Санкт-Петербургский государственный университет

Камкова Екатерина Александровна

Выпускная квалификационная работа

Исследование и разработка оптимизации функционирования системы хранения данных

Уровень образования: бакалавриат

Направление *02.03.03 «Математическое обеспечение и администрирование информационных систем»*

Основная образовательная программа *СВ.5006.2017 «Математическое обеспечение и администрирование информационных систем»*

Профиль *Системное программирование*

Научный руководитель:
проф. каф. СП, д. ф.-м. н., профессор А.Н. Терехов

Консультант:
Технический директор ООО "Рэйдикс", к.т.н. С.В. Лазарева

Рецензент:
Разработчик исследовательской лаборатории ООО "Рэйдикс" А.И. Васенина

Санкт-Петербург
2021

Saint Petersburg State University

Ekaterina Kamkova

Bachelor's Thesis

Investigation and development of optimising functioning of the data storage system

Education level: bachelor

Speciality *02.03.03 "Software and Administration of Information Systems"*

Programme *CB.5006.2017 "Software and Administration of Information Systems"*

Profile: *Software Engineering*

Scientific supervisor:
Sc.D, prof. A.N. Terekhov

Consultant:
Head of RAIDIX RD department, C.Sc. S.V. Lazareva

Reviewer:
RD Software Engineer RAIDIX A.I. Vasenina

Saint Petersburg
2021

Оглавление

Введение

В современном мире для решения профессиональных и бытовых задач люди всё чаще полагаются на информационные технологии, при этом порождая огромное количество данных, к которому рассчитывают иметь доступ. Для удовлетворения данной потребности были созданы системы хранения данных (СХД) — комплексные программно-аппаратные решения, организующие хранение больших объёмов информации и обеспечивающие к ним доступ.

Иногда происходят события, влекущие за собой потерю данных. Как решение данной проблемы, а также для ускорения работы с данными используются избыточные массивы независимых дисков (RAID), за счёт различных подходов повышающие надёжность хранения информации и скорость взаимодействия с ней. В результате пользователь получает в распоряжение большой объём памяти фиксированного размера, с которым не очень удобно взаимодействовать.

Для решения вышеописанной проблемы можно использовать менеджер томов — систему управления дисковым пространством, позволяющую абстрагироваться от физических устройств. Менеджеры томов обладают различной функциональностью, такой как:

- организация памяти в соответствии с желанием пользователя — некоторые менеджеры томов позволяют объединить физические устройства в единое хранилище, а затем разбить его на участки желаемого размера;
- тонкое выделение (thin provisioning) — механизм оптимизации эффективности использования доступного пространства; при таком подходе приложениям выделяется ровно столько пространства, сколько им требуется на данный момент, вместо запрошенного приложением размера сверх текущих потребностей; таким образом снижаются требования к пространству устройства;

- снепшот (snapshot) — снимок участка системы в определённый момент времени, с помощью которого можно будет восстановить состояние системы на момент создания снимка;

- дедупликация (deduplication) — метод сокращения объёма памяти, необходимого для хранения данных; при его использовании повторяющиеся участки памяти заменяются ссылкой на первое появление участка данных;

- сжатие (compression) — метод сокращения объёма памяти, необходимого для хранения данных; устраняет избыточности, содержащиеся в исходных данных, включает, но не ограничивается дедупликацией.

Компания RAIDIX занимается разработкой систем хранения данных для задач, требующих высокой производительности, а потому их продукты в первую очередь рассчитаны на рынки, где важен именно этот показатель. На данный момент рассматривается идея создать систему хранения данных на базе RAIDIX для Enterprise рынка, который требует от СХД более расширенной функциональности.

Для достижения данной цели изначально необходимо провести обзор существующих менеджеров томов, исследовать их влияние на производительность СХД, а также создать инструмент, позволяющий подбирать параметры и отображать потенциальное влияние новых функциональностей на СХД в зависимости от типа нагрузки.

1. Постановка задачи

Целью данной работы является исследование и сравнение менеджеров томов, их влияния на производительность, а также создание инструмента, позволяющего по типу нагрузки оценивать влияние новой функциональности и рекомендовать настройку параметров. Для ее достижения были поставлены следующие задачи.

1. Изучить менеджеры томов и выбрать наиболее подходящие.
2. Провести тестирование, в ходе которого:
   (a) изучить и сравнить показатели выбранных менеджеров томов при использовании дополнительных функциональностей;
   (b) измерить влияние новой функциональности на производительность;
   (c) обнаружить зависимость оптимальных параметров от типа нагрузки на СХД.
3. Создать инструмент, позволяющий по типу нагрузки определить оптимальные параметры для конкретной задачи.

2. Обзор используемых технологий

2.1. Объект тестирования

ERA [8] — это программный RAID, оптимизированный для работы с flash-накопителями (NVMe, SAS, SATA). За счёт параллелизации операций ввода/вывода все вычислительные ядра процессора используются равномерно, а возникающие в результате задержки устраняются при помощи разработанной в компании Lockless архитектуры. В Raidix 5.0.1 для ERA используется менеджер томов LVM для разбиения хранилища на тома (участки памяти, представляемые пользователю как единое физическое устройство).

2.2. Менеджеры томов

2.2.1. ZFS

Zettabyte file system (ZFS) [6] — это файловая система, обладающая функциональностью менеджера томов.

ZFS объединяет физические устройства в пул хранения, который описывает физические характеристики хранилища и объединяет память всех входящих в него устройств. В ZFS присутствует возможность организовывать часть пространства в том, который может быть использован как блочное устройство.

При записи данных ZFS всегда использует механизм copy-on-write. Это значит, что если необходимо изменить блок данных, вместо модификации старого блока ZFS создаёт новый блок с уже внесёнными изменениями, а затем меняет ссылку со старого блока на новый. Данный подход позволяет избежать проблему Write Hole (при неожиданном прерывании во время записи данные могут быть испорчены), но значительно увеличивает фрагментацию и необходимое для записи время.

Помимо тонкого выделения, создания снепшотов, дедупликации и сжатия, интересующих нас в рамках данной работы, ZFS обладает множеством других функций, избыточных для поставленной цели. Более того, ZFS в первую очередь — файловая система, а следовательно данный вариант не подходит пользователям, которым нужно только блочное устройство.

2.2.2. LVM

Logical Volume Manager (LVM) [5],[3] — это менеджер управления томами данных для Linux. Поддерживает организацию пространства удобным для пользователя образом, тонкое выделение и снепшоты.

Организация пространства

Чтобы использовать физическое устройство (в данном случае — RAID), необходимо проинициализировать его как физический том. Как результат, во втором 512-байтном секторе будет размещён дескриптор, содержащий идентификатор, порядковый номер, размер устройства и указатель на метаданные, которые хранят информацию о деталях конфигурации групп томов.

Когда физические тома объединяются в группу томов, их память разбивается на физические экстенты — участки памяти фиксированного размера. В дальнейшем физические экстенты будут считаться наименьшей единицей измерения памяти данной группы томов.

Из группы томов можно выделить логические тома. Пользователь будет видеть их как обычные блочные устройства. Пространство логических томов разбито на логические экстенты, которым в соответствие ставятся физические экстенты. В один логический том могут входить участки разных размеров различных физических томов. Состав томов может быть определён автоматически или задан пользователем. Пример организации пространства с помощью LVM изображён на Рис. 1.

