Основы ZFS: система хранения и производительность

Новости 1С-Битрикс (/info/news/) ›
Полезные статьи (/info/articles/) ›

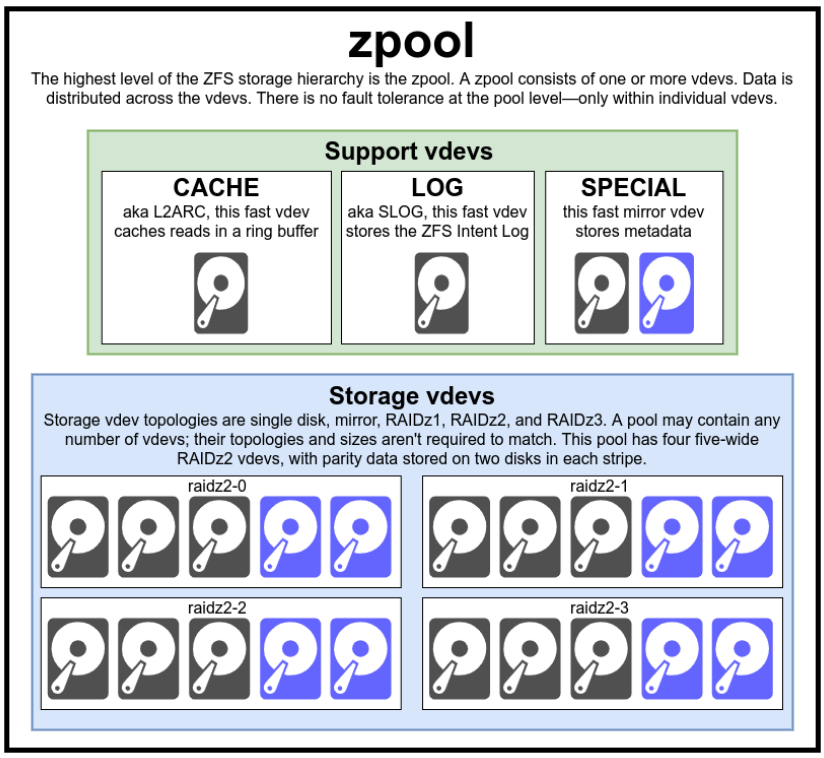
*Моя цель - предложение широкого ассортимента товаров и услуг на постоянно высоком качестве обслуживания по самым **выгодным ценам**.*

Этой весной мы уже обсудили некоторые вводные темы, например, как проверить скорость ваших дисков и что такое RAID. Во второй из них мы даже пообещали продолжить изучение производительности различных многодисковых топологий в ZFS. Это файловая система следующего поколения, которая сейчас внедряется повсюду: от Apple до Ubuntu.

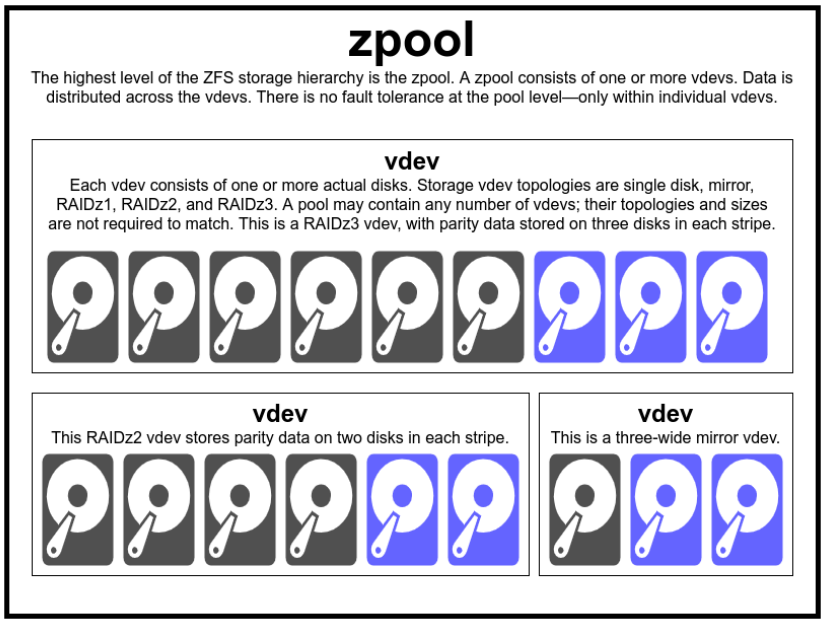
Ну что ж, сегодня самый подходящий день для знакомства с ZFS, любознательные читатели. Просто знайте, что по скромной оценке разработчика OpenZFS Мэтта Аренса, «это действительно сложно».

Но прежде чем мы доберемся до цифр — а они будут, обещаю — по всем вариантам восьмидисковой конфигурации ZFS, нужно поговорить о том, *как* вообще ZFS хранит данные на диске.

Zpool, vdev и device



Эта диаграмма полного пула включает в себя три вспомогательных vdev'а, по одному каждого класса, и четыре для RAIDz2



Обычно нет причин создавать пул из несоответствующих типов и размеров vdev — но если хотите, ничто не мешает вам это сделать

Чтобы действительно понять файловую систему ZFS, нужно внимательно посмотреть на её фактическую структуру. Во-первых, ZFS объединяет традиционные уровни управления томами и файловой системы. Во-вторых, она использует транзакционный механизм копирования при записи. Эти особенности означают, что система структурно очень отличается от обычных файловых систем и RAID-массивов. Первый набор основных строительных блоков для понимания: это пул хранения (zpool), виртуальное устройство (vdev) и реальное устройство (device).

zpool

Пул хранения zpool — самая верхняя структура ZFS. Каждый пул содержит одно или несколько

виртуальных устройств. В свою очередь, каждое из них содержит одно или несколько реальных устройств (device). Виртуальные пулы — это автономные блоки. Один физический компьютер может содержать два или более отдельных пула, но каждый полностью независим от других. Пулы не могут совместно использовать виртуальные устройства.

Избыточность ZFS находится на уровне виртуальных устройств, а не на уровне пулов. На уровне пулов нет абсолютно никакой избыточности — если какой-либо накопитель vdev или специальный vdev теряется, то вместе с ним теряется и весь пул.

Современные пулы хранения могут пережить потерю кэша или журнала виртуального устройства — хотя они могут потерять небольшое количество грязных данных, если потеряют журнал vdev во время отключения питания или сбоя системы.

Есть распространённое заблуждение, что «полосы данных» (страйпы) ZFS записываются через весь пул. Это неверно. Zpool — вовсе не забавный RAID0, это скорее забавный JBOD со сложным изменчивым механизмом распределения.

По большей части записи распределяются между доступными виртуальными устройствами в соответствии с имеющимся свободным пространством, так что теоретически все они будут заполнены одновременно. В более поздних версиях ZFS учитывается текущее использование (утилизацию) vdev — если одно виртуальное устройство значительно загруженнее другого (например, из-за нагрузки на чтение), его временно пропустят для записи, несмотря на наличие самого высокого коэффициента свободного пространства.

Механизм определения утилизации, встроенный в современные методы распределения записи ZFS, может уменьшить задержку и увеличить пропускную способность в периоды необычно высокой нагрузки — но это не *карт-бланш* на невольное смешивание медленных HDD и быстрых SSD в одном пуле. Такой неравноценный пул всё равно будет работать со скоростью самого медленного устройства, то есть как будто он целиком составлен из таких устройств.

vdev

Каждый пул хранения состоит из одного или нескольких виртуальных устройств (virtual device, vdev). В свою очередь, каждый vdev включает одно или несколько реальных устройств. Большинство виртуальных устройств используются для простого хранения данных, но существует несколько вспомогательных классов vdev, включая CACHE, LOG и SPECIAL. Каждый из этих типов vdev может иметь одну из пяти топологий: единое устройство (single-device), RAIDz1, RAIDz2, RAIDz3 или зеркало (mirror).

