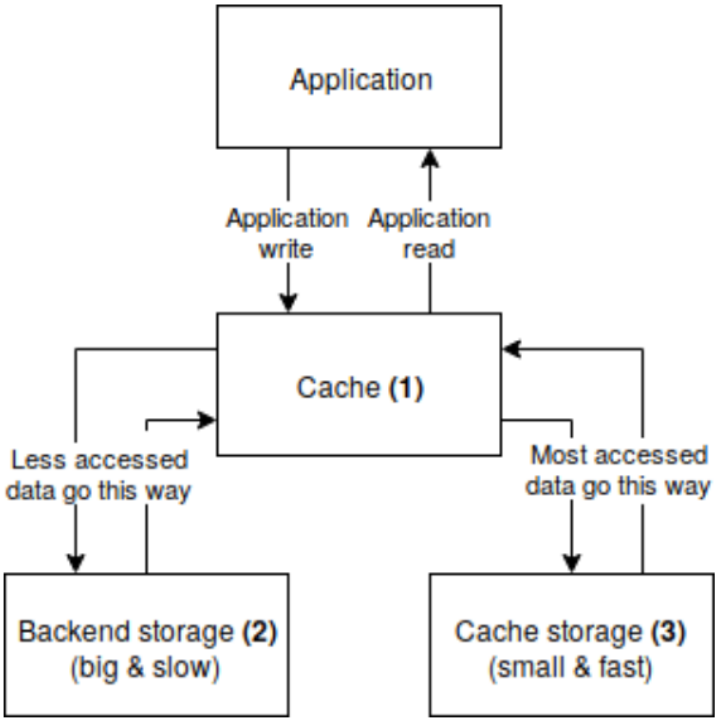


Рис. 2: Схема работы CAS из документации

Данные приложения обрабатываются кэшем с гранулярностью строк кеша.

Параметр размера строки кэша(cash line) определяет размер блока данных, с которым работает кэш. Это минимальная часть данных в бэкэнд-хранилище(core устройство), которую можно сопоставить с кэшем. Каждая отображаемая строка кэша ассоциирована с такой же строкой на core устройстве. И хранилище кеша, и внутреннее хранилище(core устройство) разбиты на блоки размером с строку кеша. Связь между строкой кэша и строкой ядра показана на рисунке ниже.

OCF позволяет установить размер строки кэша в одно из следующих значений:

- 4 KiB

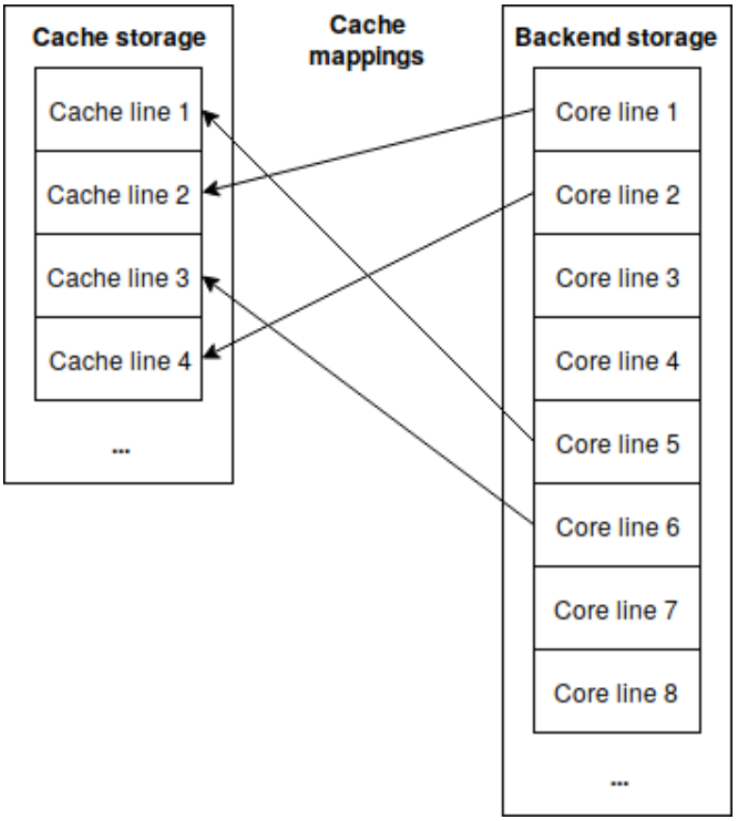


Рис. 3: Связь между строкой кэша и строкой core устройства. Изображение из документации.

- 8 KiB
- 16 KiB
- 32 KiB
- 64 KiB

Параметр режима кэширования(cash mode) определяет, как механизм кэширования обрабатывает входящие запросы ввода-вывода. В зависимости от режима кеша некоторые запросы могут проходить через кеш или нет. Режим кеша также определяет, должны ли данные, хранящиеся в кеше, всегда быть согласованными с данными, хранящимися во внутреннем хранилище (если есть вероятность появления грязных данных).

В настоящее время OCF поддерживает шесть режимов кэширования:

- Сквозная запись (Write-Through)
- Обратная запись (Write-Back)
- Круговая запись (Write-Around)
- Недействительная запись (Write-Invalidate)
- Только для записи (Write-Only)
- Сквозной (Pass-Through)

В режиме Write-Through кеш движок записывает данные в кэш-хранилище и одновременно записывает те же данные «сквозь» в бэкэнд-хранилище. Сквозная запись обеспечивает постоянную синхронизацию данных, записываемых на core device, с данными во внутреннем хранилище. Этот режим ускорит только операции чтения, так как запись необходимо выполнять как на core устройстве, так и на кэш.

В режиме обратной записи (Write-Back) механизм кэширования сначала записывает данные в кэш и подтверждает приложению, что запись

завершена, прежде чем данные будут записаны в core хранилище. Периодически эти данные «записываются обратно» в серверное хранилище по возможности (в зависимости от политики очистки). Хотя режим обратной записи улучшит как интенсивные операции записи, так и чтения, существует риск потери данных, если кэш хранилище выйдет из строя до того, как данные будут записаны во внутреннее хранилище(на core устройство).

В режиме круговой записи(Write-Around) механизм кеша записывает данные в хранилище кеша тогда и только тогда, когда эта строка кеша уже сопоставлена с кэшем (ее можно сопоставить во время запроса на чтение) и одновременно записывает те же данные «сквозь» во внутреннее хранилище. Write-Around похож на Write-Through тем, что обеспечивает 100 процентную синхронизацию ядра с серверным хранилищем. Режим кэширования Write-Around ускоряет только интенсивные операции чтения. Write-Around дополнительно оптимизирует кеш, чтобы избежать загрязнения кеша в тех случаях когда записанные данные не часто впоследствии перечитываются.

В режиме Write-Invalidate кэш-движок записывает данные непосредственно во внутреннее хранилище(core device), и если записанная строка кэша уже сопоставлена с кэш хранилишем, то она становится недействительной. В этом режиме в кеш сопоставляются только запросы на чтение (они обрабатываются так же, как и в режиме Write-Through). Режим Write-Invalidate улучшит только интенсивное чтение. Режим Write-Invalidate также уменьшает количество операций вытеснения для рабочих нагрузок, когда записанные данные не часто впоследствии считываются.

В режиме только для записи(Write-Only) механизм кэширования записывает данные точно так же, как и в режиме обратной записи(Write-Back), поэтому данные записываются в кэш-хранилище без их немедленной записи во внутреннее хранилище. Операции чтения не перемещают данные в кэш из core устрйоства. Режим Write Only ускорит только интенсивные операции записи, так как чтение необходимо выполнять только на внутреннем хранилище. Существует риск потери данных, ес-

ли кэш-хранилище выйдет из строя до того, как данные будут записаны во внутреннее хранилище.

В сквозном режиме(Pass-Through) механизм кэширования будет направлять все запросы на чтение и запись в обход кэша. Это позволяет пользователю динамически включать и отключать кэширование (путем переключения между сквозным и другими режимами кэширования) без необходимости вручную изменять пути ввода-вывода в приложении. Если кеш содержал грязные данные перед переключением в режим Pass-Through, попадания в кеш(hits) будут обрабатываться чтением данных из хранилища кеша до тех пор, пока не будут очищены все строки кеша.

В основном в тестировании будем использовать режимы Write Back, Write Only и Pass-Through.

Очистка — необходимая функция любого решения кэширования. Если есть грязные данные(Dirty), их всегда нужно очищать, сбрасывая грязные данные из кеша в основное хранилище(core устройство).

Существует несколько основных операций, на которых строится вся логика кэширования:

- Отображение или сопоставление(Mapping)- это операция связывания строки core устройства со строкой кеша в метаданных кеша. Сопоставление обновляет идентификатор ядра и номер строки ядра в метаданных строки кэша, а также информацию о хэш-карте.

- Вставка(Insertion)- это операция записи данных строки core устройства в строку кэша. Во время вставки данные запроса ввода-вывода записываются в кэш, в котором хранятся данные конкретной строки кэша, соответствующей строке на core устройстве, к которой осуществляется доступ. Действительные и грязные биты строки кэша в метаданных обновляются соответствующим образом.

- Обновление(Update)- это операция по перезаписи данных строки кэша в кэш хранилище. Операция осуществляется когда строка

core устройства, к которой обращаются во время запроса на запись, отображается в кэш.

