АВТОМАТИЧЕСКАЯ НАСТРОЙКА ПАРАМЕТРОВ ТИРИНГА, ЗАВИСЯЩАЯ ОТ ВХОДЯЩЕЙ НАГРУЗКИ, В СИСТЕМЕ ХРАНЕНИЯ ДАННЫХ

Смирнов М.А., 4 курс. кафедра системного программирования СПбГУ,

sm_m_a@mail.ru

Маров А.В., разработчик лаборатории RAIDIX

marov.a@raidix.com

Аннотация

В работе рассматривается реализация многоуровневой системы хранения данных - тир. Приводятся результаты функционального тестирования и измерения производительности получившейся системы. Описываются детали реализации системы и реализация алгоритма идентификации “горячих” данных.

Введение

В последнее время, учёными и исследователями разрабатываются и совершенствуются технологии, предназначенные для замены обычных накопителей на базе магнитных дисков (HDD). Флэш-память ярко выделяется на фоне остальных и сейчас уже стала один из наиболее популярных способов хранения информации в мобильных устройствах, персональных компьютерах, корпоративных серверах, благодаря высокой скорости доступа, низком энергопотреблении, высокой ударопрочности и портативности. Появление твердотельных накопителей (SSD), оказало значительное влияние на существующую иерархию в системах хранения данных (СХД). От организации архитектуры такой системы будет зависеть её производительность, затраты памяти, срок службы и множество других

факторов. Можно поместить SSD между основной памятью (RAM) и HDD, и флэш-память будет кэшом второго уровня, только энергонезависимым.

Информацию, с которой работают различные СХД, можно условно разделить на горячую и холодную, основываясь на предположении, будет ли она нам нужна, и как часто будем к ней обращаться в будущем. Первостепенная задача в этой области – умение идентифицировать “горячие” данные. Правильность решения этой проблемы напрямую будет влиять на производительность всей системы. В свою очередь, ускорение системы будет достигаться перемещением “горячих” данных на SSD или на SSD, обладающим поддержкой спецификации NVM Express, а “холодных” на HDD, соответственно. Получившаяся многоуровневая система хранения данных называется тир (от англ. tiered storage - многоуровневое хранение). Уровней иерархии может быть сколько угодно, но как правило используется два или три.

Обзор

Как уже говорилось выше, самым важным аспектом в реализации тира является правильная идентификация “горячих” данных. Два основных фактора, позволяющих отнести информацию к определенному типу – частота доступа и недавность обращения. Все алгоритмы идентификации, так или иначе, опираются на эти факторы.

Алгоритмы идентификации

К рассмотрению были выбраны следующие алгоритмы, представленные ниже.

1. В алгоритме Window-based Direct Address Counting статистика собирается по количеству запросов, что не подходит для тиринга, где статистика должна собираться по времени.
2. Основная суть алгоритма Multiple Hash Function заключается в экономии памяти - статистика не хранится для каждого блока отдельно. Но возможны ошибки идентификации, когда алгоритм принимает “холодные” за “горячие”.
3. Алгоритм HotDataTrap экономит память, храня статистику, не по всему ключу блока, а лишь по его части. Также возможны ошибки

идентификации.

Существующие решения

1. IBM Easy Tier мониторит ввод-вывод и задержки при обращении к данным за период 24 часа. На основе анализа промониторенных данных формируется план миграции, после чего данные начинают перемещаться между уровнями динамически в зависимости от нагрузки. Тир от IBM поддерживает до трёх уровней. Размер блока данных настраивается в диапазоне от 16 МБ до 8182 МБ.
2. EMC FAST VP позволяет настраивать продолжительность и периодичность миграции. Например, можно каждый день начинать миграцию в 23:00, когда уже никто не работает с сервером, и закончить в 7:00, когда пользователи уже начинают работу. Размер блока данных равен 256 МБ.