RAIDz1, RAIDz2 и RAIDz3 — это особые разновидности того, что олды назовут RAID двойной (диагональной) чётности. 1, 2 и 3 относятся к тому, сколько блоков чётности выделено для каждой полосы данных. Вместо отдельных дисков для обеспечения чётности виртуальные устройства RAIDz полуравномерно распределяют эту чётность по дискам. Массив RAIDz может потерять столько дисков, сколько у него блоков чётности; если он потеряет ещё один, то выйдет из строя и заберет с собой пул хранения.

В зеркальных виртуальных устройствах (mirror vdev) каждый блок хранится на каждом устройстве в vdev. Хотя наиболее распространённые двойные зеркала (two-wide), в зеркале может быть любое произвольное количество устройств — в больших установках для повышения производительности чтения и отказоустойчивости часто используются тройные. Зеркало vdev может пережить любой сбой, пока продолжает работать хотя бы одно устройство в vdev.

Одиночные vdev по своей сути опасны. Такое виртуальное устройство не переживёт ни одного сбоя — и если используется в качестве хранилища или специального vdev, то его сбой приведёт к уничтожению всего пула. Будьте здесь очень, очень осторожны.

Виртуальные устройства CACHE, LOG и SPECIAL могут быть созданы по любой из вышеперечисленных топологий — но помните, что потеря виртуального устройства SPECIAL означает потерю пула, поэтому настоятельно рекомендуется избыточная топология.

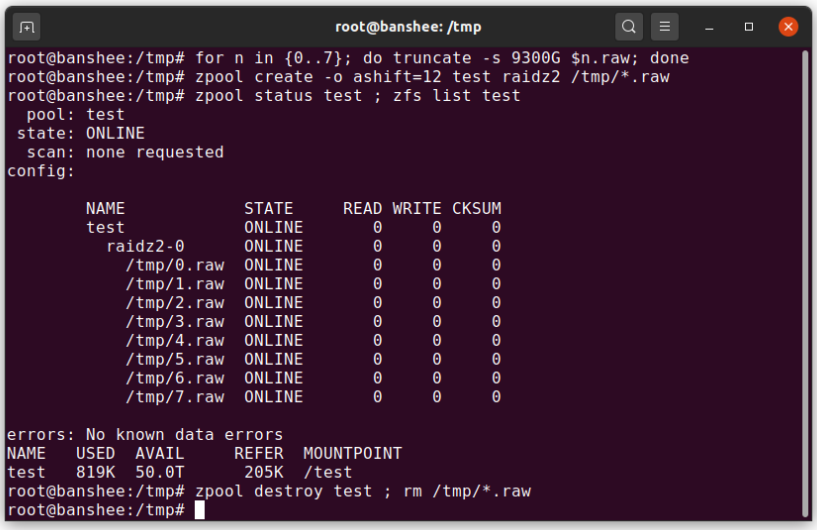
device

Вероятно, это самый простой для понимания термин в ZFS — это буквально блочное устройство произвольного доступа. Помните, что виртуальные устройства состоят из отдельных устройств, а пул сделан из виртуальных устройств.

Диски — магнитные или твёрдотельные — являются наиболее распространёнными блочными устройствами, которые используются в качестве строительных блоков vdev. Однако подойдёт

любой девайс с дескриптором в /dev — так что в качестве отдельных устройств можно использовать целые аппаратные RAID-массивы.

Простой raw-файл является одним из самых важных альтернативных блочных устройств, из которых может быть построен vdev. Тестовые пулы из разреженных файлов — очень удобный способ проверять команды пула и смотреть, сколько места доступно в пуле или виртуальном устройстве данной топологии.



Можете создать тестовый пул из разреженных файлов всего за несколько секунд — но не забудьте потом удалить весь пул и его компоненты

Допустим, вы хотите поставить сервер на восемь дисков и планируете использовать диски по 10 ТБ (~9300 ГиБ) — но вы не уверены, какая топология лучше всего соответствует вашим потребностям. В приведённом выше примере мы за считанные секунды строим тестовый пул из разреженных файлов — и теперь знаем, что RAIDz2 vdev из восьми дисков по 10 ТБ обеспечивает 50 ТиБ полезной ёмкости.

Ещё один особый класс устройств — SPARE (запасные). Устройства горячей замены, в отличие от обычных устройств, принадлежат всему пулу, а не одному виртуальному устройству. Если какой-то vdev в пуле выходит из строя, а запасное устройство подключено к пулу и доступно, то оно автоматически присоединится к пострадавшему vdev.

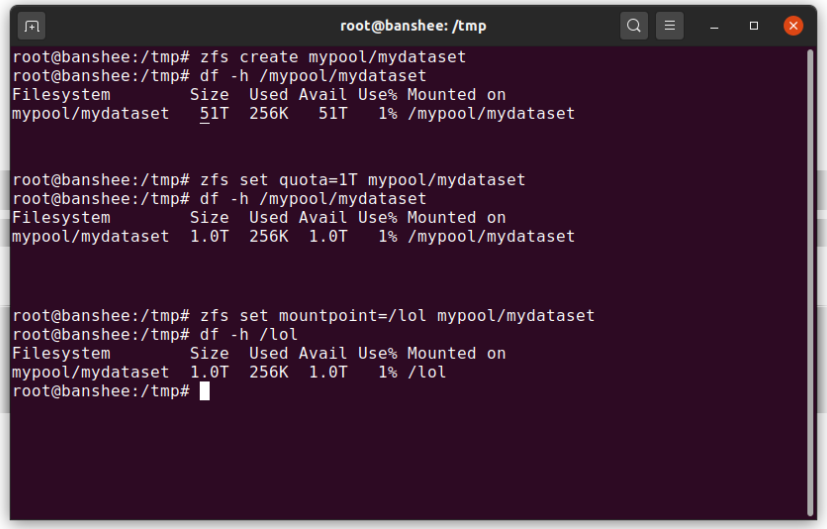
После подключения к пострадавшему vdev запасной девайс начинает получать копии или реконструкции данных, которые должны быть на отсутствующем устройстве. В традиционном RAID это называется восстановлением (rebuilding), а в ZFS это «восстановление избыточности» (resilvering).

Важно отметить, что запасные устройства не навсегда заменяют вышедшие из строя устройства. Это лишь временная замена для сокращения времени, в течение которого наблюдается деградация vdev. После того, как администратор заменил вышедшее из строя устройство vdev, то происходит восстановление избыточности на это постоянное устройство, а SPARE отсоединяется от vdev и возвращается к работе запасным на весь пул.