- Аннулирование или инвалидация(Invalidation)- это операция пометки одного или нескольких секторов строки кэша как недействительных. Это может быть сделано, например, во время обработки запроса на удаление, во время операции очистки или в результате некоторых операций кэша.

- Вытеснение(Eviction)- это операция удаления сопоставлений строк кэша. Выполняется, когда в хранилище кеша недостаточно места для отображения(ассоциирования) данных входящих запросов ввода-вывода (когда заполнение кеша близко к 100 процентов). Строки кэша, подлежащие вытеснению, выбираются алгоритмом политики вытеснения.

- Сброс(Flushing)- это операция синхронизации данных в кэш-хранилище и внутреннем хранилище(core устройстве). Включает в себя чтение секторов, помеченных как грязные из хранилища кеша, запись их во внутреннее хранилище и установку грязных битов в метаданных строки кеша на ноль. Сброс — это операция управления кешем, и ее должен запускать пользователь, и обычно это делается для безопасного отключения core устройства от кеша без риска потери данных.

- Очистка(Cleaning)- это операция, очень похожая на сброс, но она выполняется кэшем автоматически в качестве фоновой задачи. Процесс очистки контролируется политикой очистки.

В Open CAS Framework, в зависимости от выбранного режима кэширования и конкретного варианта использования, данные очищаются в разных ситуациях и разными средствами. Например, грязные данные могут быть очищены в фоновом режиме с помощью задачи очистки, или очистка может быть запущена механизмом ввода-вывода. Кроме того, грязные данные должны быть очищены, чтобы выполнить вытес-

нение, когда кэш заполнен на 100 процентов, а данные на 100 процентов грязные(dirty). Также пользователь может запросить сброс данных вручную, вызвав соответствующую функцию управления.

Open CAS Framework реализует различные стратегии определения того, когда и какие данные подлежат очистке в фоне. Эти стратегии представляют собой различные политики очистки (cleaning policy). Очистка выполняется периодическим вызовом определенного метода OCF. При вызове метода выполняются проверки работоспособности. Если они успешны, вызывается операция Perform cleaning(), специфичная для политики очистки. Определив, какие данные подлежат очистке, политика вызывает функцию очистки. Эта функция выполняет фактическую очистку, перемещая грязные(dirty) данные на основное устройство.

Алгоритм, используемый для принятия решения о том, что и когда очищать, определяется политикой очистки(cleaning policy). Политика очистки может быть изменена во время работы. В Open CAS реализованы три политики очистки: ALRU, ACP, NOP.

Политика очистки ALRU - Approximately Least Recently Used. Это политика очистки по умолчанию. Основана на классическом подходе списка наименее недавно использовавшихся (LRU) в котором вытесняются значения которые дольше всего не запрашивались.

Политика ALRU использует несколько параметров конфигурации для настройки поведения:

- Время пробуждения(Wake Up Time) — временя пробуждения очитски в режиме ожидания (по умолчанию: 20 с). Время между попытками очистки, когда не было данных для очистки или когда очистка была невозможна из-за ограничения времени активности.

- Staleness Time — время, в течение которого строка кэша находится в списке ALRU, прежде чем ее можно будет очистить (по умолчанию: 120 с).

- Flush Max Buffers — максимальное количество грязных строк кэша, которые нужно сбросить за одну операцию очистки, т. е. один

вызов метода очистки (по умолчанию: 100 строк кэша).

- Порог активности(Activity Threshold) — период времени активности операций ввода-вывода кэша. Очистка не будет начата до истечения этого периода времени с момента последней операции ввода-вывода. (по умолчанию: 10000 мс)

Список LRU растет по мере создания грязных данных — когда строка кэша горячая, т. е. была только что записана механизмом обратной записи(Write Back) или механизмом только записи(Write Only), она добавляется в начало списка LRU с тегом текущей временной метки.

Если строка кэша уже существует в списке, она перемещается в начало списка, а ее временная метка обновляется.

По истечении значения параметра Wake Up Time необходимо выполнить фоновую очистку, вызвав функцию ocf cleaner run(), за которую отвечает адаптер OCF. Если ALRU решает, что необходимо выполнить еще одну итерацию очистки, он передает значение интервала 0, ожидая, что адаптер немедленно вызовет ocf cleaner run(). Этот процесс повторяется до тех пор, пока не останется данных для очистки. Однако фактическая очистка будет выполняться только по истечении времени параметра Activity Threshold, то есть порогового времени активности операций ввода-вывода кэша.

Если все условия соблюдены, строится список сброса до размера значения параметра Flush Max Buffers путем удаления строк кэша из конца списка LRU, при условии истечения времени устаревания (параметра Staleness Time).

Затем этот список передается очистителю OCF с помощью вызова ocf cleaner fire(). Затем очиститель OCF выдает запрос на асинхронную очистку, в конечном итоге записывая грязные данные на основное устройство.

Политика агрессивной очистки (ACP) использует несколько иной подход, чем ALRU. Она организует основное устройство в куски(фрагменты) по 100 МБ, поддерживая их список. Фрагменты помещаются в «ведра» с разным процентом грязных данных. В ACP есть 11 сегментов, содер-

жащих фрагменты данных от 0 процентов грязных до 100 процентов грязных, с гранулярностью 10 процентов. ACP сначала очищает самые грязные фрагменты, как описано ниже.

ACP использует следующие параметры конфигурации для настройки поведения очистки:

- Время пробуждения(Wake Up Time) — временя пробуждения очистки. Время между попытками очистки.
- Flush Max Buffers — максимальное количество грязных строк кэша, которые нужно сбросить за одну операцию очистки (по умолчанию: 100 строк кэша).

ACP создает списки фрагментов для каждого core устройства как часть процесса его добавления. Фрагмент представляет собой линейную часть адресного пространства core устройства размером 100 МБ. Создается специфичное для ACP сопоставление строк кэша с фрагментами. Затем эти фрагменты помещаются в сегменты в зависимости от процента грязных строк кэша в фрагменте.

Когда строка кэша записана и созданы грязные данные, счетчик грязных строк в фрагменте, к которому принадлежит строка кэша, увеличивается. Поскольку фрагменты помещаются в сегменты в соответствии с процентным содержанием грязных строк в фрагменте, фрагмент может быть перемещен в другой сегмент после достижения процентного порога.

По истечении времени пробуждения (параметр Wake Up Time) адаптер должен вызвать фоновую очистку, выполнив ocf cleaner run(). В ACP логика очистки выбирает фрагмент для очистки, начиная с сегментов с наибольшим процентом грязных данных. Как только в сегменте появляется непустой список фрагментов, выбирается первый фрагмент из этого списка.

Список строк кэша размером параметра Flush Max Buffers предоставляется очистителю OCF. Вызывается ocf cleaner fire(), и очиститель OCF выдает запрос на асинхронную очистку, в конечном итоге записывая грязные данные на core устройство.

После успешного сброса будет проивзеден обратный вызов ocf cleaner run() с параметром временного интервала Wake Up Time, и снова начнется выбор фрагмента для очистки. Если фрагмент, который был ранее очищен, все еще содержит некоторые грязные данные, он будет выбран снова, и ограничения очистки будут оценены снова. Следующая порция грязных строк кэша в колчиестве значения параметра Flush Max Buffers этого фрагмента будет очищена.

Политика очистки (NOP) используется когда необходимо отключить функцию очистки в фоне. Данные могут быть очищены вручную с помощью управляющей команды.

Политика последовательного отсечения(Sequential cutoff или Seq-cutoff) используется чтобы не кэшировать последовательные операции ввода-вывода. В Open CAS существует три политики параметра Seq-cutoff:

- Always - последовательное отсечение включено всегда, вне зависимости от заполненности кеша; последовательные данные вообще не будут кэшироваться после достижения порога.
- Full - последовательное отсечение включается только при заполнении кеша.
- Never - последовательное отсечение отключено и не сработает; последовательные данные обрабатываются с использованием текущей политики кэширования.

Параметр Seq-cutoff будет всегда настроен с политикой Never чтобы последовательные запись и чтение шли в кэш.

Open CAS Linux предоставляет возможность управления кэшированием данных с помощью классификации запросов(IO classification). Open CAS Linux может анализировать каждый ввод-вывод на лету, чтобы определить, является ли запрошенный блок метаданными файловой системы или обычнмыи данными, и, если это обычные данные то определить размер целевого файла. Используя эту информацию, администратор может определить наилучшие параметры конфигурации

класса ввода-вывода для типичной рабочей нагрузки и указать, какие классы ввода-вывода следует кэшировать, а какие нет, а также установить уровень приоритета для каждого класса ввода-вывода. Описание полей файла конфигурации, операторов и условий представлены на странице документации IO Classification Guide [6].

4.3. Настройка ACP very agressive politic

При включенной политике очистки ACP происходит сброс грязных строк кэша в количестве равном значению параметра Flush Max Buffers за один раз. То есть данный параметр является глубиной очереди сброса которую можно регулировать. Проведем тест чтобы это подтвердить.

Настраиваем CAS следующим образом: cash mode Write Back, cleaning policy ACP(параметры по умолчанию), sequential cutoff Never. Запускаем FIO тест последовательной записи, ждем когда кеш заполнится, останавливаем тест, с помощью утилиты Blktrace записываем трассировку операций ввода-вывода после окончания теста FIO.