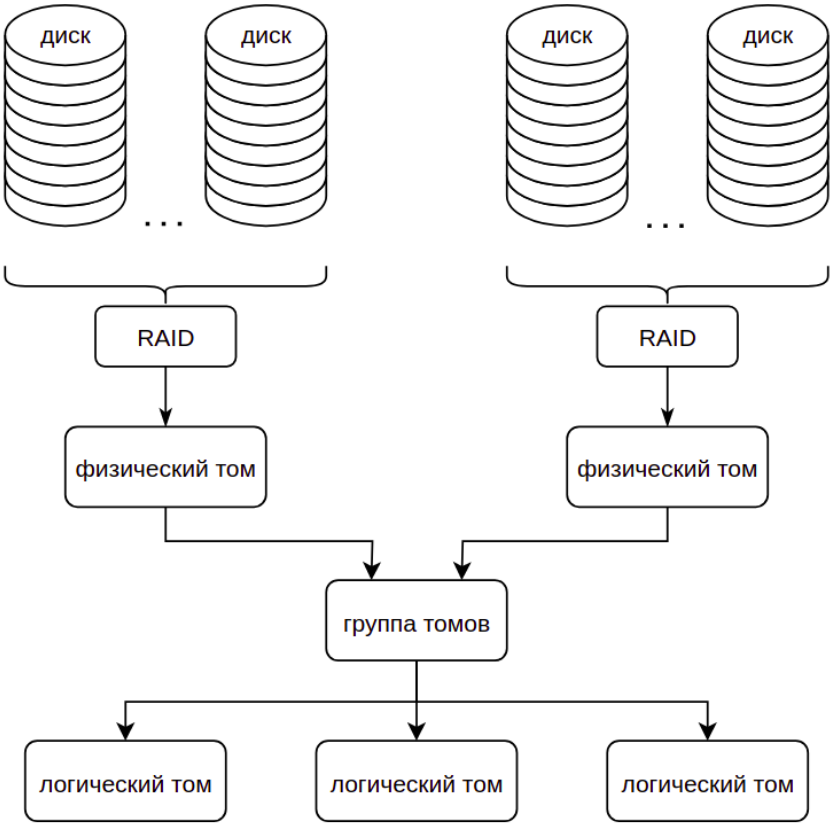

Рис. 1: Структура LVM

Помимо обычных, LVM предоставляет тома, обладающие различными дополнительными свойствами. В рамках данной работы будут рассмотрены тома-снепшоты (далее просто снепшоты), тома с тонким выделением (далее тонкие тома) и тонкие тома с тонкими снепшотами.

Снепшоты

В момент создания снепшот пуст (см. Рис. 2 пункт 1). Если пользователь решит изменить блок данных, блок сперва будет скопирован в снепшот, и только потом изменён в самом томе (см. Рис. 2 пункт 2). При последующих изменениях уже скопированного в снепшот блока, никаких дополнительных действий со снепшотом не произойдёт — блок просто будет перезаписан в исходном томе (см. Рис. 2 пункт 3). Когда пользователь решит восстановить состояние тома на момент создания

снепшота, все блоки из снепшота будут скопированы в соответствующие блоки исходного тома, а снепшот — удалён.

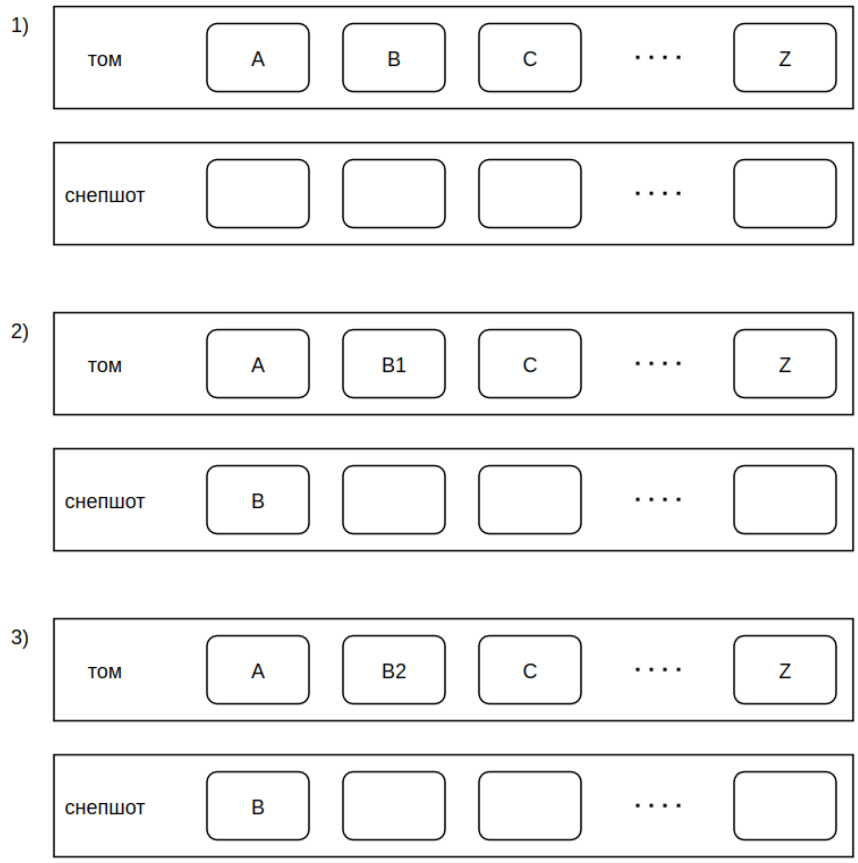

Рис. 2: Снепшот LVM

Поскольку в снепшот копируются только изменяющиеся блоки данных, размер следует задавать, учитывая, насколько локальны изменения и как долго будет храниться снепшот. Создание снепшота размера, превышающего размер исходного тома, бессмысленно. Если в снепшоте закончится свободное место, он будет удалён. Имеется возможность динамически изменять его размер.

В LVM существует возможность чтения и записи в снепшоты. При чтении происходит проверка, находится ли искомый блок в снепшоте. Если да, то он читается из снепшота, иначе — из исходного тома. В случае записи происходит аналогичный процесс: если блок находится в снепшоте, он модифицируется, иначе блок копируется из исходного тома

в снепшот, и изменяется в снепшоте.

Необходимость копирования блоков данных во время записи в исходный том может значительно влиять на производительность, особенно при большом количестве операций на запись в разные участки тома.

Тонкие тома

Изначально в группе томов создаются два тома. Один из них содержит все блоки памяти, которые будут выделяться тонким томам. Второй предназначен для метаданных, которые хранят информацию о принадлежности блоков различным тонким томам. Далее они объединяются в тонкий пул томов, из которого впоследствии выделяются тонкие тома (см. Рис. 3).

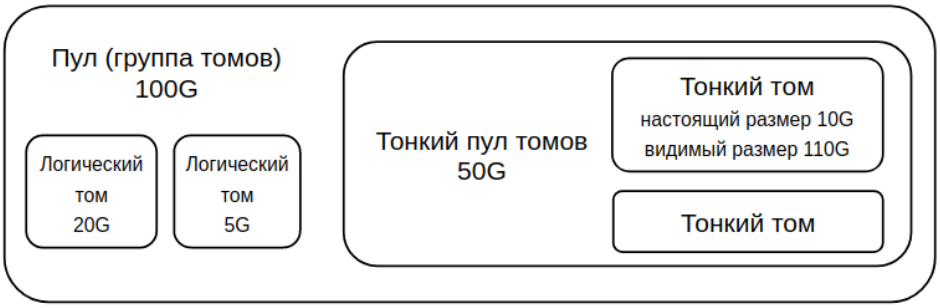

Рис. 3: Тонкий том LVM

Несмотря на видимый размер, при создании тонкий том пуст. Только при записи ему выделяются блоки памяти. Если место в тонком пуле томов закончится, запись в тонкие тома будет остановлена, пока память не освободится или том не расширится.

Поскольку блоки памяти для тонкого тома выделяются динамически, при записи приходится тратить время на поиск свободного места и выделение. Более того, данный подход увеличивает фрагментацию данных, что влияет на скорость чтения. Таким образом, тонкие тома позволяют оперировать большим виртуальным пространством, при этом

негативно влияя на производительность.