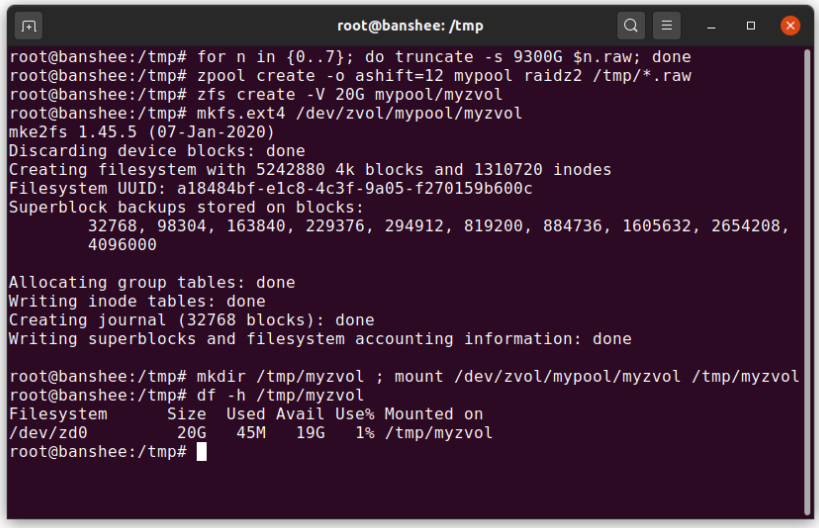
Наборы данных, блоки и секторы

Следующий набор строительных блоков, которые нужно понять в нашем путешествии по ZFS, относится не столько к аппаратному обеспечению, сколько к тому, как организованы и хранятся сами данные. Мы здесь пропускаем несколько уровней — таких как metaslab — чтобы не нагромождать детали, сохраняя понимание общей структуры.

Набор данных (dataset)



Когда мы впервые создаём набор данных, он показывает всё доступное пространство пула. Затем мы устанавливаем квоту — и меняем точку монтирования. Магия!



Zvol — это по большей части просто набор данных, лишённый своего слоя файловой системы, который мы заменяем здесь совершенно нормальной файловой системой ext4

Набор данных ZFS примерно аналогичен стандартной смонтированной файловой системе. Как и обычная файловая система, на первый взгляд он кажется «просто ещё одной папкой». Но также, как и у обычных монтируемых файловых систем, у каждого набора данных ZFS собственный набор базовых свойств.

Прежде всего, у набора данных может быть назначенная квота. Если установить `zfs set quota=100G poolname/datasetname`, то вы не сможете записать в смонтированную папку `/poolname/datasetname` больше, чем 100 ГиБ.

Заметили наличие — и отсутствие — слэшей в начале каждой строки? У каждого набора данных своё место как в иерархии ZFS, так и в иерархии системного монтирования. В иерархии ZFS нет ведущего слэша — вы начинаете с имени пула, а затем пути от одного набора данных к следующему. Например, `pool/parent/child` для набора данных с именем `child` под родительским набором данных `parent` в пуле с креативным названием `pool`.

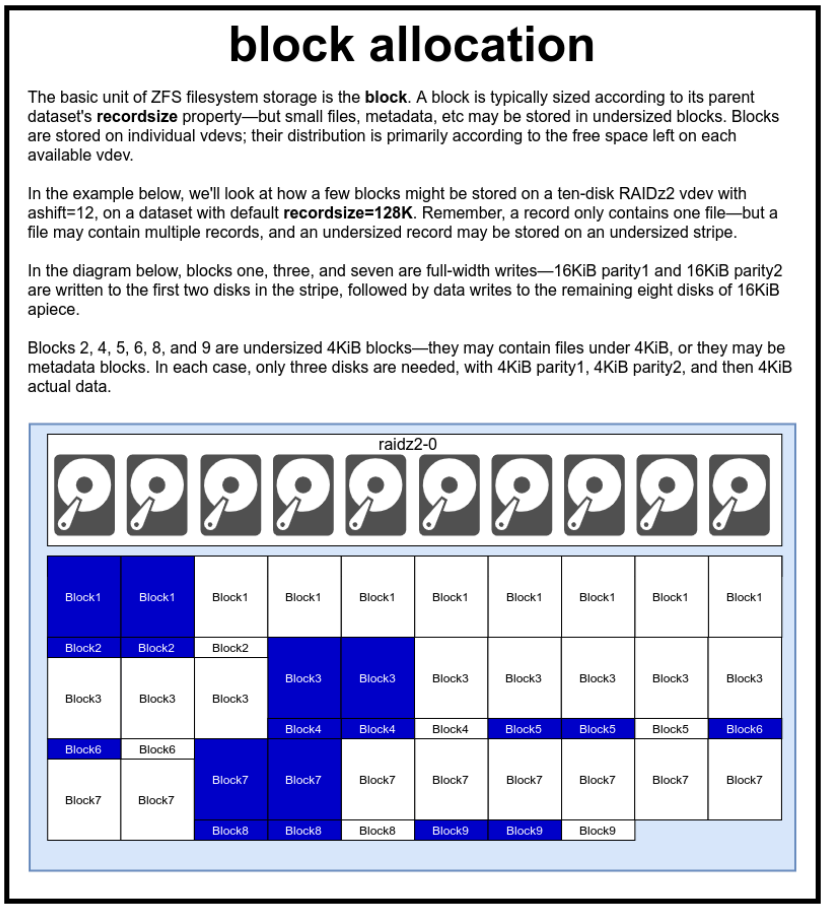
По умолчанию, точка монтирования набора данных будет эквивалентна его имени в иерархии ZFS, со слэшем в начале — пул с названием `pool` примонтируется как `/pool`, набор данных `parent` монтируется в `/pool/parent`, а дочерний набор данных `child` смонтируется в `/pool/parent/child`. Однако системную точку монтирования набора данных можно изменить.

Если мы укажем `zfs set mountpoint=/lol pool/parent/child`, то набор данных

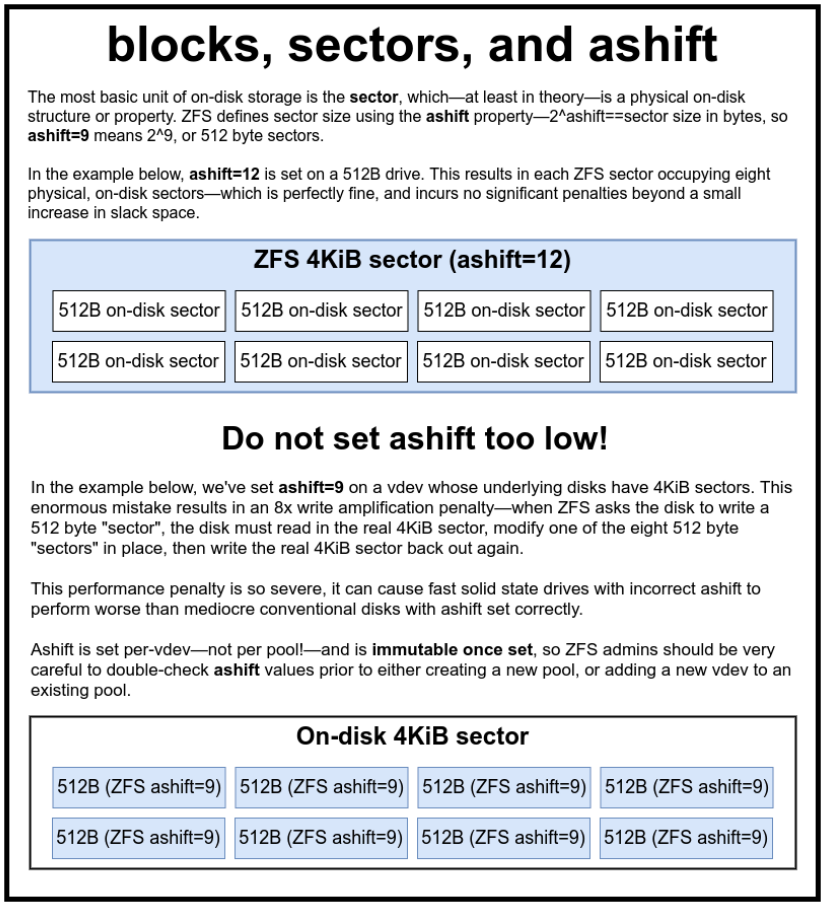
`pool/parent/child` смонтируется в систему как `/lol`.