На рисунке 4 и 5 приведены результаты работы утилиты Blktrace при сбросе CAS после окончания нагрузки. В очередь записи поставлено 128 блоков размером 128÷2=64K. Цифра согласуется с параметром Flush Max Buffers значение которого по умолчанию 128.

Увеличим параметр Flush Max Buffers политики очистки ACP до максимально возможного значения кратного 2 в степени. Максимально возможное значение параметра ACP Flush Max Buffers 10000. При Flush Max Buffers = 8192 показатели скорости сброса и очистки всегда лучше чем при максимальном значении 10000 так как DCR Raid лучше реагирует на глубину очереди кратную 2 в степени. При этом стоит уменьшить задержку Wake Up Time до минимального значения, то есть единицы, чтобы поток очистки запускался как можно чаще.

Таким образом получаем специально настроенную политику очистки ACP с максимально большой глубиной очереди(Flush Max Buffers=8192) и минимальным временем пробуждения потока очистки(Wake Up Time = 1), которую назовем ACP very agressive politic. Такая политика про-

blktrace

```
259,0   108     510   27.423760507     0  C   W 247275136 + 128 [0]
259,0   108     511   27.432208897     0  C   W 247274880 + 128 [0]
259,0   108     512   27.443666057     0  C   W 247266432 + 128 [0]
259,0   109       6   27.457406076 385069  Q   W 247275520 + 128 [cas_io_1_109]
259,0   109       7   27.457410106 385069  Q   W 247275648 + 128 [cas_io_1_109]
259,0   109       8   27.457411676 385069  Q   W 247275776 + 128 [cas_io_1_109]
259,0   109       9   27.457413196 385069  Q   W 247275904 + 128 [cas_io_1_109]
259,0   109      10   27.457415016 385069  Q   W 247276032 + 128 [cas_io_1_109]
259,0   109      11   27.457416656 385069  Q   W 247276160 + 128 [cas_io_1_109]
259,0   109      12   27.457417986 385069  Q   W 247276288 + 128 [cas_io_1_109]
259,0   109      13   27.457419196 385069  Q   W 247276416 + 128 [cas_io_1_109]
259,0   109      14   27.457420586 385069  Q   W 247276544 + 128 [cas_io_1_109]
259,0   109      15   27.457422046 385069  Q   W 247276672 + 128 [cas_io_1_109]
259,0   109      16   27.457423386 385069  Q   W 247276800 + 128 [cas_io_1_109]
259,0   109      17   27.457424836 385069  Q   W 247276928 + 128 [cas_io_1_109]
259,0   109      18   27.457426416 385069  Q   W 247277056 + 128 [cas_io_1_109]
259,0   109      19   27.457428046 385069  Q   W 247277184 + 128 [cas_io_1_109]
259,0   109      20   27.457429686 385069  Q   W 247277312 + 128 [cas_io_1_109]
```

Рис. 4: Часть 1 вывода blktrace при значении параметра Flush Max Buffers 128

```
259,0   109     129   27.457907516 385069  Q   W 247291264 + 128 [cas_io_1_109]
259,0   109     130   27.457913516 385069  Q   W 247291392 + 128 [cas_io_1_109]
259,0   109     131   27.457919596 385069  Q   W 247291520 + 128 [cas_io_1_109]
259,0   109     132   27.457925656 385069  Q   W 247291648 + 128 [cas_io_1_109]
259,0   109     133   27.457932246 385069  Q   W 247291776 + 128 [cas_io_1_109]
259,0   109     134   27.459021376     0  C   W 247291776 + 128 [0]
259,0   109     135   27.459321826     0  C   W 247289728 + 128 [0]
```

Рис. 5: Часть 2 вывода blktrace при значении параметра Flush Max Buffers 128

изводит очистку кэша быстрее всех других.

4.4. Настройки CAS для повышения производительности работы с файловой системой Lustre

В качестве не основной задачи подтверждающей корректное применение описанных выше настроек решено продемонстрировать влияние Open CAS на паттерн нагрузки файловой системы Lustre.

Lustre — это распределённая файловая система массового параллелизма, используемая обычно для крупномасштабных кластерных вычислений [8]. Реализованный под лицензией GNU GPL, проект предоставляет высокопроизводительную файловую систему для кластеров с десятками тысяч узлов сети и петабайтными хранилищами информации [8].

На одном из выступлении Storage Field Day в Santa Clara, CA в марте 2016 г. Ron Thornburg говорит о достоинстве Open CAS который не нуждается в какой-либо оптимизации для повышения производительности систем построенных на технологиях Ceph или Lustre делая акцент на возможности классификации запросов и работе Open CAS на блочном уровне [13]. Хотя это справедливо для кэширования горячих данных попробуем оптимизировать Open CAS под работу с файловой системой Lustre не ориентируясь на горячие данные и попадания в кэш.

Lustre специально настроена на работу с файлами с размером блоков 2M и метаданных файловой системы с размером блоков 4K за счет того что инициализированы все таблицы до начала записи. С помощью утилиты blktrace удалось выяснить что глубина очереди запросов записи с размером блоков 2M равна 8, а глубина очереди запросов записи метаданных файловой системы с размером блоков 4K равна 10. С помощью FIO произведены тесты производительности в результате которых делаем вывод что не стоит кэшировать блоки данных размером 2M и глубиной очереди 8 так как скорость с исопльзованием CAS при такой нагрузке меньше чем скорость без CAS.

Для того чтобы не пускать блоки размером 2M в кэш создан кон-

фигурационный файл IO classification поля которого представленны на рисунке 6. Сначала классификатор CAS самостоятельно обнаруживает метаданные файловой системы и направляет их в кэш. Если метаданные не обнаружены, используется простое отсечение: блоки с размером менее 128K идут в кэш, блоки с размерами более 128K направляются мимо кэша. Данные которые не удалось классифицировать направляются мимо кэша.

IO class id	IO class name	Eviction priority	Allocation
0	unclassified	22	0.00
1	metadata&done	0	1.00
2	request_size:le:128000	1	1.00

Рис. 6: Поля и значения файла конфигурации IO classification для работы с файловой системой Lustre

В процессе тестирования политика ACP оказалась слишком агрессивной. Решено использовать политику ALRU. Настройки ALRU подобраны так чтобы блоки 4K накапливались, а потом изредка сбрасывались на core device. Такую политику будем называть special ALRU policy.

Настройки special ALRU policy:

- Wake Up Time - 2 с;
- Staleness Time - 5 с;
- Flush Max Buffers - 256;
- Activity Threshold - 500 мс;

В результате для оптимизации DCR Raid и файловой системы Lustre следует использовать следующие настройки CAS:

- Cache mode Write Back;
- Seq-cutoff Never;
- IO classification ON;

- Cleaning policy: ALRU special policy

При использовании файловой системы Lustre и Open CAS с приведенными выше настройками скорость записи 2100MB/s. Без использования CAS скорость 867MB/s. Удалось повысить производительность при использовании CAS в 2.4 раза.

5. Тесты производительности DCR Raid и Open CAS

5.1. Исходная конфигурация

В этом разделе будут описаны версии ПО, исходные конфигурации сервера на котором проводились тесты производительности.

В качестве DCR Raid используется Raid 60 из 40 дисков HDD, 4 группы, по 10 дисков в группе, 8 для данных и 2 синдромных. Raid Blocksize равен 4K. DCR Raid проинициализирован.

Версия Open CAS 22.06.2.0723. Open CAS настраивается в режим Sequential Cutoff Never чтобы все запросы направлялись в кэш.

Файловой системы на CAS или DCR не создается. Тестирование производительности производится нагрузкой непосредственно на блочное устройство.

Аппаратная и программная конфигурация тестовой среды:

- Процессор AMD EPYC 7763 64-Core Processor(Multithreading on), L1d cache 32K,L1i cache 32K,L2 cache 512K, L3 cache 32768K
- RAM 128GB Memory, DDR4 4x 32GB 3200MHz 1.2V
- Storage 40x GHST HDD 3.7TB, 7200 об/мин, объем буфера 128MB
- Версия ядра Linux 4.18.0-348.2.1.el8-lustre.x86-64
- Операционная система Oracle Linux Server release 8.6

5.2. Ограничения Intel CAS

В процессе тестирования произвоительности CAS было замечено что существует ограничения на скорость записи в кэш и скорость сброса. Было принято решение протестировать CAS и исследовать его собственные ограничения перед тестами производительности для случайных, последовательных и смешанных нагрузок.

Сначала производится замер скорости записи на RAM устройстве. Результаты на рисунке 7. В строках таблицы указаны размеры блоков, в столбцах numjobs/iodepth.

	1/1	1/4	1/32
4K	2466MB/s	2445MB/s	2396MB/s
8K	3946MB/s	3934MB/s	3921MB/s
16K	5889MB/s	5871MB/s	5866MB/s
32K	7706MB/s	7693MB/s	7675MB/s
64K	9427MB/s	9414MB/s	9402MB/s
128K	10.1GB/s	10GB/s	10GB/s
256K	10.5GB/s	10.4GB/s	10.4GB/s
512K	10.8GB/s	10.8GB/s	10.7GB/s
1024K	11.2GB/s	11.2GB/s	11.1GB/s

Рис. 7: Тесты fio скорости записи на ram диск

Далее собираем CAS: в качестве кэш устройства используется RAM диск, в качестве core устройства используется Null device. Null device-это файл устройства который подтверждает успешную запись, принимает данные но не записывает их действительно. Таким образом собираем максимально возможно быструю конфигурацию устройства кэша и core устройства.