Тонкие снепшоты

Как и в случае тонкого тома, создать тонкий снепшот можно только в тонком пуле, и пространство под хранение данных выделено не будет. Сразу после создания тонкий снепшот — это указатели на блоки данных исходного тома (Рис. 4, пункт 1). Когда в исходный том произойдёт запись, будет создан новый блок с новыми данными, и указатель исходного тома с устаревшего блока перейдёт на новый (Рис. 4, пункт 2).

Рис. 4: Тонкий снепшот тонкого тома LVM

Логический том (в случае, когда он неактивен и read-only), тонкий том и тонкий снепшот могут стать исходным томом для тонкого снепшота. В случае, когда создаётся несколько тонких снепшотов одного исходного тома, одинаковые блоки не дублируются, а становятся общими благодаря использованию указателей, что позволяет значительно сокращать используемое пространство при хранении большого количества снепшотов.

В отличие от обычных снепшотов, тонкие снепшоты могут существовать и после удаления исходного тома, поскольку в таком случае будут удалены только новые, изменённые блоки.

Узнав процесс создания и взаимодействия с тонкими снепшотами, можно сделать вывод, что их использование замедляет производительность в сравнении с тонким томом без снепшота, но не значительно. Всё ещё необходимо тратить время на поиск свободного места для изменённого блока, зато больше нет необходимости копировать данные.

Параметры, влияющие на производительность

При создании снепшота имеется возможность задать параметр chunksize — размер блоков данных, которыми происходит копирование данных из исходного тома в снепшот. Аналогичный параметр есть и при создании тонких томов. Он задаёт размер блоков, которыми оперирует тонкий пул. Данный параметр может значительно влиять на производительность и требует подробного рассмотрения.

2.2.3. VDO

Virtualization Data Optimizer (VDO) [10],[7] — это модуль ядра, позволяющий сократить использование дискового пространства за счёт тонкого выделения, дедупликации, сжатия и устранения блоков данных, состоящих полностью из нулей.

VDO том

VDO позволяет создать том на основе блочного устройства, который после создания сам будет восприниматься другими программами как блочное устройство. В отличие от LVM, том VDO не может быть создан с использованием нескольких физических носителей. Также в VDO нет технологии создания нескольких томов на одном физическом устройстве.

Тома VDO всегда создаются с использованием технологии тонкого выделения. Данный подход реализуется при помощи карты блоков — оптимизированного по памяти префиксного дерева. Предоставляемое устройством пространство разбивается на физические блоки заданного пользователем размера, которым при помощи карты блоков ставятся в соответствие логические блоки. Впоследствии пользователь будет оперировать именно логическими блоками.

При создании VDO тома, доступное пространство разделяется на две части: раздел VDO и индекс UDS (Рис. 5). Раздел VDO — это пространство, используемое для хранения данных и связанных с ними ме-

таданных. Индекс UDS хранит уникальную подпись и адрес каждого дедуплицированного блока данных. Он состоит из двух частей:

- в RAM: подпись недавно записанных уникальных блоков, количество которых определяется дедупликационным окном;
- на диске: хранит подпись каждого уникального блока.

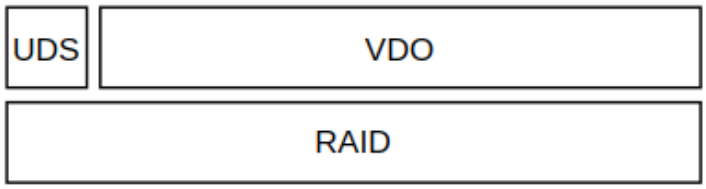

Рис. 5: VDO том

Существует два режима работы UDS индекса:

- dense — записывает код каждого уникального блока; в среднем требует 1G на хранилище физического размера 4T;
- sparse — опираясь на расположение данных, выбирает, какие блоки наиболее выгодно дедуплицировать; обеспечивает большее покрытие в сравнении с dense, но находит меньше дубликатов; в среднем требует 1G на хранилище физического размера 40T.

Обработка данных (устранение нулевых блоков, дедупликация, компрессия)

Решение VDO состоит из двух модулей:

- UDS модуль — взаимодействует с UDS (Universal Deduplication Service) индексом тома и анализирует данные на наличие уже существующих дубликатов;
- KVDO модуль — представляет блочное устройство, обрабатывает запросы на чтение/запись с использованием дедупликации, сжатия и тонкого выделения;

Рассмотрим поближе процессы чтения/записи на VDO том. При запросе на чтение KVDO модуль с помощью карты блоков находит физический блок данных, соответствующий логическому запрошенному блоку, и считывает информацию. VDO предоставляет несколько различных режимов записи данных. Алгоритм действия в случае режимов sync и async изображены на Рис. 6, различия в алгоритмах выделены серым.

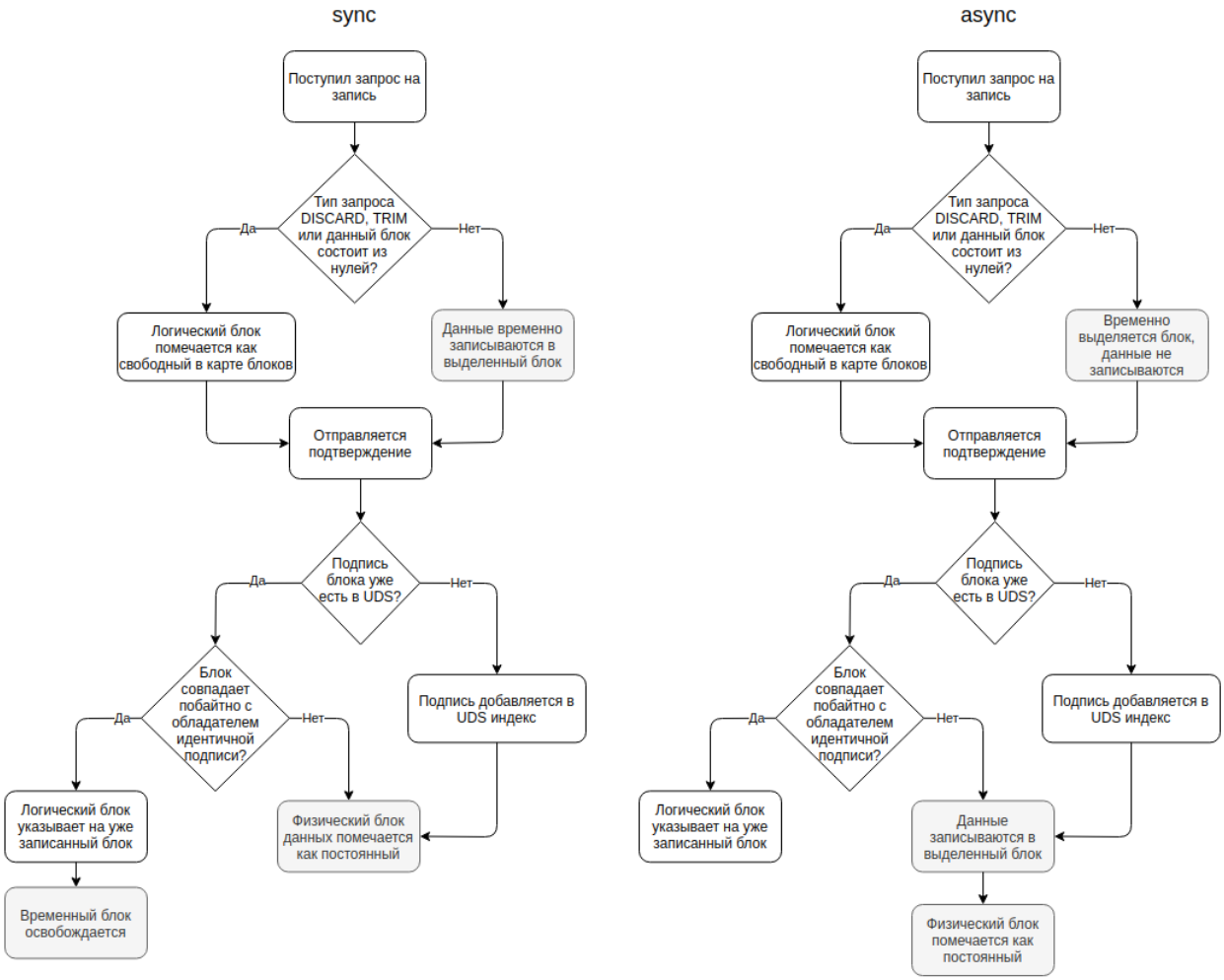

Рис. 6: Алгоритм записи блоков в том VDO

Помимо перечисленных выше sync и async алгоритмов присутствуют:

- async-unsafe — аналогичен алгоритму async, но не соответствует принципам ACID; возможна потеря данных в файловых системах, подразумевающих выполнение ACID, зато имеет лучшую производительность в сравнении с async;

- auto — автоматически выбирает sync или async, в зависимости от характеристик системы, стоит по умолчанию.