В дополнение к наборам данных, мы должны упомянуть тома (zvols). Том примерно аналогичен набору данных, за исключением того, что в нём фактически нет файловой системы — это просто блочное устройство. Вы можете, например, создать `zvol` с именем `mypool/myzvol`, затем отформатировать его с файловой системой ext4, а затем смонтировать эту файловую систему — теперь у вас есть файловая система ext4, но с поддержкой всех функций безопасности ZFS! Это может показаться глупым на одном компьютере, но имеет гораздо больше смысла в качестве бэкенда при экспортировании устройства iSCSI.

Блоки



*Файл представлен одним или несколькими блоками. Каждый блок хранится на одном виртуальном устройстве. Размер блока обычно равен параметру **recordsize**, но может быть уменьшен до **2^ashift**, если содержит метаданные или небольшой файл.*



*Мы действительно, **действительно** не шутим по поводу огромного ущерба производительности, если установить слишком маленький ashift*

В пуле ZFS все данные, включая метаданные, хранятся в блоках. Максимальный размер блока для каждого набора данных определяется в свойстве `recordsize` (размер записи). Размер записи может изменяться, но это не изменит размер или расположение любых блоков, которые уже были записаны в набор данных — он действует только для новых блоков по мере их записи.

Если не определено иное, то текущий размер записи по умолчанию равен 128 КиБ. Это своего рода непростой компромисс, в котором производительность будет не идеальной, но и не ужасной в большинстве случаев. `Recordsize` можно установить на любое значение от 4K до 1M (с дополнительными настройками `recordsize` можно установить ещё больше, но это редко бывает хорошей идеей).

Любой блок ссылается на данные только одного файла — вы не можете втиснуть два разных файла в один блок. Каждый файл состоит из одного или нескольких блоков, в зависимости от размера. Если размер файла меньше размера записи, он сохранится в блоке меньшего размера — например, блок с файлом 2 КиБ займёт только один сектор 4 КиБ на диске.

Если файл достаточно велик и требует несколько блоков, то все записи с этим файлом будут иметь размер `recordsize` — включая последнюю запись, основная часть которой может оказаться неиспользуемым пространством.

У томов zvol нет свойства `recordsize` — вместо этого у них есть эквивалентное свойство `volblocksize`.

Секторы

Последний, самый базовый строительный блок — сектор. Это наименьшая физическая единица, которая может быть записана или считана с базового устройства. В течение нескольких десятилетий в большинстве дисков использовались секторы по 512 байт. В последнее время большинство дисков настроено на сектора 4 КиБ, а в некоторых — особенно SSD — сектора 8 КиБ или даже больше.

В системе ZFS есть свойство, которое позволяет вручную установить размер сектора. Это

свойство `ashift`. Несколько запутанно, что ashift является степенью двойки. Например, `ashift=9` означает размер сектора 2^9, или 512 байт.

ZFS запрашивает у операционной системы подробную информацию о каждом блочном устройстве, когда оно добавляется в новый vdev, и теоретически автоматически устанавливает ashift должным образом на основе этой информации. К сожалению, многие диски лгут о своём размере сектора, чтобы сохранить совместимость с Windows XP (которая была неспособна понять диски с другими размерами секторов).

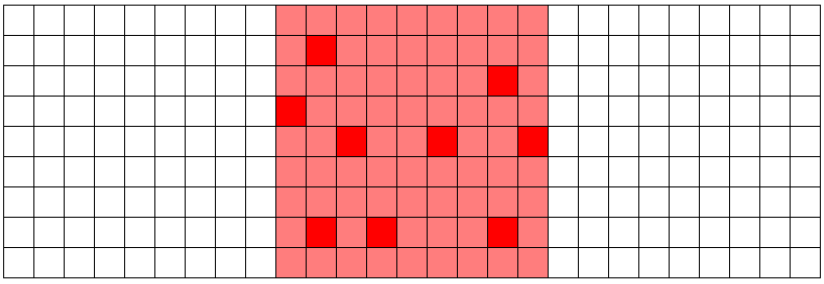
Это означает, что администратору ZFS настоятельно рекомендуется знать фактический размер сектора своих устройств и вручную устанавливать `ashift`. Если установлен слишком маленький ashift, то астрономически увеличивается количество операций чтения/записи. Так, запись 512-байтовых «секторов» в реальный сектор 4 КиБ означает необходимость записать первый «сектор», затем прочитать сектор 4 КиБ, изменить его со вторым 512-байтовым «сектором», записать его обратно в новый сектор 4 КиБ и так далее для каждой записи.

В реальном мире такой штраф бьёт по твёрдотельным накопителям Samsung EVO, для которых должен действовать `ashift=13`, но эти SSD врут о своём размере сектора, и поэтому по умолчанию устанавливается `ashift=9`. Если опытный системный администратор не изменит этот параметр, то этот SSD работает *медленнее* обычного магнитного HDD.

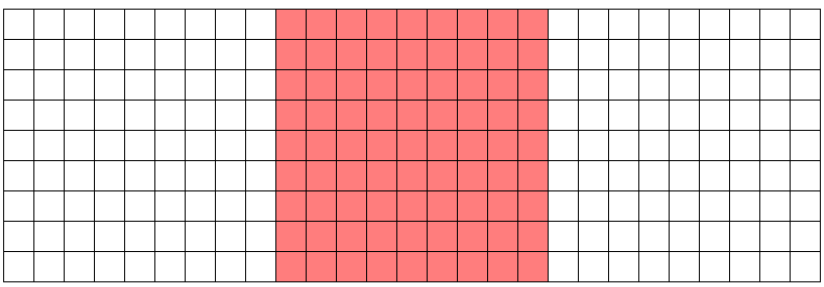
Для сравнения, за слишком большой размер `ashift` нет практически никакого штрафа. Реального снижения производительности нет, а увеличение неиспользуемого пространства бесконечно мало (или равно нулю при включённом сжатии). Поэтому мы настоятельно рекомендуем даже тем дискам, которые действительно используют 512-байтовые секторы, установить `ashift=12` или даже `ashift=13`, чтобы уверенно смотреть в будущее.

Свойство `ashift` устанавливается для каждого виртуального устройства vdev, а *не для пула*, как многие ошибочно думают — и не изменяется после установки. Если вы случайно сбили `ashift` при добавлении нового vdev в пул, то вы безвозвратно загрязнили этот пул устройством с низкой производительностью и, как правило, нет другого выхода, кроме как уничтожить пул и начать всё сначала. Даже удаление vdev не спасёт от сбитой настройки `ashift`!