Замерим скорости заполнения кэша. Описание хода эксперимента ниже.

Эксперимент 1:

1. Установка cleaning policy NOP.
2. Запуск теста Fio, замер скорости.
3. Конец, остановка теста перед полным заполнением. кэша(100 процентов Dirty).

4. Очистка кэша с помощью управляющей команды очистки или смены политики очистки на ACP.

	1/1	1/4	1/8	1/16	1/32
4K	583MB/s	700MB/s	708MB/s	724MB/s	724MB/s
8K	1082MB/s	1224MB/s	1249MB/s	1259MB/s	1274MB/s
16K	1875MB/s	2027MB/s	2081MB/s	2117MB/s	2152MB/s
32K	2830MB/s	2960MB/s	2941MB/s	3096MB/s	3270MB/s
64K	4239	более 4.5GB/s	более 4.5GB/s	более 4.5GB/s	более 4.5GB/s

Рис. 8: Тесты Fio скоростей заполнения кеша

Протестируем скорости сброса. Описание эксперимента ниже.
Эксперимент 2:

1. Отключаем сброс с помощью установки cleaning policy NOP.
2. Запуск теста Fio, ожидаем заполнения кэша(100 процентов Dirty).
3. Конец, остановка теста после полного заполнения кэша(100 процентов Dirty).
4. Установка cleaning policy ACP agressive politic.
5. Замер скорости сброса.

Результаты на рисунке 9. Скорость сброса везде примерно одинакова и составляет 2200MB/s. Это программное ограничение CAS.

Замерим скорость сброса CAS на DCR raid для разных strip size тем же способом(эксперимент 2) с одним лишь отличием, в начале политика очистки всегда ACP. На рисунке 10 приведены скорости сброса CAS на DCR рейд для двух ss: 64K и 128K. Merge включен и подобраны лучшие настройки для конкретной нагрузки.

	1/1	1/4	1/8	1/16	1/32
4K	2168MB/s	2170MB/s	2153MB/s	2100MB/s	2200MB/s
8K	2108MB/s	2120MB/s	2132MB/s	2135MB/s	2148MB/s
16K	2124MB/s	2120MB/s	2140MB/s	2100MB/s	2100MB/s
32K	2084MB/s	2100MB/s	2111MB/s	2100MB/s	2148MB/s
64K	2211MB/s	2140MB/s	2100MB/s	2170MB/s	2120MB/s
128K	2200MB/s	2100MB/s	2164MB/s	2200MB/s	2200MB/s
256K	2120MB/s	2120MB/s	2174MB/s	2150MB/s	2200MB/s
512K	2144MB/s	2100MB/s	2100MB/s	2100MB/s	2100MB/s
1M	2200MB/s	2120MB/s	2100MB/s	2120MB/s	2200MB/s

Рис. 9: Скорости сбсроса CAS на DCR ss=64

	4K 1/32	16K 1/32	64K 1/32
DCR ss=64	1,77GB/s	1.65GB/s	1.54GB/s
DCR ss=128	1.71GB/s	1.52GB/s	1.52GB/s

Рис. 10: Скорости сбсроса CAS на DCR с разными ss

Исходя из скоростей сброса CAS на DCR Raid и особенностей работы merge(чем меньше blocksize и чем больше ss тем больше должно быть время ожидания накопления страйпа при последовательной записи) для blocksize = 64K и менее лучше использовать strip size = 64K.

В итоге найдены два ограничения: минимальная скорость заполнения кэша и максимальная скорость сброса. Максимальная скорость сброса с CAS не превышает 2200MB/s. Минимальная скорость заполнения кэша равна 583MB/s и получена для теста FIO 1/1 blocksize = 4K. Так как скорость записи в кэш при любых условияъ быстрее, чем на HDD, то для нас главынм ограничивающим параметром использования Open CAS является скорость сброса. Так как она ограничена 2.2GB/s, то нецелесообразно через кэш пускать паттерны, чья скорость на само блочное устройство (без использования кэша) больше 2.2GB/s.

5.3. Тесты случайных операций записи

Конфигурация CAS:

- Cash mode Write Back
- Будем использовать только два режима cleaning policy: very agressive ACP или very agressive ALRU
- Seq Cutoff Never

Найдена особая конфигурация очистки ALRU very agressive politic при установке которой скорости выше чем при very aggressive ACP politic. В этой конфигурации ALRU устанавливается наиболее агрессивный режим сброса. Ее настройки ниже:

Настройки:

- Cleaning policy ALRU
- –wake-up = 0 second
- –staleness-time = 1 second
- –flush-max-buffers = 8192
- –activity-threshold = 0 milliseconds

Эти настройки увеличивают количество сбрасываемых блоков за один цикл очистки до максимально возможного кратного 2 в степени и уменьшают все задержки до минимально возможных.

Очистка в данной конфигурации работает следующим образом: накапливаются строки кэша(cash line) отмеченные как dirty а после они сбрасываются за один раз сильно нагружая DCR. Скорость bandwidth меняется периодически от максимальной(около 8000 iops) до минимальной(около 5 iops), в среднем скорость выше чем при ACP very agressive politic, стандартное отклонение согласно результатам теста FIO 900 IOPS.

Функция merge выключена так как тестируем случайную запись а не последовательную.

Описание эксперимента 3:

1. Пустой кэш(0 процентов Dirty)
2. Установка политики очистки(ACP или ALRU).
3. Запуск теста FIO с новым значением параметра offset для того чтобы избежать попаданий в кэш(hits).
4. Ожидание заполнения кэша(100 процентов Dirty). Исходя из таблицы на рисунке 8
5. Ожидание стабилизации нагрузки(5 минут)
6. Замер скорости теста FIO(3 минуты).
7. Конец теста, ожидание очистки кэша.

Далее привожу результаты тестов FIO случайной записи blocksize 4K разных iodepth на рисунках 11,12,13. Тесты производились согласно описанию эксперимента 3. Полупрозрачным зеленым цветом отмечены ячейки в которых самые высокие скорости с CAS для политики очистки ACP very agressive politic. Зеленым цветом отмечены ячейки с наилучшими показателями скорости(при ALRU very agressive politic).

Согласно таблице на рисунке 11 ss не влияет на случайные операции записи.

Исходя из результатов тестирования представленных на риснунках 12 и 13 для случайных операций записи при blocksize = 4K лучше всего использовать ss=64K. Для увеличения скорости случайной записи маленьким блоком(blocksize = 4K) необходимо использовать Open CAS с политикой очистки ACP very agressive politic(стандартное отклонение согласно результатам теста FIO 100 IOPS), или с политикой очистки ALRU very agressive politic для максимальной скорости(стандартное отклонение согласно результатам теста FIO 900 IOPS). Если важно чтобы стандартное отклонение было менее 10 процентов надо использовать

blocksize 4k rand write	1/32 iops	1/16 iops	1/8 iops	1/4 iops	1/1 iops
DCR ss=128 merge off	930	540	330	203	62
DCR ss=64 merge off	929	560	337	201	63
DCR ss=32 merge off	929	566	337	201	63
DCR ss=16 merge off	930	568	338	202	64

Рис. 11: Тесты FIO случайной записи на DCR Raid с разными ss, blocksize 4K

blocksize 4k rand write	1/32 iops	1/16 iops	1/8 iops	1/4 iops	1/1 iops	cash hits
CAS cash line = 64						
DCR **ss=128**+CAS very agressive politic(-b 8192) **merge off**	1045	828	632	523	368	no
DCR **ss=128**+CAS very agressive politic(-b 8192) **merge on** (default)	1058	817	630	513	368	no

Рис. 12: Тесты FIO случайной записи CAS(DCR Raid ss=128), blocksize 4K

blocksize 4k rand write	1/32 iops	1/16 iops	1/8 iops	1/4 iops	1/1 iops	cash hits
DCR **ss=64**+CAS ACP very agressive politic(-b 8192) merge off	1065	817	635	523	372	no
DCR ss=64+CAS special very agressive alru policy merge off	1632	1604	1495	1589	1675	no

Рис. 13: Тесты FIO случайной записи CAS(DCR Raid ss=64K), blocksize 4K

ACP very agressive politic, если не важно надо использовать ALRU very agressive politic, потому что скорость при такой политике превосходит скорость при политике ACP very agressive politic на разных iodepth в среднем в 2.5 раза.

На графике на рисунке 14 наблюдаем следующую особенность: наилучший прирост производительности получается при глубине очереди 1.