Потоки

В VDO процесс обработки запросов ввода/вывода разбит на несколько этапов, под выполнение которых отводится различное, настраиваемое вручную количество потоков. Существуют следующие типы потоков:

- Logical zone threads — каждый поток переводит номера логических блоков в номера соответствующих им физических блоков на соответствующих потокам зонах кэша карты блоков;
- Physical zone threads — во время запросов на запись параллельно обрабатывают запросы к соответствующим областям физического устройства;
- I/O submission threads — передают операции ввода/вывода на устройство хранения;
- CPU-processing threads и Hash zone threads — используются для работы с интенсивным использованием ЦП, такой как хеширование или сжатие;
- I/O acknowledgement threads — подтверждают запросы пользователя на ввод/вывод.

Взаимодействие с LVM

Существуют три способа взаимодействия VDO с LVM (см. Рис. 7).

1. Том VDO может быть создан поверх логических томов LVM типа linear (просто том, без дополнительных функций) или stripe (“полоски” данных заданного размера по очереди записываются на разные физические тома; после записи “полоски” на последний

том, следующая записывается на первый). Это позволяет создавать том VDO поверх нескольких физических устройств, а также при необходимости расширить его.

2. Также том VDO может быть использован как физический том LVM. Его можно добавить в группу томов и получить логические тома LVM с дедупликацией и сжатием от VDO.

3. Совсем недавно в группе томов LVM появилась экспериментальная функция создания логического тома VDOLV, обладающего свойствами тома, созданного менеджером томов VDO [4].

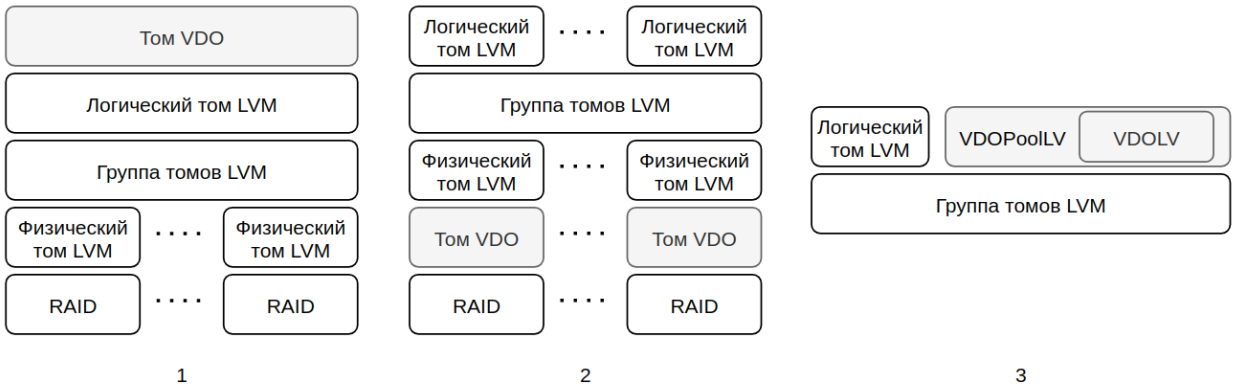

Рис. 7: Способы взаимодействия LVM и VDO

Параметры

При изучении VDO и работы “Performance testing of Virtual Data Optimizer storage layer” [7] как наиболее значимые параметры при настройке VDO можно выделить количество потоков каждого типа, размер кэша карты блоков, политику записи и discard size.

В рамках данной работы исследование влияния данных параметров на производительность проведено не будет в силу нехватки времени.

Влияние на производительность

- VDO тратит время на проверку содержимого блока (состоит ли он целиком из нулей), при этом в случае нулевых блоков запись не происходит, что ускоряет работу. Поэтому влияние данной функциональности на производительность не очевидно.

- Все тома VDO тонкие, поэтому тратится дополнительное время на поиск свободного места при записи и поиск нужного фрагмента при чтении, что замедляет производительность.

- Дедупликация и сжатие требуют дополнительных временных затрат. При этом данные процессы происходят после отправки подтверждения, и их обработкой занимаются различные потоки, что снижает негативное влияние на производительность. Более того, в случае асинхронного алгоритма, запись не происходит вообще в случае, когда найден дубликат, что ускоряет работу. Следовательно, влияние данных функций на производительность не очевидно, требуется дополнительное тестирование.

2.2.4. Вывод

ZFS в первую очередь является файловой системой, а значит не подходит пользователям, которым нужно только блочное устройство. Это стало одной из основных причин выбора LVM и VDO, все функции которых реализованы на уровне блочного устройства. Также немаловажными факторами являются возможность использования тома VDO как одного из логических томов LVM и уже проведённая интеграция LVM в ERA.

2.3. Бенчмарки

Бенчмарк — это инструмент для оценки производительности. Существует множество бенчмарков, оптимальных для конкретных задач. Чтобы получить полное представление о работе СХД с различными типами нагрузки, необходимо провести тестирование с помощью комплекса бенчмарков.

Для оценки производительности и функциональности при максимальных нагрузках в компании RAIDIX используются бенчмарки fio и vdbench. Рассмотрим их поближе.

Fio [13] — это бенчмарк для оценки производительности процессоров и памяти, предназначенный для СХД на Linux. Fio создаёт потоки записи и чтения данных с дисков, следуя заданным в конфигурационном файле (формата .fio) настройкам и условиям теста. Например, на Рис. 8 (справа) задаются следующие параметры: способ ввода/вывода, наличие буферизации, время обработки информации, количество модулей ввода/вывода, вид отчётности, размер блоков информации, тип и объекты нагрузки. Обычно Fio используется для генерации однотипной нагрузки (например, 100 % операций — последовательное чтение).

```
[global]
ioengine=libaio
direct=1
runtime=120
iodepth=2
group_reporting
blocksize=1M
rw=read

[job1]
filename=/dev/sdc

[job2]
filename=/dev/sdd
```

Рис. 8: Пример конфигурационного файла Fio

Vdbench [11] — это бенчмарк для генерации I/O-нагрузки на дисковые системы, позволяющий оценить производительность и целостность СХД. Vdbench, как и Fio, генерирует нагрузку, опираясь на данные из конфигурационного файла, пример которого можно увидеть на Рис. 9. Первая строка задаёт устройства, на которые будет идти нагрузка, вторая строка описывает тип нагрузки (процент операций чтения, размер блока данных...), а третья — саму нагрузку (верхнее ограничение по количеству ввода/вывода в секунду, время работы, промежуток между выводом данных...). Vdbench поддерживает журнал со всеми операциями записи, что позволяет проверить целостность данных. В отличие от Fio, vdbench используется для генерации более естественных нагрузок (например, 40 % операций последовательного чтения, 30 % — случай-

ного чтения, 30 % — случайной записи).

```
sd=sd1,lun=/dev/sdd,openflags=o_direct
wd=wd1,sd=sd1,xfersize=4096,rdpct=100
rd=run1,wd=wd1,iorate=100,elapsed=10,interval=1
```

Рис. 9: Пример конфигурационного файла Vdbench

3. Анализ рабочей нагрузки

Тестируемый в рамках данной работы программный RAID ERA, а также тома LVM и VDO, обладают интерфейсом блочного устройства. Это означает, что запись и чтение данных происходят блоками заданного размера. Любой запрос включает в себя следующие параметры:

- адрес блока, к которому происходит запрос (lba);
- время прихода запроса (time);
- тип запроса — чтение или запись (read/write);
- размер запроса (bs — block size).