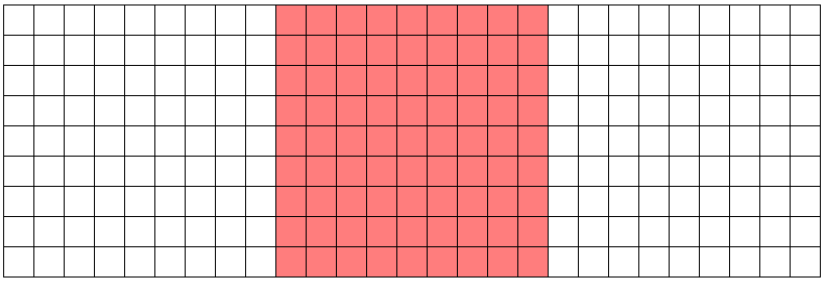
Механизм копирования при записи



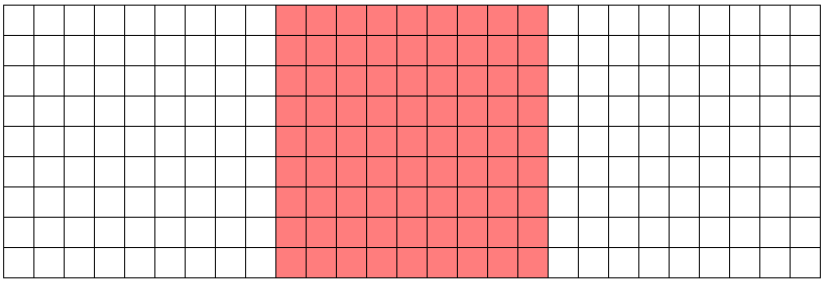
Если обычной файловой системе нужно перезаписать данные — она изменяет каждый блок там, где он находится



Файловая система с копированием при записи записывает новую версию блока, а затем разблокирует старую версию



В абстрактном виде, если игнорировать реальное физическое расположение блоков, то наша «комета данных» упрощается до «червя данных», который перемещается слева направо по карте доступного пространства



Теперь мы можем получить хорошее представление, как работают снапшоты копирования при записи — каждый блок может принадлежать нескольким снапшотам, и сохранится до тех пор, пока не будут уничтожены все связанные снапшоты

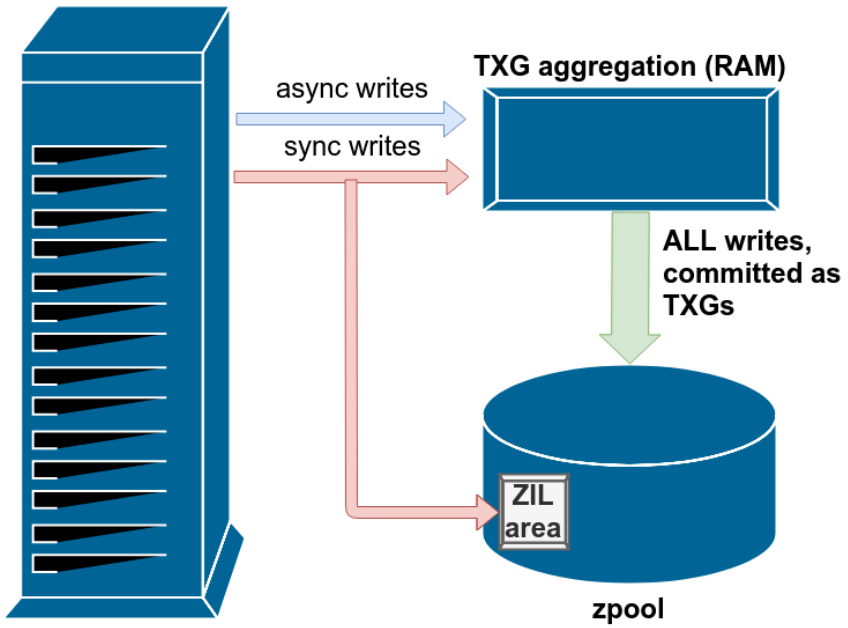
Механизм копирования при записи (Copy on Write, CoW) — фундаментальная основа того, что делает ZFS настолько потрясающей системой. Основная концепция проста — если вы попросите традиционную файловую систему изменить файл, она сделает именно то, что вы просили. Если вы попросите файловую систему с копированием при записи сделать то же самое, она скажет «хорошо» — но соврёт вам.

Вместо этого файловая система с копированием при записи записывает новую версию изменённого блока, а затем обновляет метаданные файла, чтобы разорвать связь со старым блоком и связать с ним новый блок, который вы только что записали.

Отсоединение старого блока и связывание нового осуществляется за одну операцию, поэтому её нельзя прервать — если вы сбрасываете питание после того, как это произойдёт, у вас есть новая версия файла, а если вы сбрасываете питание раньше, то у вас есть старая версия. В любом случае, в файловой системе не возникнет конфликтов.

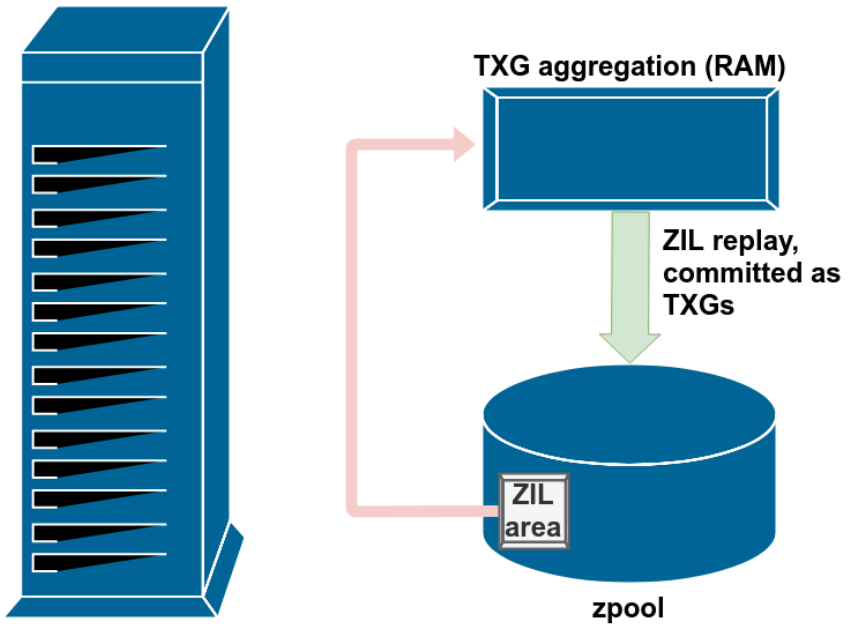
Копирование при записи в ZFS происходит не только на уровне файловой системы, но и на уровне управления дисками. Это означает, что ZFS не подвержена пробелу в записи (дыре в RAID) — феномену, когда полоса успела только частично записаться до сбоя системы, с повреждением массива после перезагрузки. Здесь полоса пишется атомарно, vdev всегда последователен, и Боб — твой дядя.

ZIL: журнал намерений ZFS



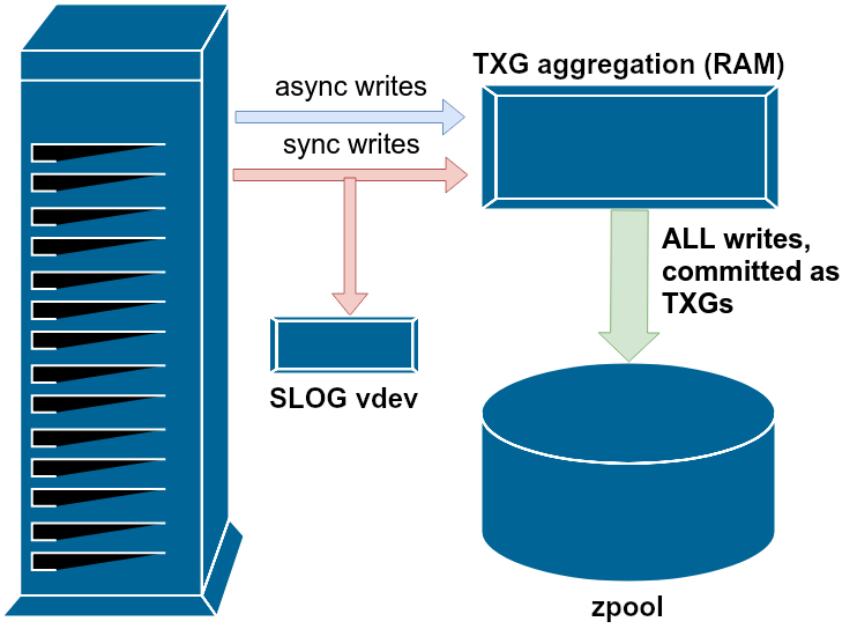
Normal operation - without SLOG

Система ZFS обрабатывает синхронные записи особым образом — она временно, но немедленно сохраняет их в ZIL, прежде чем позже записать их на постоянной основе вместе с асинхронными записями