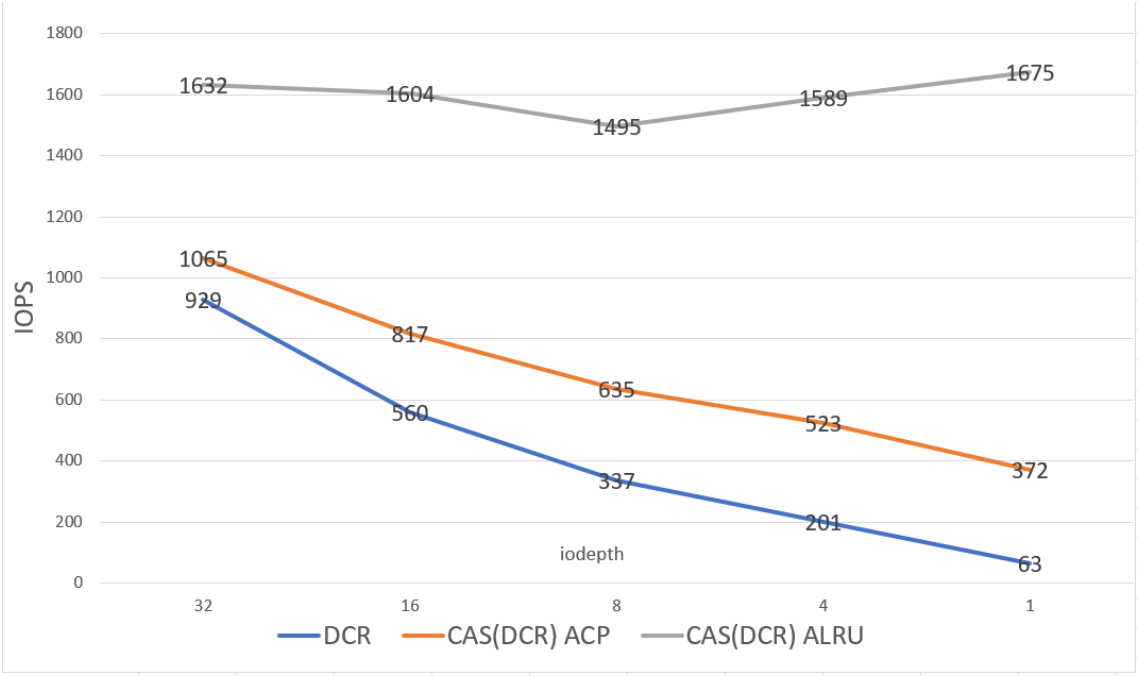


Рис. 14: График зависимости IOPS от глубины очереди нагрузки в тестах FIO случайной записи CAS(DCR рейд ss=64K), blocksize 4K

На графике на рисунке 15 представлено отношение скоростей до включения CAS и после на разных глубинах очереди случайной записи блока 4K. Наилучший прирост производительности получается при глубине очереди 1.

5.4. Тесты смешанных нагрузок

Под смешанной нагрузкой подразумеваются случайные запись и чтение в следующих соотношениях:

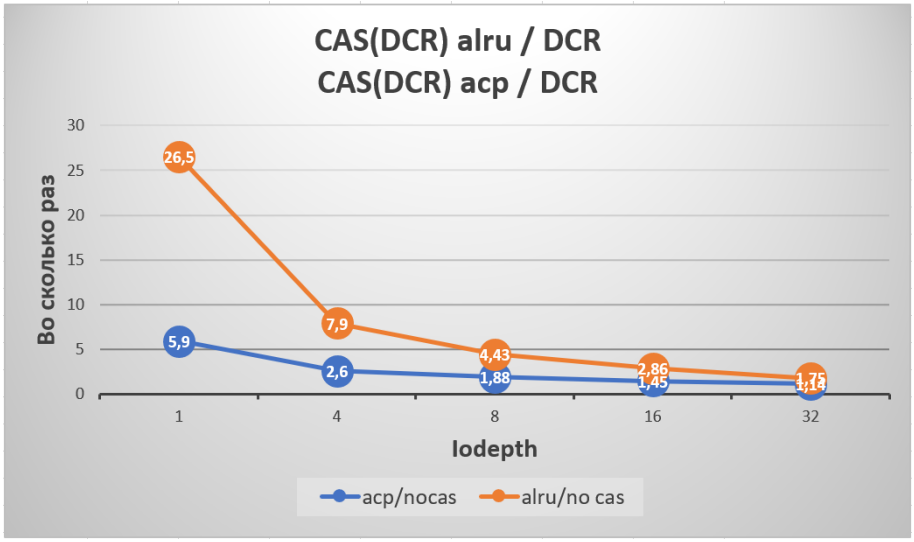


Рис. 15: График отношений показателей с CAS и без для разных глубин очереди в тестах FIO случайной записи CAS(DCR Raid ss=64K), blocksize 4K

- 50 процентов записи к 50 процентам чтения
- 30 процентов записи к 70 процентам чтения
- 70 процентов записи к 30 процентам чтения

Учитывая результаты тестирования случайной записи будем использовать два режима очистки ACP very agressive politic и ALRU very agressive politic и DCR ss=64K. Нет смысла пускать чтения через кэш, поэтому будет использоваться cash mode Write Only. В таком режиме чтения идут мимо кэша, а запись в кэш. Merge выключен так как проводятся тесты случайной записи и чтения. Тесты FIO проводятся по описанию эксперимента 3. Результаты на рисунках 16 и 17. Зеленым цветом отмечены ячейки где скорость превосходит все остальные значения. Скорости указаны в IOPS.

Лучшие показатели скорости достигаются при ALRU very agressive politic и при cash mode Write Only.

Зафиксируем нагрузку mix 50/50 и проведем тесты FIO blocksize=4K на разных iodepth.

Наилучшие значения по приросту производительности получаются при больших глубинах.

1/32	mix 50/50 4K, IOPS	mix 30/70 4K, IOPS	mix 70/30 4K, IOPS
DCR ss=64 merge off	r=591,w=591	r=324,w=751	r=930,w=399
DCR ss=128 merge off	r=591,w=590	r=324,w=749	r=922,w=396
DCR **ss=128**+CAS **WB** ACP very agressive politic(-b 8192) **merge off**	r=685,w=684	r=380,w=888	r=1065,w=455
DCR **ss=128**+CAS **WB** ACP very agressive politic(-b 8192) **merge on (default)**	r=686,w=686	r=374,w=863	r=1069,w=459
DCR **ss=128**+CAS **WO** ACP very agressive politic(-b 8192) **merge off**	r=676, w=677	r=373,w=865	r=1035,w=443

Рис. 16: Тесты производительности FIO 1/32 blocksize 4K смешанного случайного чтения и записи DCR рейда(ss=64, ss=128) и CAS(DCR ss=128)

1/32	mix 50/50 4K, IOPS	mix 30/70 4K, IOPS	mix 70/30 4K, IOPS	
DCR **ss=64** CAS **WB** ACP very agressive politic, **merge off**	r=763,w=764	r=415,w=963	r=1190,w=511	кэш заполняется
DCR **ss=64** CAS **WB** ACP very agressive politic, **merge on**	r=752,w=753	r=409,w=944	r=1168,w=502	кэш заполняется
DCR **ss=64** CAS **WB** ALRU very agressive politic**, merge off**	r=1005,w=1003	r=585,w=1366	r=1395,w=598	кэш не заполняется
DCR **ss=64** CAS **WO** ALRU very agressive politic**, merge off**	r=1159,w=1156	r=613,w=1430	к=1459,w=626	кэш не заполняется
DCR **ss=64** CAS **WO** ALRU very agressive politic,**merge on**	r=1200,w=1198	r=622,w=1450	r=1522,w=652	кэш не заполняется

Рис. 17: Тесты производительности FIO 1/32 blocksize 4K смешанного случайного чтения и записи CAS(DCR ss=64)

mix5050 4k	1/32	1/16	1/8	1/4	1/1
DCR ss=64 no CAS	r=590,w=592	r=359,w=357	r=217,w=215	r=182,w=182	r=63,w=62
DCR ss=64+CAS WO special very agressive ALRU policy merge off	r=1347,w=1347	r=901,w=903	r=571,w=571	r=334,w=331	r=98,w=98

Рис. 18: Тесты FIO blocksize=4K смешанного случайного чтения и записи(50 процентов чтения, 50 процентов записи) CAS(DCR ss=64) с политикой очистки ALRU

График смешанной нагрузки случайного чтения и записи представлен на рисунке 19, на котором отображены результаты тестов DCR и CAS(DCR ss=64K) случайного чтения записи в соотношениях 50 к 50, 30 к 70, 70 к 30. На смешанных паттернах CAS улучшает производительность в среднем в два раза.

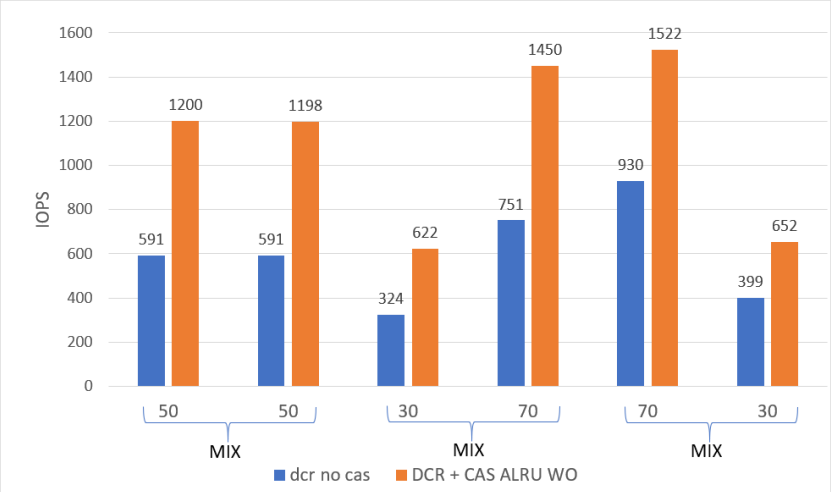


Рис. 19: Диаграмма тестов FIO смешанного случайного чтения и записи blocksize 4K 1/32 DCR и CAS(dcr ss=64K) с политикой очистки ALRU

5.5. Тесты последовательных операций записи малыми блоками

Малым блоком считаем blocksize = 64K и менее. Будут производиться FIO тесты с blocksize 4K,8K,16K,32K,64K с разными iodepth: 1,4,8,16,32.