Существует ещё один значимый параметр, который можно вывести из представленных выше. Допустим, размер предыдущего запроса равен bs, его адрес равен lba, а текущего — lba + bs, и между запросами прошло не более t времени. В таком случае запросы считаются последовательными, иначе — случайными. Таким образом запросы можно разделить на четыре типа: random read (случайные запросы на чтение), random write (случайные запросы на запись), sequential read (последовательные запросы на чтение), sequential write (последовательные запросы на запись).

Главными свойствами, характеризующими СХД, являются надёжность, производительность и величина задержки (время от момента создания запроса до завершения его обработки). Основными метриками, описывающими работу СХД, являются:

- BW — пропускная способность — максимальная скорость передачи данных по заданному пути (используется для измерения производительности при запросах большого размера);
- IOPS — среднее количество операций ввода/вывода в секунду (используется для измерения производительности при запросах малого размера).

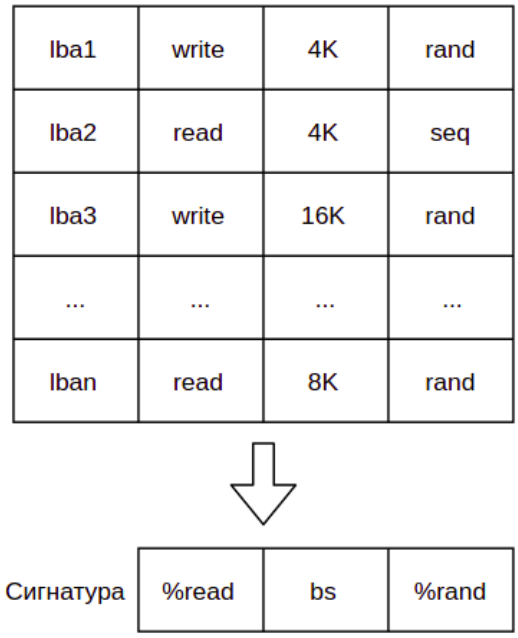

Рис. 10: Создание сигнатуры

При разработке СХД основными задачами являются максимизация вышеперечисленных характеристик и снижение величины задержки. Поскольку данная работа заключается в измерении и оптимизации производительности, задача снижения задержки рассматриваться не будет. Следовательно, в параметре времени нет необходимости.

Введём понятие сигнатуры. Пусть за время t на СХД поступило n запросов. Если сосчитать процент случайных запросов, процент запросов на чтение и среднее взвешенное размеров блоков и объединить их в вектор, получится сигнатура данной рабочей нагрузки (см. Рис. 10).

В статье “Intelligent Block Level I/O workload characterization for a temporal and spatial locality aware workload generator” [2] приведены примеры нагрузок реальных приложений, по которым можно получить сигнатуры реальных приложений (см. Рис. 11).

Application	Block Size in Bytes	% read/write	% Rand/Seq	I/O Performance Metric
Web File Server	4KB, 8KB, 64KB	95%/5%	75%/25%	IOPS
Database Online Transaction Processing (OLTP)	8KB	70%/30%	100%/0%	IOPS
Exchange Email	4KB	67%/33%	100%/0%	IOPS

Рис. 11: Пример реальных нагрузок ©[2]

4. Тестирование

Помимо трёх параметров, описывающих сигнатуру (%rand, %read, %bs), в каждом тесте присутствуют другие, неизменные значения:

- время тестирования (3 минуты);
- глубина — количество одновременных операций ввода/вывода (32);
- количество потоков (32 в случае fio, 8 в случае vdbench);
- входные точки для каждого из потоков равномерно распределены по физическому устройству.

Для тестирования будут использоваться следующие тестовые сервера:

- устройство 1 с характеристиками: Oracle Linux Server 8.3, версия ядра 5.4, Intel(R) Xeon(R) CPU E5-2620 v4 (32x 2.10GHz), 32G RAM; на нём был создан программный RAID ERA 5 уровня на 5 nvme дисках размера 1.5T;
- устройство 2 с характеристиками: Oracle Linux Server 8.3, версия ядра 3.10, AMD EPYC 7552 48-Core Processor (96x 2.2GHz), 125G RAM; на нём был создан программный RAID ERA 6 уровня на 16 nvme дисках размера 780G.

Для оценки производительности необходимо сгенерировать нагрузку на тестируемое физическое устройство таким образом, чтобы оно обрабатывало максимально возможное в данной ситуации количество запросов. Считается, что погрешность при измерении производительности продуктов RAIDIX — 10%.

4.1. Сравнение тонкого выделения и снепшотов с различными комбинациями LVM и VDO

Данное тестирование проводилось при помощи Fio с базовыми нагрузками, используемыми в компании RAIDIX, для первоначального сравнения технологий с не оптимизированными значениями параметров и

приблизительной оценки ухудшения производительности относительно программного RAID ERA и обычного тома LVM. Впоследствии было проведено повторное тестирование в случаях использования снепшотов, тонкого выделения или тонких снепшотов LVM с оптимальными значениями параметров. Их результаты также будут представлены в таблицах с результатами (см. Рис.12, 13, 14).

Тонкое выделение

Сравнивается производительность следующих устройств/томов:

- era — программный RAID ERA 5 уровня (уровень чередования блоков с распределением чётности), созданный на устройстве 1; все последующие тома создаются на его основе;
- lv — обычный том LVM;
- thin lv — тонкий том LVM;
- opt thin lv — тонкий том LVM с оптимальным значением параметра;
- thin vdolv — том с тонким выделением VDOLV, обладающего свойствами тома, созданного с помощью VDO;
- thin vdo — тонкий том VDO;
- lv thin vdo — тонкий том VDO, созданный поверх обычного тома LVM;
- thin vdo lv — обычный том LVM, созданный поверх тонкого тома VDO.

На Рис. 12 изображены результаты тестирования. Слева серым выделены параметры, задающие тип нагрузки. Символ k в результатах тестирования обозначает 1000. Исходя из представленных выше результатов можно сделать следующие выводы:

rand	read	bs	era	lv	thin lv	opt thin lv	thin vdolv	thin vdo	lvm thin vdo	thin vdo lvm
100	100	4K	2807k	2578k	155k	156k	378k	461k	455k	385k
100	0	4K	306k	280k	1120	20.6k	32.4k	38.4k	37.5k	38k
0	100	4K	2702k	2455k	57.8k	314k	384k	463k	461k	385k
0	0	4K	294k	279k	182k	151k	148k	147k	147k	148k
0	100	1M	14.9k	15.0k	2395	2390	1495	1471	1478	1501
0	0	1M	2547	2331	1872	5907	595	606	605	613
100	30	4K	444k	397k	1265	29.2k	54.1k	73k	73.4k	71.3k
100	50	4K	590k	512k	2238	40.6k	68.2k	85.2k	85.4k	85.1k
100	70	4K	806k	767k	5291	50.9k	85.2k	106.1k	106.3k	105k

Рис. 12: Сравнение различных томов с тонким выделением

- тонкое выделение значительно замедляет работу устройств;
- правильно подобранные параметры способны значительно ускорить обработку запросов;
- даже с оптимальными параметрами тонкое выделение LVM в производительности проигрывает тонкому выделению VDO на патернах с рандомными запросами, и не на много выигрывает при последовательной нагрузке с большими блоками;
- тонкий том VDOLV и том LVM, созданный поверх тонкого тома VDO, уступают в производительности VDO, зато выигрывают у LVM;
- тонкий том VDO, созданный поверх тома LVM, не уступает в производительности тому VDO.