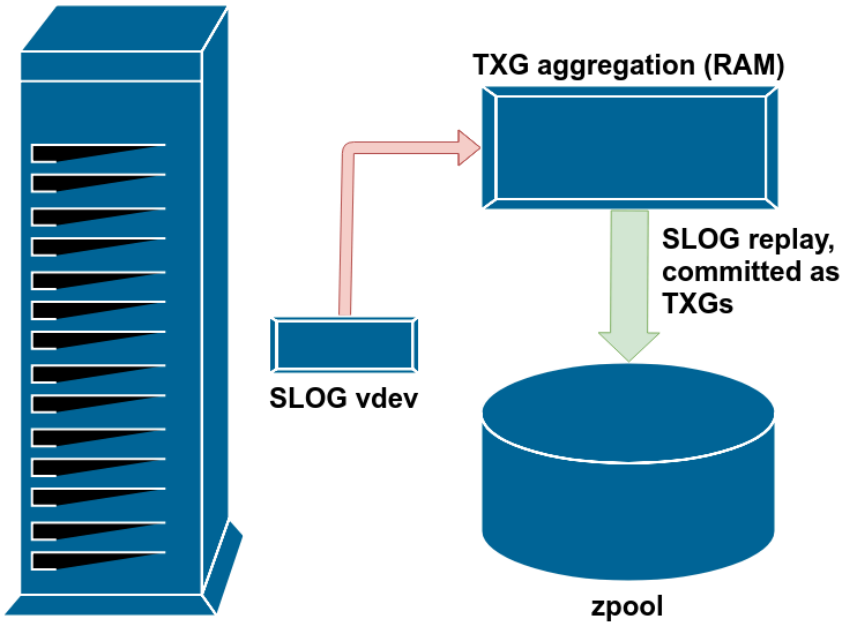
After crash - without SLOG

Обычно данные, записанные на ZIL, больше никогда не считываются. Но это возможно после сбоя системы



Normal operation - with SLOG vdev

SLOG, или вторичное LOG-устройство, — это просто специальный — и, желательно, очень быстрый — vdev, где ZIL может храниться отдельно от основного хранилища



After crash - with SLOG vdev

После сбоя все грязные данные в ZIL воспроизводятся — в данном случае ZIL находится на SLOG, так что они воспроизводятся именно оттуда

Существует две основные категории операций записи — синхронные (sync) и асинхронные (async). Для большинства рабочих нагрузок подавляющее большинство операций записи являются асинхронными — файловая система позволяет агрегировать их и выдавать пакетами, уменьшая фрагментацию и значительно увеличивая пропускную способность.

Синхронные записи — совершенно другое дело. Когда приложение запрашивает синхронную запись, оно говорит файловой системе: «Тебе нужно зафиксировать это в энергонезависимой памяти *прямо сейчас*, а до тех пор я больше ничего не могу сделать». Поэтому синхронные записи должны быть немедленно зафиксированы на диске — и если это увеличивает фрагментацию или уменьшает пропускную способность, так тому и быть.

ZFS обрабатывает синхронные записи иначе, чем обычные файловые системы — вместо того, чтобы немедленно заливать их в обычное хранилище, ZFS фиксирует их в специальной области

хранения, которая называется журнал намерений ZFS — ZFS Intent Log, или ZIL. Хитрость в том, что эти записи *также* остаются в памяти, будучи агрегированными вместе с обычными асинхронными запросами на запись, чтобы позже быть сброшенными в хранилище как совершенно нормальные TXG (группы транзакций, Transaction Groups).

В нормальном режиме работы ZIL записывается и никогда больше не читается. Когда через несколько мгновений записи из ZIL фиксируются в основном хранилище в обычных TXG из оперативной памяти, они отсоединяются от ZIL. Единственное, когда что-то считывается из ZIL — это при импорте пула.

Если происходит сбой ZFS — сбой операционной системы или отключение питания — когда в ZIL есть данные, эти данные будут считаны во время следующего импорта пула (например, при перезапуске аварийной системы). Всё, что находится в ZIL, будет считано, объединено в группы TXG, зафиксировано в основном хранилище, а затем отсоединено от ZIL в процессе импорта.

Один из вспомогательных классов vdev называется LOG или SLOG, вторичное устройство LOG. У него одна задача — обеспечить пул отдельным и, желательно, гораздо более быстрым, с очень высокой устойчивостью к записи, устройством vdev для хранения ZIL, вместо хранения ZIL на главном хранилище vdev. Сам ZIL ведёт себя одинаково независимо от места хранения, но если у vdev с LOG очень высокая производительность записи, то синхронные записи будут происходить быстрее.

Добавление vdev с LOG в пул никак **не может** улучшить производительность асинхронной записи — даже если вы принудительно выполняете все записи в ZIL с помощью `zfs set sync=always`, они всё равно будут привязаны к основному хранилищу в TXG таким же образом и в том же темпе, что и без журнала. Единственным прямым улучшением производительности является задержка синхронной записи (поскольку бóльшая скорость журнала ускоряет выполнение операций `sync`).

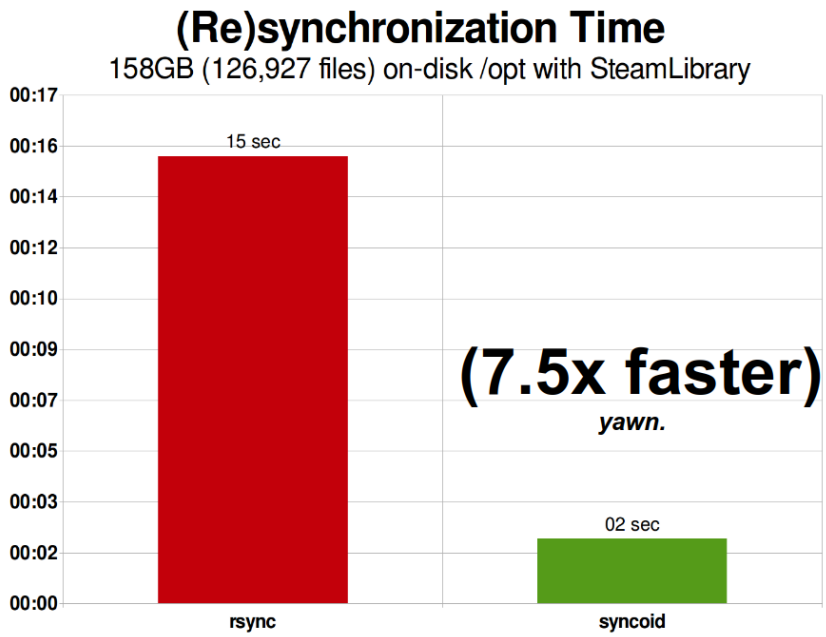
Однако в среде, которая уже требует большого количества синхронных записей, vdev LOG может косвенно ускорить асинхронную запись и некэшированное чтение. Выгрузка записей ZIL в отдельный vdev LOG означает меньшую конкуренцию за IOPS в первичном хранилище, что в некоторой степени повышает производительность всех операций чтения и записи.