Настройки merge подобраны лучшие для каждого паттерна нагрузки. CAS в режиме WB, Seq Cutoff Never. Используется DCR ss=64K.

Описание эксперимента 4:

1. Пустой кэш(0 процентов Dirty)
2. Установка политики очистки ACP very agressive politic.
3. Запуск теста FIO с новым значением параметра offset для того чтобы избежать попаданий в кэш(hits).
4. Если при политике очистки ACP very agressive politic кэш заполняется, переходим к следующему пункту 5. Если нет, уменьшаем глубину очереди сброса в два раза, то есть значение параметра flush-max-buffers. Если с уменьшенной глубиной очереди сброса кэш заполняется, то берем предыдущее значение параметра flush-max-buffers и переходим к пункту 6.
5. Ожидание заполнения кэша(100 процентов Dirty).
6. Ожидание стабилизации нагрузки(3 минуты)
7. Замер скорости теста FIO(3 минуты).
8. Конец теста, ожидание очистки кэша.

Замеры скорости на DCR raid ss = 64K приведены на рисунке 20.

	1/1	1/4	1/8	1/16	1/32
4K	5MB/s	12MB/s	18.9MB/s	26.8MB/s	32.9MB/s
8K	10MB/s	23MB/s	26.4MB/s	42.2MB/s	39.8MB/s
16K	19MB/s	47MB/s	50.9MB/s	66.6MB/s	1220MB/s
32K	35MB/s	69MB/s	110.6MB/s	1377.9MB/s	2773MB/s
64K	64MB/s	155MB/s	1495.3MB/s	2999MB/s	5415MB/s

Рис. 20: Последовательная запись малым блоком на DCR Raid ss=64K

Тесты CAS производятся так как описано в эксперименте 4. Синим цветом отмечены те ячейки где при тесте кэш заполняется, т.е. скорость замерялась после заполнения кеша(рисунок 21).

	1/1	1/4	1/8	1/16	1/32
4K -w 1 -b 32	534MB/s	637MB/s	652MB/s	640MB/s	640MB/s
8K	931MB/s	950MB/s	803MB/s	757MB/s	657MB/s
16K	882MB/s	1217MB/s	1100MB/s	1130MB/s	1239MB/s
32K результаты точны	1230MB/s	1368MB/s	1296MB/s	1235MB/s	881MB/s
64K	1101MB/s	1065MB/s	1012MB/s	1018MB/s	1355MB/s

Рис. 21: Скорости последовательной записи малым блоком на CAS(DCR ss=64K)

Сравним полученные результаты скоростей последовательной записи малым блоком на CAS с максимальными скоростями заполнения кеша, которые представлены на рисунке 22).

	1/1	1/4	1/8	1/16	1/32
4K	583MB/s	700MB/s	708MB/s	724MB/s	724MB/s
8K	1082MB/s	1224MB/s	1249MB/s	1259MB/s	1274MB/s
16K	1875MB/s	2027MB/s	2081MB/s	2117MB/s	2152MB/s
32K	2830MB/s	2960MB/s	2941MB/s	3096MB/s	3270MB/s
64K	4239				

Рис. 22: Максимальные скорости заполнения кеша

Отобразим процент скорости последовательной записи маленьким

блоком на CAS(DCR) от максимальной скорости заполнения кеша. Красным цветом указаны ячейки в который процент скорости менее 65. Зеленым цветом отмечены ячейки в которых процент от максимальной скорости заполнения кеша более 65.

	1/1	1/4	1/8	1/16	1/32
4K	91%	91%	92%	88,3%	88,3
8K	86%	77%	64%	60%	51%
16K	47%	60%	52%	53%	57%
32K	43%	46%	44%	39%	26%
64K	25%				

Рис. 23: Проценты скоростей последовательной записи малым блоком на CAS от максимальных скоростей заполнений кеша

Исходя из результатов тестирования приведенных на рисунке 23 делаю следующий вывод: для малых блоков(blocksize <= 64K) когда кэш заполнен(100 процентов Dirty) и есть операции Pass Through процент скорости теста от максимальной скорости заполнения кеша тем меньше чем больше размер блока и глубина очереди.

Мы считаем что идеальными настройками cleaning policy ACP для малых блоков(blocksize <= 64K) являются те, при которых скорость заполнения кэша ненамного больше скорости сброса. Такой ситуации можно достичь например для blocksize 4K 1/1 или 1/4, для которых подобраны специальные настройки cleaning policy ACP. Скорость заполнения кэша для blocksize 4K 1/1 или 1/4 538MB/s и 700MB/s, скорректирована глубина очереди сброса для того чтобы уменьшить скорость сброса. Скорость в тесте составляет 91 процент от скорости заполнения кэша. На статистике CAS видны колебания процентов заполненности кэша(dirty) от 0 до 4-5 процентов.

Необходимо отметить особенность сброса CAS при использовании cleaning policy ACP. Если кеш заполняется медленно(время заполнения кэша размером 29.2GB более 15 минут), то из-за особенностей сброса этой политики: алгоритма выбора фрагментов, ведер и гранулярности,

блоки сильно перемешиваются, производительность падает и скорость меньше границы максимальной скорости сброса в полтора два раза. Такое поведение наблюдается при последовательной записи например для blocksize = 16K 1/1, там где скорость заполнения кеша низкая(время заполнения кэша размером 29.2GB более 15 минут). При этом если взять тот же паттерн нагрузки и произвести тест по описанию эксперимента 2, то есть заполнить кэш с выключенным сбросом чтобы блоки не перемешивались, и измерить скорость, то скорость будет в полтора, два раза выше.

Сравнивая результаты скорости с CAS(рисунок 21) с результатами тестирования DCR теми же последовательными нагрузками(рисунок 20) можно сделать следующий вывод: для blocksize 4K и 8K CAS улучшает производительность на всех глубинах очереди(1,4,8,16,32), для blocksize 16K улучшает на всех кроме iodepth 32, для blocksize 32K улучшает на всех кроме iodepth 32 и 16, для blocksize 64K улучшает только на iodepth 1 и 4.

Построим график(рисунок 24) отношения скоростей с CAS к скоростям без CAS на котором можно наблюдать что наилучшие значения прироста производительности получаются при малых глубинах.

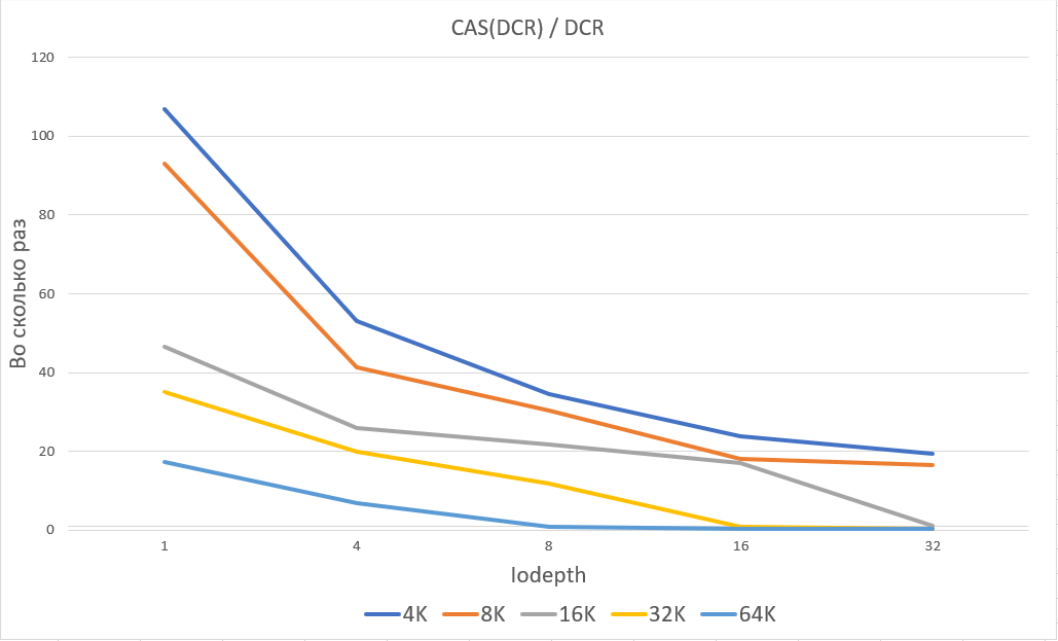


Рис. 24: Графики отношения скоростей CAS(DCR) к скоростям DCR без CAS разными блоками

Таблица, отображающая на какой нагрузке следует использовать CAS приведена на рисунке 25. Если ячейка закрашена зеленым цветом значит следует использовать CAS, если красным значит не следует.

	1/1	1/4	1/8	1/16	1/32
4K	534MB/s	637MB/s	652MB/s	640MB/s	640MB/s
8K	931MB/s	950MB/s	803MB/s	757MB/s	657MB/s
16K	882MB/s	1217MB/s	1100MB/s	1130MB/s	1239MB/s
32K	1230MB/s	1368MB/s	1296MB/s	1235MB/s	881MB/s
64K	1101MB/s	1065MB/s	1012MB/s	1018MB/s	1355MB/s

Рис. 25: Скорости последовательной записи малым блоком с CAS

5.6. Тесты последовательных операций записи большими блоками

В тестах CAS(DCR) большим блоком(blocksize = 128K и более) кэш заполняется всегда. Поэтому политика очистки всегда ACP very agressive politic.