Снепшоты

Сравнивается производительность следующих устройств/томов:

- era — программный RAID ERA 5 уровня (уровень чередования блоков с распределением чётности), созданный на устройстве 1; все последующие тома создаются на его основе;

- lv — обычный том LVM;
- lv snap — обычный том LVM с обычным снепшотом;
- opt lv snap — обычный том LVM с обычным снепшотом; в случаях, когда оптимальное значение параметра совпадает со значением по умолчанию, в таблицу заносился результат тестирования lv snap;
- thin lv — тонкий том LVM;
- opt thin lv — тонкий том LVM с оптимальным значением параметра;
- thin lv snap — тонкий том LVM с тонким снепшотом;
- opt thin lv snap — тонкий том LVM с тонким снепшотом с оптимальным параметром;
- thin vdo lv — обычный том LVM, созданный поверх тонкого тома VDO;
- thin vdo lv snap — обычный том LVM с обычным снепшотом, созданный поверх тонкого тома VDO;
- opt thin vdo lv snap — обычный том LVM с обычным снепшотом с оптимальным параметром, созданный поверх тонкого тома VDO; в случаях, когда оптимальное значение параметра совпадает со значением по умолчанию, в таблицу заносился результат тестирования thin vdo lv snap.

На Рис. 13, 14 изображены результаты тестирования. Исходя из них можно сделать следующие выводы:

- обычные снепшоты предсказуемо не влияют на производительность при чтении и значительно ухудшают её при записи;
- тонкие тома с тонкими снепшотами LVM не уступают в производительности тонким томам без снепшотов;

rand	read	bs	era	lv	lv snap	opt lv snap
100	100	4K	2807k	2578k	2347k	2347k
100	0	4K	306k	280k	23.6k	23.6k
0	100	4K	2702k	2455k	2275k	2275k
0	0	4K	294k	279k	22.7k	123k
0	100	1M	14.9k	15.0k	14.0k	14.9k
0	0	1M	2547	2331	79	1030
100	30	4K	444k	397k	33.1k	33.1k
100	50	4K	590k	512k	46.4k	46.4k
100	70	4K	806k	767k	76.3k	76.3k

Рис. 13: Оценка производительности обычных томов со снепшотами относительно программного RAID и обычного тома

rand	read	bs	era	thin lv	opt thin lv	thin lv snap	opt thin lv snap	thin vdo lv	thin vdo lv snap	opt thin vdo lv snap
100	100	4K	2807k	155k	156k	160k	151k	385k	331k	331k
100	0	4K	306k	1120	20.6k	2259	22.5k	38k	34.8k	34.8k
0	100	4K	2702k	57.8k	314k	65.0k	320k	385k	336k	336k
0	0	4K	294k	182k	151k	179k	151k	148k	39.0k	60.4k
0	100	1M	14.9k	2395	2390	2394	2440	1501	1528	1526
0	0	1M	2547	1872	5907	1932	6402	613	132	245
100	30	4K	444k	1265	29.2k	2075	32.1k	71.3k	44.7k	44.7k
100	50	4K	590k	2238	40.6k	3914	43.6k	85.1k	60.2k	60.2k
100	70	4K	806k	5291	50.9k	6648	54.1k	105k	83.6k	83.6k

Рис. 14: Сравнение производительности тонких томов с тонкими снепшотами

- обычные тома с обычными снепшотами, созданные поверх VDO тома, замедляют производительность, но всё ещё эффективнее тонких томов с тонкими снепшотами LVM.

Вывод

Тома, созданные с помощью технологии тонкого выделения VDO, показывают лучшую производительность в сравнении с LVM. Тем не менее в LVM есть функции, не реализованные в VDO. Комбинируя эти менеджеры томов можно получить необходимую функциональность с наилучшей производительностью.

4.2. Подбор значения параметра chunksize для LVM

При создании тонкого тома или тома со снепшотом имеется возможность задать значение параметра chunksize, который определяет:

- для тонкого тома — размер блоков, которыми происходит выделение нового пространства для тонкого тома;
- для снепшота — размер блоков, которыми происходит копирование данных из исходного тома в снепшот.

Изначально было проведено тестирование с использованием сигнатур, основанных на реальных нагрузках, с различными значениями параметра. Результаты некоторых из них можно видеть на Рис. 15. На нём слева задаются название приложения и сигнатура, а в самой таблице указана производительность тонкого тома с различными значениями параметра chunksize в IOPS. Ярко зелёным цветов выделены наилучшие результаты, красным — наихудшие.

				chunksize				
Application	**bs**	**% rand**	**% read**	**64K**	**128K**	**256K**	**512K**	**1M**
Exchange Email	4K	100	67	85561.2	23012.1	12729.1	6460	3377.9
Decision Support Systems (DSS)	1M	100	100	7441.3	9365.2	10222.7	10356.7	11048.2
Media Streaming	64K	0	98	85991.8	98010.8	111764.6	105986.6	58756.1

Рис. 15: Производительность тонких томов с разными снепшотами на реальных нагрузках

Изучив Рис. 15 можно сделать следующие выводы:

- параметр chunksize значительно влияет на производительность тонкого тома;
- при разных сигнатурах оптимальны различные значения параметра;
- иногда несколько значений могут быть оптимальными (с учётом погрешности измерений в 10%).

Следовательно, для эффективного использования тонких томов LVM важно изучить зависимость оптимального значения параметра chunksize от сигнатуры. Для этого необходимо расширить количество примеров путём создания и тестирования набора искусственных сигнатур, определить стратегии выбора оптимального параметра, а также создать инструмент, позволяющий по сигнатуре определять наиболее оптимальное значение параметра chunksize.

Также было сделано предположение, что оптимизированные тонкий том и снепшоты будут закономерно влиять на производительность. Следовательно, стоит рассмотреть создание инструмента, способного по сигнатуре и производительности обычного тома LVM на данной нагрузке предсказать процент падения производительности тома с дополнительной функциональностью.

4.2.1. Тесты

Тестирование проводилось с помощью бенчмарка vdbench, так как он лучше подходит для генерации смешанной, приближенной к реальности, нагрузки. Для тестирования были созданы следующие наборы тестов:

- примеры реальных нагрузок из работы “Intelligent Block Level I/O workload characterization for a temporal and spatial locality aware workload generator” [2] и производные от них;

- “сетка” из всех возможных комбинаций следующих значений rand, read и bs:
  - rand: 0%, 30%, 50%, 70%, 100%
  - read: 0%, 30%, 50%, 70%, 100%
  - bs: 4K, 16K, 64K, 256K, 1024K
- ряд тестов для более полного покрытия всех возможных ситуаций.

4.2.2. Тонкое выделение

При работе с тонкими томами и снепшотами параметр chunksize определяет размер чанков, которые будут выделяться тонким томам по мере необходимости. Данный параметр должен быть равен числу, являющемуся степенью 2 от 64K до 1G. Поскольку данный промежуток достаточно велик, практическая необходимость использовать большой chunksize мала, а разница в производительности тестируемого устройства рассматриваемыми тестами после 1M незначительна, исследуемый промежуток ограничен сверху 1M.

В случаях, когда при разных значениях параметра производительность не отличается более чем на 10% (погрешность при тестировании), оптимальное значение выбирается следующим образом:

1. Выделяется значение параметра, при котором производительность наиболее высокая.
2. Выделяются все значения, при которых производительность уступает максимальной не более чем на 10%.
3. Среди выделенных фиксируется наибольшее значение параметра. Именно оно считается оптимальным для конкретной нагрузки.

Данный алгоритм был выбран, поскольку каждый новый блок данных, выделенный для тонкого тома, влечёт создание метаданных. Поэтому увеличивая размер чанков, мы уменьшаем их количество, тем самым сокращая пространство, необходимое для хранения метаданных.