Снапшоты

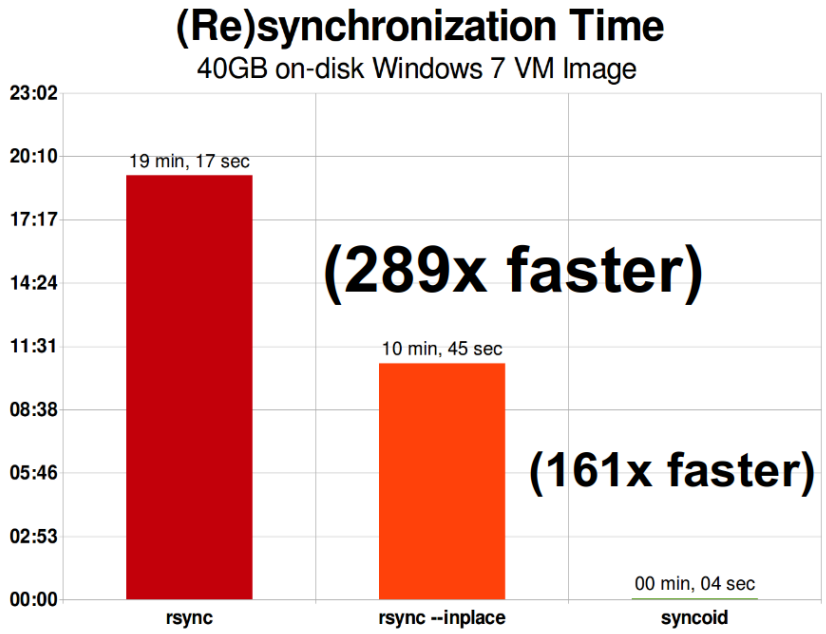
Механизм копирования при записи также является необходимой основой для атомарных моментальных снимков ZFS и инкрементальной асинхронной репликации. В активной файловой системе есть дерево указателей, отмечающее все записи с текущими данными — когда вы делаете снапшот, вы просто делаете копию этого дерева указателей.

Когда в активной файловой системе перезаписывается запись, ZFS сначала записывает новую версию блока в неиспользуемое пространство. Затем отсоединяет старую версию блока от текущей файловой системы. Но если какой-то снапшот ссылается на старый блок, он всё равно остается неизменным. Старый блок фактически не будет восстановлен как свободное пространство, пока все снапшоты, ссылающиеся на этот блок, не будут уничтожены!

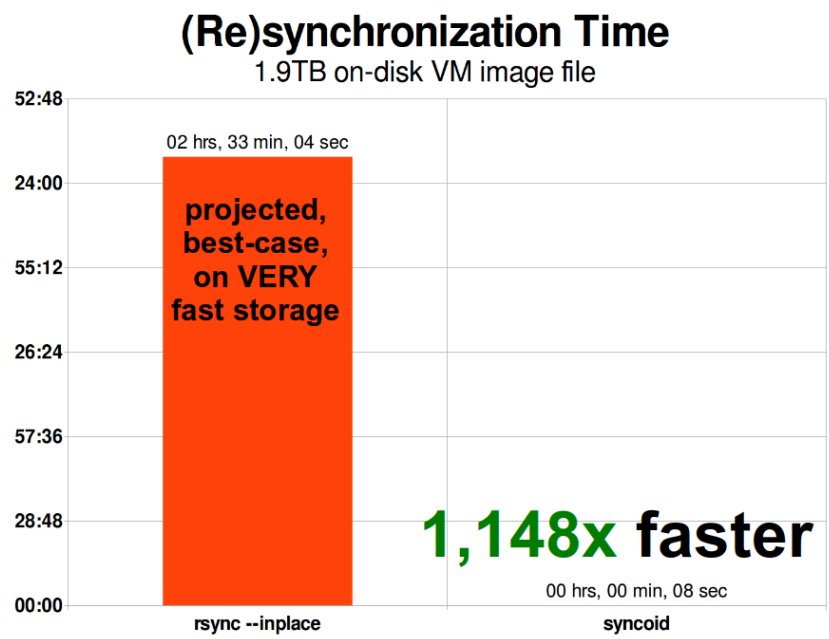
Репликация



Моя библиотека Steam в 2015 году занимала 158 ГиБ и включала 126 927 файлов. Это довольно близко к оптимальной ситуации для rsync — репликация ZFS по сети была «всего лишь» на 750% быстрее.



В той же сети репликация одного 40-гибибайтного файла образа виртуальной машины Windows 7 — совершенно другая история. Репликация ZFS происходит в 289 раз быстрее, чем rsync — или «всего» в 161 раз быстрее, если вы достаточно подкованы, чтобы вызвать rsync с ключом --inplace.



Когда образ виртуальной машины масштабируется, проблемы rsync масштабируются вместе с ним. Размер 1,9 ТиБ не такой большой для современного образа виртуальной машины — но он достаточно велик, чтобы репликация ZFS оказалась в 1148 раз быстрее, чем rsync, даже с аргументом rsync --inplace

Как только вы поймёте, как работают снапшоты, будет несложно уловить суть репликации. Поскольку снапшот — это просто дерево указателей на записи, из этого следует, что если мы делаем `zfs send` снапшота, то мы отправляем и это дерево, и все связанные с ним записи. Когда мы передаём этот `zfs send` в `zfs receive` на целевой объект, он записывает как фактическое содержимое блока, так и дерево указателей, ссылающихся на блоки, в целевой набор данных.

Всё становится ещё интереснее на втором `zfs send`. Теперь у нас две системы, каждая из которых содержит `poolname/datasetname@1`, а вы снимаете новый снапшот `poolname/datasetname@2`. Поэтому в исходном пуле у вас `datasetname@1` и `datasetname@2`, а в целевом пуле пока только первый снапшот `datasetname@1`.

Поскольку между источником и целью у нас есть общий снапшот `datasetname@1`, мы можем сделать *инкрементальный* `zfs send` поверх него. Когда мы говорим системе `zfs send -i poolname/datasetname@1 poolname/datasetname@2`, она сравнивает два дерева указателей. Любые указатели, которые существуют только в `@2`, очевидно, ссылаются на новые блоки — поэтому нам понадобится содержимое этих блоков.

В удалённой системе обработка инкрементального `send` такая же простая. Сначала мы записываем все новые записи, включённые в поток `send`, а затем добавляем указатели на эти блоки. Вуаля, у нас `@2` в новой системе!

Асинхронная инкрементальная репликация ZFS — это огромное улучшение по сравнению с более ранними методами, не основанными на снапшотах, такими как rsync. В обоих случаях передаются только изменённые данные — но rsync должен сначала *прочитать* с диска все данные с обеих сторон, чтобы проверить сумму и сравнить её. В отличие от этого, репликация ZFS не считывает ничего, кроме деревьев указателей — и любых блоков, которые не представлены в общем снапшоте.

Встроенное сжатие

Механизм копирования при записи также упрощает систему встроенного сжатия. В традиционной файловой системы сжатие проблематично — как старая версия, так и новая версия изменённых данных находятся в одном и том же пространстве.

Если рассмотреть фрагмент данных в середине файла, который начинает свою жизнь как мегабайт нулей от 0x00000000 и так далее — его очень легко сжать до одного сектора на диске. Но что произойдёт, если мы заменим этот мегабайт нулей на мегабайт несжимаемых данных, таких как JPEG или псевдослучайный шум? Неожиданно этому мегабайту данных потребуется не один, а 256 секторов по 4 КиБ, а в этом месте на диске зарезервирован только один сектор.

У ZFS нет такой проблемы, так как изменённые записи всегда записываются в неиспользуемое пространство — исходный блок занимает только один сектор 4 КиБ, а новая запись займёт 256, но это не проблема — недавно изменённый фрагмент из «середины» файла был бы записан в неиспользуемое пространство независимо от того, изменился его размер или нет, поэтому для ZFS это вполне штатная ситуация.