Конфигурация CAS:

- Cash mode Write Back
- Cleaning policy- very agressive ACP
- Seq Cutoff Never

Будем тестировать DCR с ss=64K и ss=128K. Настройки merge подобраны наилучшие для каждого теста. Зеленым цветом на нижеследующих результатах тестов отмечены ячейки где скорость CAS превосходит скорость такого же теста на DCR, то есть скорость выше чем без использования кэша.

При последовательной записи блоком 128K CAS повышает производительность при ss=64 только на глубине очереди 1(рисунок 26), при ss=128 только на глубине очереди 1 и 4(рисунок 27). Так как скорости

при ss=64 без CAS на 1/4 и 1/8 больше чем скорости при ss=128, а скорость с CAS на 1/1 больше при ss=64K, то для последоватльной записи blocksize 128K следует использовать ss=64, а CAS только для глубины очереди 1.

ss=64	128K 1/1	128K 1/4	128K 1/8
DCR merge on	79MB/s	1566MB/s	3193MB/s
CAS(DCR) merge on	1274MB/s	1381MB/s	менее 1400MB/s

Рис. 26: Тесты FIO blocksize=128K, DCR и CAS(DCR), ss=64K

ss=128	128K 1/1	128K 1/4	128K 1/8	128K 1/16	128K 1/32
DCR merge on	90MB/s	234MB/s	1293MB/s	2657MB/s	5310MB/s
CAS(DCR) merge on	1200MB/s	1320MB/s	1197MB/s	1266MB/s	менее 1400MB/s

Рис. 27: Тесты FIO blocksize=128K, DCR и CAS(DCR), ss=128K

При последовательной записи blocksize 256K CAS повышает производительность при ss=64 только на глубине очереди 1(рисунок 28), при ss=128 только на глубине очереди 1 и 4 (рисунок 29). При этом для ss=128K 1/4 скорости примерно равны скоростям без CAS, на рисунке 29 отмечены желтым цветом. Так как скорости без CAS при ss=64K на 1/4 и 1/8 больше чем при ss=128K, а скорость с CAS при ss=64K больше чем при ss=128K, то для последовательной записи blocksize 256K следует использовать ss=64K и CAS для iodepth 1.

ss=64	256K 1/1	256K 1/4	256K 1/8
DCR merge on	122MB/s	3257MB/s	5740MB/s
CAS(DCR) merge 1000 300	1380MB/s	1625MB/s	менее 1600MB/s

Рис. 28: Тесты FIO blocksize=256K, DCR и CAS(DCR), ss=64K

ss=128	256K 1/1	256K 1/4	256K 1/8	256K 1/16
DCR merge on	141MB/s	1327MB/s	2710MB/s	5388MB/s
CAS(DCR) merge on	1172MB/s	1374MB/s	1327MB/s	менее 1400MB/s

Рис. 29: Тесты FIO blocksize=256K, DCR и CAS(DCR), ss=128K

При последовательной записи блоком 512K CAS повышает производительность при ss=64 только на глубине очереди 1(рисунок 30), при ss=128 тоже только на глубине очереди 1 (рисунок 31). Так как скорости без CAS при ss=64K на 1/4 больше чем при ss=128K, а скорость с CAS при ss=64K больше чем при ss=128K, то для последовательной записи blocksize 512K следует использовать ss=64K и CAS для iodepth 1.

ss=64	512K 1/1	512K 1/4
DCR merge on	1646MB/s	5909MB/s
CAS(DCR) merge on	2200MB/s	2660MB/s

Рис. 30: Тесты FIO blocksize=512K, DCR и CAS ss=64K

ss=128	512K 1/1	512K 1/4	512K 1/8
DCR merge on	230MB/s	2720MB/s	5341MB/s
CAS(DCR) merge on	1301MB/s	1719MB/s	менее 1700MB/s

Рис. 31: Тесты FIO blocksize=512K, DCR и CAS ss=128K

При последовательной записи блоком 1M CAS повышает производительность при ss=64 на глубинах очереди 1,4,8(рисунок 32), а при ss=128 только на глубине очереди 1 (рисунок 33). Так как скорости без CAS при ss=128K на 1/4 больше чем при ss=64K, а скорость с CAS при ss=128K больше чем при ss=64K, то для последовательной записи blocksize 1M следует использовать ss=128K и CAS для iodepth 1.

ss=64	1M 1/1	1M 1/4	1M 1/8
DCR merge on	2788MB/s	2183MB/s	2191MB/s
CAS(DCR) merge on	2283MB/s	2921MB/s	2950MB/s

Рис. 32: Тесты FIO blocksize=1M, DCR и CAS ss=64K, две настройки merge

ss=128	1M 1/1	1M 1/4
DCR merge on	1368MB/s	5500MB/s
CAS(DCR) merge on	2323MB/s	3068MB/s

Рис. 33: Тесты FIO blocksize=1M, DCR и CAS ss=128K, две настройки merge

В результате тестирования больших blocksize, то есть 128K и более, подтверждаем выводы раздела 5.1 о том что не следует использовать CAS при тех паттернах где скорости на само блочное устройство(без использования кэша) больше 2.2GB/s.

Исходя из результатов тестирования последовательной записи блоками 128K, 256K, 512K и 1M делаем следующий вывод: следует использовать CAS только для iodepth 1; ss=64K для blocksize 128K, 256K и 512K; ss=128K для blocksize 1M.

Показатели наиболее высоких скоростей с CAS для последовательной записи iodepth 1 для ss=64 blocksize 128K, 256K, 512K и для ss=128 blocksize 1M представлены на рисунке 34.

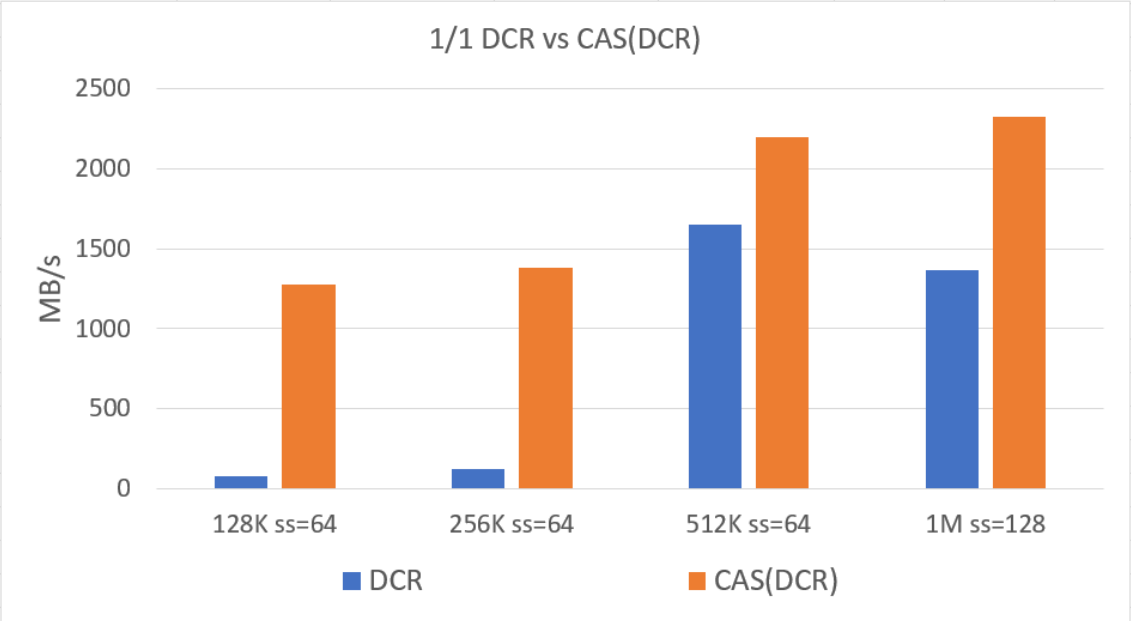


Рис. 34: Лучшие показатели скорости DCR и CAS в тестах FIO 1/1 blocksize 128K, 256K, 512K, 1M. ss=64K и ss=128K

6. Общие рекомендации по использованию Open CAS

Учитывая тесты в параграфе 5 подведем итоги: когда следует использовать CAS и с какими параметрами.

Для операций случайной записи blocksize 4K следует использовать CAS для всех глубин очереди в режиме WB и ALRU agressive politic.

Для смешанных операций случайной записи и чтения маленькими блоками следует использовать CAS в режиме WO и ALRU agressive politic.

Для последовательных операций записи маленьким блоком(blocksize 64K и менее) следует использовать CAS для определенных паттернов нагрузки. Таблица, отображающая на какой нагрузке следует использовать CAS приведена на рисунке 35. Если ячейка закрашена зеленым цветом значит следует использовать CAS, если красным значит не следует.