4.2.3. Снепшоты

При создании обычного снепшота параметр chunksize задаёт размер блоков (далее чанков), которыми будет происходить копирование из исходного тома в снепшот. Данный параметр должен быть равен числу, являющемуся степенью 2 от 4K до 512K.

В случаях, когда при разных значениях параметра производительность не отличается более чем на 10% (погрешность при тестировании), оптимальное значение выбирается следующим образом:

1. Выделяется значение параметра, при котором производительность наиболее высокая.
2. Выделяются все значения, при которых производительность уступает максимальной не более чем на 10%.
3. Среди выделенных фиксируется размер чанков, наиболее близкий к размеру блоков. Именно он считается оптимальным для конкретной нагрузки.

Данный алгоритм был выбран, поскольку в большинстве ситуаций кажется эффективным производить копирование в снепшот чанками того же размера, что и блоки в нагрузке.

4.2.4. Результаты

На Рис. 16 представлены результаты подбора параметров для томов со снепшотами, тонких томов и тонких томов с тонкими снепшотами соответственно на тестах из сетки на устройстве 1 и устройстве 2. Строки соответствуют параметрам % rand и % read, столбцы — параметру bs. В самой же таблице указаны оптимальные значения параметра chunksize в килобайтах для каждого типа нагрузки, выбранные в соответствии с описанными выше алгоритмами.

Обычный том со снепшотом

		blocksize				
rand	**read**	**4**	**16**	**64**	**256**	**1024**
0	0	512	512	128	256	512
	0.3	512	512	64	256	512
	0.5	512	256	64	256	512
	0.7	512	16	64	256	512
	1	4	16	64	256	512
0.3	0	4	16	64	256	512
	0.3	4	16	64	256	512
	0.5	4	16	64	256	512
	0.7	4	16	64	256	512
	1	4	16	64	256	512
0.5	0	4	16	64	256	512
	0.3	4	16	64	256	512
	0.5	4	16	64	256	512
	0.7	4	16	64	256	512
	1	4	16	64	256	512
0.7	0	4	16	64	256	512
	0.3	4	16	64	256	512
	0.5	4	16	64	256	512
	0.7	4	16	64	256	512
	1	4	16	64	256	512
1	0	4	16	64	256	512
	0.3	4	16	64	256	512
	0.5	4	16	64	256	512
	0.7	4	16	64	256	512
	1	4	16	64	256	512

Тонкий том

		blocksize				
rand	**read**	**4**	**16**	**64**	**256**	**1024**
0	0	1024	1024	64	256	1024
	0.3	1024	1024	1024	256	1024
	0.5	1024	1024	1024	256	1024
	0.7	1024	1024	64	256	1024
	1	128	512	1024	1024	1024
0.3	0	64	128	64	256	1024
	0.3	64	128	64	256	1024
	0.5	128	128	64	256	1024
	0.7	128	256	64	256	1024
	1	1024	1024	1024	1024	1024
0.5	0	128	128	64	256	1024
	0.3	128	128	64	256	1024
	0.5	128	256	64	256	1024
	0.7	128	256	64	256	1024
	1	1024	1024	1024	1024	1024
0.7	0	128	128	64	256	1024
	0.3	128	128	64	256	1024
	0.5	128	128	64	256	1024
	0.7	128	128	64	256	1024
	1	1024	1024	1024	1024	1024
1	0	128	128	64	256	1024
	0.3	128	128	64	256	1024
	0.5	128	128	64	256	1024
	0.7	64	64	64	256	1024
	1	1024	1024	1024	1024	1024

Тонкий том с тонким снепшотом

		blocksize				
rand	**read**	**4**	**16**	**64**	**256**	**1024**
0	0	1024	1024	64	256	1024
	0.3	1024	1024	1024	256	1024
	0.5	1024	1024	1024	256	1024
	0.7	1024	1024	1024	256	1024
	1	128	512	1024	1024	1024
0.3	0	64	128	64	256	1024
	0.3	128	128	64	256	1024
	0.5	128	128	64	256	1024
	0.7	128	256	64	256	1024
	1	1024	1024	1024	1024	1024
0.5	0	128	128	64	256	1024
	0.3	128	128	64	256	1024
	0.5	128	256	64	256	1024
	0.7	128	256	64	256	1024
	1	1024	1024	1024	1024	1024
0.7	0	128	128	64	256	1024
	0.3	128	128	64	256	1024
	0.5	128	128	64	256	1024
	0.7	128	128	64	256	1024
	1	1024	1024	1024	1024	1024
1	0	128	128	64	256	1024
	0.3	128	128	64	256	1024
	0.5	128	128	64	256	1024
	0.7	128	64	64	256	1024
	1	1024	1024	1024	1024	1024

Рис. 16: Оптимальные значения chunksize в зависимости от типа нагрузки

После изучения таблицы можно отметить, что, хотя закономерности заметны, оптимальные значения не всегда очевидны. В связи с этим было принято решение создать расширение для автоматической генерации рекомендаций оптимального размера chunksize для конкретной нагрузки.

5. Модуль подбора chunksize

5.1. QoSmic

На данный момент существует функциональное расширение ПО RAIDIX — QoSmic [14]. Его главная цель — на основе результатов анализа запросов, исходящих от различных инициаторов, оценить значимость работы конкретных приложений и расставить соответствующие приоритеты. Таким образом QoSmic позволяет автоматически оптимизировать производительность СХД, ограничивая обработку менее важных запросов от нецелевых программ и служебных утилит.

Рассмотрим принцип работы СХД с расширением QoSmic, опираясь на схему, изображённую на Рис. 17.

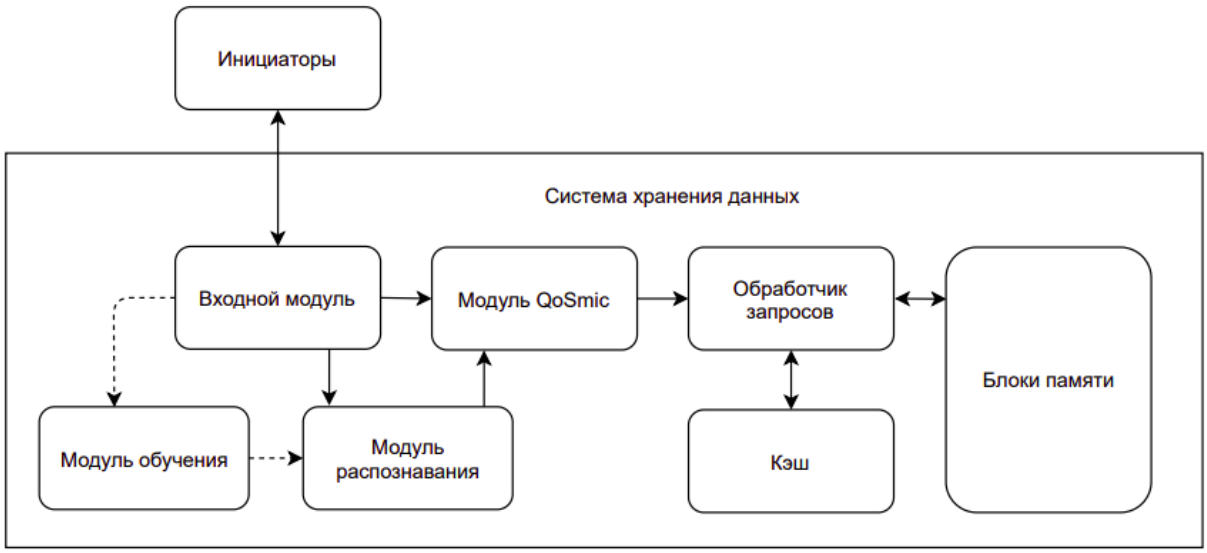

Рис. 17: Схема СХД с расширением QoSmic

Во время работы СХД Инициаторы посылают запросы во Входной модуль. Запросы фиксируются в течение заданного времени в Модуле распознавания. Дальнейшие действия зависят от режима работы QoSmic.