Встроенное сжатие ZFS отключено по умолчанию, и система предлагает подключаемые алгоритмы — сейчас среди них LZ4, gzip (1-9), LZJB и ZLE.

- **LZ4** — это потоковый алгоритм, предлагающий чрезвычайно быстрое сжатие и декомпрессию и выигрыш в производительности для большинства случаев использования — даже на довольно медленных CPU.
- **GZIP** — почтенный алгоритм, который знают и любят все пользователи Unix-систем. Он может быть реализован с уровнями сжатия 1-9, с увеличением степени сжатия и использования CPU по мере приближения к уровню 9. Алгоритм хорошо подходит для всех текстовых (или других чрезвычайно сжимаемых) вариантов использования, но в противном случае часто вызывает проблемы с CPU — используйте его с осторожностью, особенно на более высоких уровнях.
- **LZJB** — оригинальный алгоритм в ZFS. Он устарел и больше не должен использоваться, LZ4 превосходит его по всем показателям.
- **ZLE** — кодировка нулевого уровня, Zero Level Encoding. Она вообще не трогает нормальные данные, но сжимает большие последовательности нулей. Полезно для полностью несжимаемых наборов данных (например, JPEG, MP4 или других уже сжатых форматов), так как он игнорирует несжимаемые данные, но сжимает неиспользуемое пространство в итоговых записях.

Мы рекомендуем сжатие LZ4 практически для всех вариантов использования; штраф за производительность при встрече с несжимаемыми данными очень мал, а *прирост* производительности для типичных данных значителен. Копирование образа виртуальной машины для новой инсталляции операционной системы Windows (свежеустановленная ОС, никаких данных внутри ещё нет) с `compression=lz4` прошло на 27% быстрее, чем с `compression=none`, в этом тесте 2015 года.

ARC — кэш адаптивной замены

ZFS — это единственная известная нам современная файловая система, которая использует собственный механизм кэширования чтения, а не полагается на кэш страниц операционной системы для хранения копий недавно прочитанных блоков в оперативной памяти.

Хотя собственный кэш не лишён своих проблем — ZFS не может реагировать на новые запросы о выделении памяти так же быстро, как ядро, поэтому новый вызов `mallocate()` может потерпеть неудачу, если ему потребуется оперативная память, занятая в настоящее время ARC. Но есть веские причины использовать собственный кэш, по крайней мере сейчас.

Все известные современные ОС, включая MacOS, Windows, Linux и BSD, для реализации кэша страниц используют алгоритм LRU (Least Recently Used). Это примитивный алгоритм, который поднимает кэшированный блок «вверх очереди» после каждого чтения и вытесняет блоки «вниз очереди» по мере необходимости, чтобы добавить новые промахи кэша (блоки, которые должны были быть прочитаны с диска, а не из кэша) вверх.

Обычно алгоритм работает нормально, но в системах с большими рабочими наборами данных LRU легко приводит к трэшингу — вытеснению часто необходимых блоков, чтобы освободить место для блоков, которые никогда больше прочитаются из кэша.

ARC — гораздо менее наивный алгоритм, который можно рассматривать как «взвешенный» кэш. После каждого считывании кэшированного блока он становится немного «тяжелее» и его становится труднее вытеснить — и даже после вытеснения блок *отслеживается* в течение определённого периода времени. Блок, который был вытеснен, но затем должен быть считан обратно в кэш, также станет «тяжелее».

Конечным результатом всего этого является кэш с гораздо большим коэффициентом попадания (hit ratio) — соотношением между попаданиями в кэш (чтение, выполняемое из кэша) и промахами (чтение с диска). Это чрезвычайно важная статистика — мало того, что сами хиты кэша обслуживаются на порядки быстрее, промахи кэша также могут обслуживаться быстрее, так как чем больше хитов кэша — тем меньше параллельных запросов к диску и тем меньше задержка для тех оставшихся промахов, которые должны обслуживаться с диска.

Заключение

После изучения основной семантики ZFS — как работает копирование при записи, а также отношений между пулами хранения, виртуальными устройствами, блоками, секторами и файлами, — мы готовы обсудить реальную производительность с реальными числами.

В следующей части мы рассмотрим фактическую производительность пулов с зеркальными vdev и RAIDz, друг по сравнению с другом, а также по сравнению с традиционными RAID-топологиями ядра Linux, которые мы исследовали ранее.

Сначала мы хотели рассмотреть только основы — сами топологии ZFS — но после *такого* будем готовы говорить о более продвинутой настройке и тюнинге ZFS, включая использование вспомогательных типов vdev, таких как L2ARC, SLOG и Special Allocation.

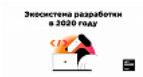
Источник: https://habr.com/ru/post/504692/

Вернуться к списку (/info/articles/)

Интересные статьи

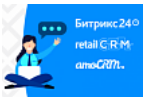
(/info/articles/mg
dri-dlya-

МГЛ (ДРИ) для освещения в квартире или рабочем месте, основы и выбор + обзор ламп на 35Вт (/info/articles/mgl-dri-dlya-osveshcheniya-v-kvartire-ili-rabochem-meste-osnovy-i-vybor-obzor-lamp-na-35vt/)
Хочу вам рассказать о «самой слабой», ну почти самой слабой лампе МГЛ, доступной в общей продаже. Лампы лучше выбирать «стандартной мощности» они (почти)всегда есть в сво...



(/info/articles/ek
razrabotki-v-
2020-godu-

Экосистема разработки в 2020 году: чем живут программисты в России и мире (/info/articles/ekosistema-razrabotki-v-2020-godu-chem-zhivut-programmisty-v-rossii-i-mire/)
Привет, Хабр! В начале 2020 года мы провели уже четвертый ежегодный опрос о состоянии экосистемы разработки, чтобы выяснить, чем живут программисты, какие языки, технологии и инструме...



Обзор CRM систем: amoCRM, retailCRM, Битрикс24 (/info/articles/obzor-crm-sistem-amocrm-retailcrm-bitrix24/)
Сравнивать CRM системы – дело неблагодарное. Очень уж сильно они отличаются в целях создания, реализации, в деталях.

(/info/articles/inv
na-birzhe-kak-
sposob-

Инвестиции на бирже как способ сохранения финансов: 3 работающих способа (/info/articles/investitsii-na-birzhe-kak-sposob-sokhraneniya-finansov-3-rabotayushchikh-sposoba/)
Согласно статистике, на июль 2019 года наивысшая ставка по банковским депозитам в России не превышает 9,5% годовых, что не очень много. Других «пассивных» способов сохранения и приумножения д...

(/info/articles/pc
proizvoditelnost-
ssd-c-kolen-

Поднимаем производительность SSD с колен, куда нажимать и зачем (/info/articles/podnimaem-proizvoditelnost-ssd-c-kolen-kuda-nazhimat-i-zachem/)
На самом деле, речь сегодня пойдёт не только о бывших в использование какое-либо время накопителях, ведь проблема низкого быстродействия может затронуть даже только что принесённый из магазина SS...

ШАБЛОНЫ САЙТОВ (/CATALOG/)

МОИ РАБОТЫ (/PROJECTS/)

УСЛУГИ (/SERVICES/)

НОВОСТИ (/INFO/NEWS/)

СТАТЬИ (/INFO/ARTICLES/)

КОНТАКТЫ (/CONTACTS/)

BitrixTM (skype:bitrixtm)

info@temofeev.ru (mailto:info@temofeev.ru)