	1/1	1/4	1/8	1/16	1/32
4K	534MB/s	637MB/s	652MB/s	640MB/s	640MB/s
8K	931MB/s	950MB/s	803MB/s	757MB/s	657MB/s
16K	882MB/s	1217MB/s	1100MB/s	1130MB/s	1239MB/s
32K	1230MB/s	1368MB/s	1296MB/s	1235MB/s	881MB/s
64K	1101MB/s	1065MB/s	1012MB/s	1018MB/s	1355MB/s

Рис. 35: Скорости DCR с CAS после корректировки параметров

Для последовательных операций записи большим блоком(blocksize 128K и более) следут использовать CAS только для iodepth 1. Для blocksize 128K, 256K, 512K следует использовать ss=64, для блока 1M следует использовать ss=128K.

В результате учета вышеизложенных выводов по использованию Open CAS был написан анализатор запросов на языке программирования python 3. Программа снимает трассировку текущей нагрузки, определяет какие запросы следует направить в кэш а какие нет и устанавливает необходимые настройки CAS. Блок схема программы приведена в приложении 1.

Анализатор запросов работает следующим образом. Сначала устанавливается режим кэша WO чтобы не кэшировать операции чтения. Далее снимается трассировка нагрузки, по которой определяется наличие случайной записи. Если есть случайная запись программа включает политику очистки ALRU very agressive politic, определенную как наилучшую для случайной записи чтения в разделе 5.2 и 5.3 текста работы.

При отсутствии случайной записи производится анализ входящего потока. После анализа получается сигнатура в которой содержатся: наличие случайной записи, размер блока, глубина очереди каждого задания. По каждому заданию принимается решение о кэшировании и конфигурации настроек.

Наличие случайных операций записи определяется с помощью трассировки. Определяется несоответствия адреса записи адресу записи преды-

дущего блока данных а также с помощью разметки трассировки где запись блока отмечена как не последовательная.

Экспериментально удалось установить что глубина очереди соответствует максимально возможному количеству операций постановки в очередь последовательных блоков встречающихся за раз. При достаточном времени сбора трассировки(30 секунд) всегда находится такая последовательность соответствующая глубине очереди.

Размер блока определяется по выводу трассировки как одно из значений поля размера блоков данных. Программа агрегирует все встречающиеся размеры блоков.

Далее выполняется принятие решения по сигнатуре каждого задания нагрузки о кэшировании и настройках. Для того чтобы определить наилучшие настройки произведены тесты производиетльности о которых далее пойдет речь.

7. Заключение

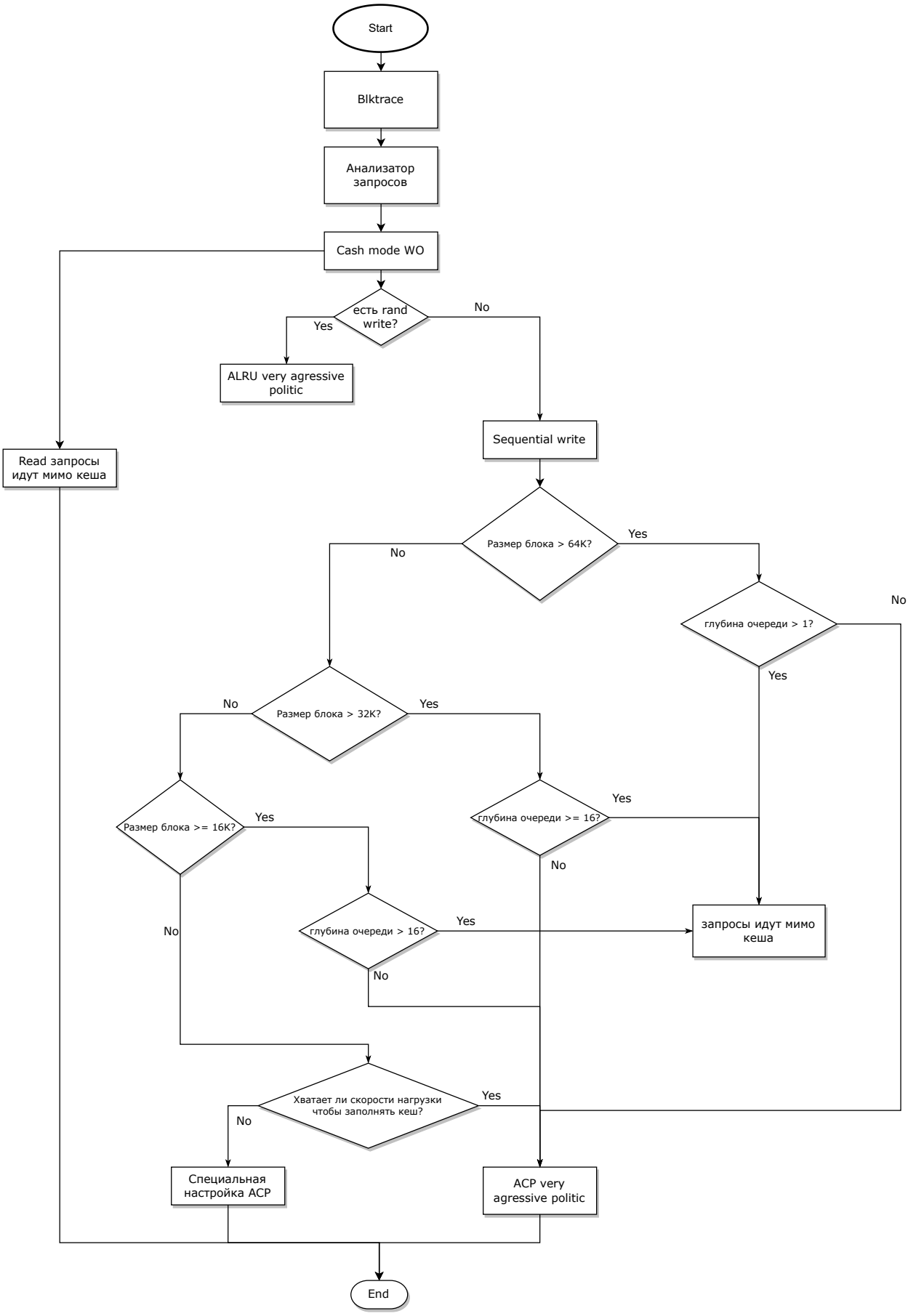
В результате выполнения производственной практики/вкр были получены следующие результаты:

- Произведен обзор применения Open CAS для повышения производителньости на различных системах;
- Проанализировано влияние разных настроек и параметров Open CAS на произвоидетльность;
- Получены результаты тестов производительности Open CAS с DCR Raid в качестве core устройства;
- Выработаны рекомендации о использовании Open CAS для различных паттернов нагрузок по результатам тестирования;
- С учетом выработанных рекомендаций создан анализатор запросов для установки параметров CAS в зависимости от нагрузки.

В будущем предстоит:

- Реализовать другую политику LRU в которой не будет дискретности
- Увеличить линию кэша
- Реализовать политику упреждающего чтения для нагрузок последовательного чтения
- Частотные случайные запросы направить в кэш.

8. Приложение 1



Список литературы

[1] Accomplish Optimal I/O Performance on SAS 9.3 with Intel Cache Acceleration Software and Intel DC S3700 Solid State Drive / Ying ping (Marie) Zhang, Jeff Curry, Frank Roxas, Benjamin Donie. — 2013.

[2] Axboe Jens. Flexible I/O tester. — URL: https://fio.readthedocs.io/en/latest/fio_doc.html (online; accessed: 08.12.2022).

[3] Blktrace man page. — URL: https://man7.org/linux/man-pages/man8/blktrace.8.html (online; accessed: 02.01.2023).

[4] Chen Tingjie, Zhu Vivian. RESEARCH ON PERFORMANCE TUNING OF HDD-BASED CEPH* CLUSTER USING OPEN CAS. — 2020.

[5] Dev Null. — URL: https://ru.wikipedia.org/wiki//dev/null (online; accessed: 05.01.2023).

[6] IO Classification Guide. — URL: https://open-cas.github.io/guide_io_classification.html (online; accessed: 07.01.2023).

[7] Intel. Accelerating Swift with Intel Cache Acceleration Software. — URL: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/accelerating-swift-white-paper.pdf (online; accessed: 15.02.2023).

[8] Lustre (сетевая файловая система). — URL: https://ru.wikipedia.org/wiki/Lustre_(сетевая_файловая_система)#Объекты_данных_и_разделение_данных (online; accessed: 12.01.2023).

[9] Open CAS main page. — URL: https://open-cas.github.io/ocf_intro.html (online; accessed: 02.02.2023).

[10] RAM-диск. — URL: https://ru.wikipedia.org/wiki/RAM-диск (online; accessed: 04.01.2023).

[11] Raid. — URL: https://ru.wikipedia.org/wiki/RAID (online; accessed: 22.11.2022).

[12] SUSE Enterprise Storage v5 and Intel Cache Acceleration Software Implementation. — URL: https://www.supermicro.com/solutions/Supermicro-SES-5-CAS-Implemetation-Guide.pdf (online; accessed: 05.02.2023).

[13] Storage Field Day 9. — URL: https://youtu.be/uMRdzl701t8?t=682 (online; accessed: 14.01.2023).

[14] Xinnor. — URL: https://xinnor.io/docs/xiRAID/E/en/AG/title.html (online; accessed: 25.11.2022).

[15] Анализ производительности блочных устройств с blktrace. — URL: https://habr.com/ru/companies/selectel/articles/199350/ (online; accessed: 03.01.2023).