- Режим обучения (происходит в Модуле обучения): на основе собранных запросов создаётся сигнатура, которая в дальнейшем будет характеризовать конкретное приложение; таким образом происходит “знакомство” СХД с новым приложением.

- Режим распознавания (происходит в Модуле распознавания): в режиме реального времени происходит идентификация приложений с помощью уже существующих сигнатур; классификация происходит при помощи алгоритма машинного обучения Random Forest [9],[12].

Затем на основе заданных правил назначения приоритета и результатов, полученных на предыдущих этапах, инициатору выставляется приоритет.

Алгоритм классификации Random Forest заключается в следующем:

1. Из набора данных случайным образом выбираются параметры и конкретные примеры.
2. По каждому набору строится дерево принятия решений — дерево, в котором каждая вершина, не являющаяся листом, представляет точку разбиения данных в соответствии с определённым критерием, а листья представляют финальное прогнозируемое значение.
3. Результат предсказаний каждого из деревьев усредняется. Таким образом Random Forest предсказывает верное значение.

Random Forest был выбран для классификации в силу следующих своих достоинств:

- модель обучается достаточно быстро;
- работа алгоритма завершается за фиксированное число операций;
- алгоритм способен обрабатывать большие объёмы данных;
- высокое качество получаемых моделей, сравнимое с нейронными сетями и ансамблями нейронных сетей, слишком тяжеловесными в данной ситуации;
- простота настройки.

Безусловно, данный алгоритм также обладает рядом недостатков, из которых наиболее ощутимый — склонность к переобучению на зашумленных данных.

5.2. Расширение для подбора chunksize

В будущем планируется интеграция подбора оптимального значения параметра chunksize в ПО RAIDIX в виде расширения. Алгоритм работы такого расширения представлен ниже.

1. По аналогии с режимом обучения из расширения QoSmic на основе накопленных логов создать сигнатуру.
2. С помощью сгенерированной сигнатуры определить оптимальный размер chunksize.
3. Создать том с искомой технологией на основе результата шага 2.

Для реализации шага 1 в будущем будет модифицирован алгоритм из расширения QoSmic таким образом, чтобы получаемая сигнатура соответствовала требованиям данной задачи, а именно имела вид (%rand, %read, bs).

Для реализации шага 2 на данный момент принято решение использовать алгоритм машинного обучения Random Forest в силу его вышеописанных достоинств, а также потому что он уже интегрирован в систему.

Для применения алгоритма классификации Random Forest необходимо подготовить исходные данные, выделить признаки (параметры, ориентируясь на которые модель будет принимать решение о сопоставлении класса определённой сигнатуре), настроить и обучить модель. Теория данных процессов подробно описана в статье Analysis and classification of multimedia I/O requests to storage system [1].

Подготовка обучающей выборки

В качестве признаков для обучающей модели используется сигнатура, выведенная в результате анализа рабочей нагрузки в главе 3 "Анализ рабочей нагрузки" (см. Рис. 10). В обучающую выборку вошли как сигнатуры реальных приложений, так и созданные искусственно. На данный момент выборка состоит из 220 различных сигнатур.

Классификация проводилась:

- для тонкого выделения — между степенями 2 от 64K до 1024K;
- для снепшотов — между степенями 2 от 4K до 512K.

Также было сделано предположение, что если добавить к трём вышеописанным признакам первоначальную производительность обычного тома LVM, появится возможность предсказать примерное поведение производительности. В данном случае классификация будет проводиться для классов 0, 10, 20, ..., 100, каждый из которых означает процент производительности тома с новой функциональностью относительно производительности обычного тома LVM.

5.3. Результаты

Изначально выборка была разбита в соотношении 80 к 20 для обучения и тестирования. Поскольку данная выборка мала и несбалансированна, для подбора параметра (количество деревьев в модели) использовалась кросс-валидация с пропорциональным распределением элементов между обучающей и тестовой выборкой. Ниже можно увидеть точность получившейся модели для каждой из ситуаций.
В таблице представлены две метрики:

- точность — количество верных классификаций разделённое на количество тестируемых примеров;

Тип тома	Подбор chunksize		Падение производительности	
	сбаланс. точность	точность	сбаланс. точность	точность
Том со снепшотом	0.75	0.89	0.52	0.79
Тонкий том	0.70	0.86	0.80	0.81
Тонкий том со снепшотом	0.72	0.89	0.73	0.79

Рис. 18: Точность предсказаний модели, обученной на текущей выборке

- сбалансированная точность — метрика, используемая для несбалансированных данных, которая учитывает разницу в количестве элементов, принадлежащих разным классам.

Поскольку тестирование требует много времени, выборка для обучения модели недостаточно велика, и разница в количестве элементов разных классов значительна. Как следствие, точность предсказаний далека от единицы. Тем не менее, уже сейчас модель показывает неплохие результаты. Следовательно, в будущем стоит продолжить тестирование, завершить обучение модели, возможно, использовать для решения данной задачи более совершенный алгоритм машинного обучения — XGboost.

Заключение

В результате проведённой работы были выполнены следующие задачи.

1. Изучены различные менеджеры томов.
2. Проведён анализ нагрузки; выделены атрибуты, на основе которых созданы сигнатуры.
3. Проведено сравнение производительности тонких томов и снепшотов, созданных с использованием менеджеров томов LVM и VDO.
4. Проведено исследование влияния параметра chunksize на производительность с целью создания базы для выявления зависимости параметра от сигнатуры.
5. Выделены признаки для обучения модели для подбора оптимального значения chunksize и оценки падения производительности.
6. Проведено первичное тестирование модели, обученной на уже имеющейся выборке.

Список литературы

[1] Analysis and classification of multimedia I/O requests to storage system. — URL: https://dl.acm.org/doi/10.1145/2687233.2687243 (online; accessed: 30.05.2021).

[2] Intelligent Block Level I/O workload characterization for a temporal and spatial locality aware workload generator. — URL: https://conservancy.umn.edu/bitstream/handle/11299/165578/Palanivel_umn_0130M_15120.pdf?sequence=1&isAllowed=y (online; accessed: 30.05.2021).

[3] LVM man page. — URL: https://www.mankier.com/package/lvm2 (online; accessed: 30.05.2021).

[4] LVMVDO man page. — URL: https://www.mankier.com/7/lvmvdo (online; accessed: 30.05.2021).

[5] Logical volume manager administration. — URL: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/logical_volume_manager_administration/index (online; accessed: 30.05.2021).

[6] OpenZFS man page. — URL: https://openzfs.org/wiki/Main_Page (online; accessed: 30.05.2021).

[7] Performance testing of Virtual Data Optimizer storage layer. — URL: https://is.muni.cz/th/rq7e2/petrovic_diploma_thesis.pdf (online; accessed: 30.05.2021).

[8] RAIDIX ERA. — URL: https://www.raidix.ru/products/era/ (online; accessed: 30.05.2021).

[9] Random Forest с примерами на R. — URL: http://www.algorithmist.ru/2012/05/random-forest-r.html (online; accessed: 30.05.2021).

[10] Red Hat Enterprise Linux 8. Deduplicating and compressing storage. — URL: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/deduplicating_and_compressing_storage/index (online; accessed: 30.05.2021).

[11] Vdbench users guide. — URL: https://www.oracle.com/technetwork/server-storage/vdbench-1901683.pdf (online; accessed: 30.05.2021).

[12] A complete guide to the Random Forest algorithm. — URL: https://builtin.com/data-science/random-forest-algorithm (online; accessed: 30.05.2021).

[13] Документация Fio. — URL: https://fio.readthedocs.io/en/latest/fio_doc.html#fio-flexible-i-o-tester-rev-version (online; accessed: 30.05.2021).

[14] Функциональное расширение QoSmic. — URL: https://raidix.ru/files/QoSmic_RU.pdf (online; accessed: 30.05.2021).