Алгоритм идентификации

Выбранный для реализации алгоритм Multiple Hash Function базируется на понятии Bloom Filter. Для работы Bloom Filter требуется битовый массив и произвольное количество хэш-функций, может быть даже одна. (см. Рис. 1) Изначально битовый массив заполнен нулями. На вход хэш-функций подаётся значение (на практике LBA), и индекс в битовом массиве, соответствующий значению хэш-функции становится равным единице.

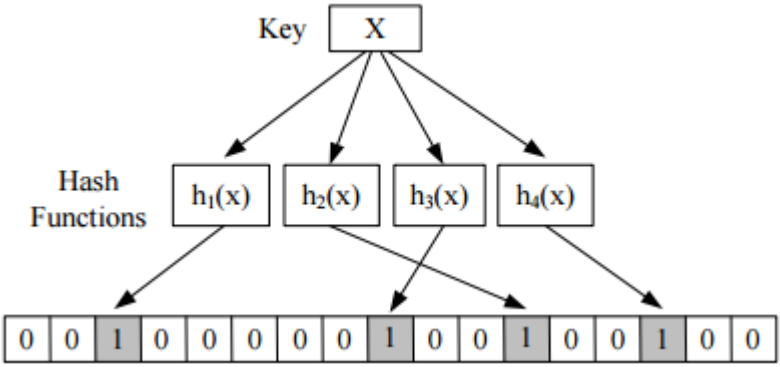

Рисунок 1: Bloom Filter

В алгоритме Multiple Hash Function роль битового массива выполняет массив значений, когда приходит запрос значение в массиве увеличивается на единицу. Для определения “горячих” данных нам необходимо определить некоторое условие, выполнение которого будет говорить об этом. Например, если значения в массиве, соответствующие значениям хэш-функций, больше некоторого числа, то данные “горячие”, и “холодные”, если меньше. Устаревание данных реализуется делением значений в массиве на выбранную или настраиваемую константу. Для проверки является ли определённая ячейка данных “горячей” необходимо подставить её LBA в хэш-функции и проверить выполняется ли условие.

Реализация

Для работы с тиром требуется создать на сервере том, который и будет исходным тиром. Для этого создаются RAID массивы для каждого уровня тира. Например, для трёхуровневого можно создать RAID с HDD дисками, с SSD дисками и с SSD дисками, обладающим поддержкой спецификации NVM Express. Код тира реализован на C в виде модуля ядра Linux.

Для реализации первой версии рабочего тира было решено упростить часть, связанную с идентификацией. Для простоты каждой ячейке памяти соответствует значение в массиве, величина которого равна количеству обращений к этой ячейке. Когда необходимо начать переносить информацию между уровнями тира массивы “холодных” и “горячих” данных сортировались по частоте доступа. “Холодные” – по убыванию, “горячие” – по возрастанию. Ячейки, с наибольшей частотой доступа среди “холодных”, меняются местами с ячейками, с наименьшей частотой среди “горячих”. Если пользователь при создании тира не соотносит все RAID-ы на сервере с уровнями, то оставшиеся автоматически определяются под “холодные” данные. После реализации трёхуровневого тира, в существующую систему был встроен алгоритм Multiple Hash Function. Теперь мы можем определить “горячие” данные и усовершенствовать систему. Во-первых, теперь нет необходимости хранить статистику для

каждой ячейки отдельно, что экономит память. Во-вторых, перед миграцией в массивы не добавляются данные, соответствующие своему уровню. Если “горячие” данные уже лежат на верхнем уровне, то нет смысла их переносить. Возможно есть более “горячие” данные, но миграция затратный процесс. Перед миграцией смотрим процент “горячих” данных, если он слишком большой (например, >40%), то повышаем уровень "горячести" данных, если наоборот (<10%), то уменьшаем. Проценты определяются объёмом, который занимают уровни относительно друг друга. После миграции "устареваем" данные, при этом те данные, к которым не было обращения с момента последней миграции, устаревают сильнее.

Функциональное тестирование и измерение производительности

Для тестирования использовалась утилита fio, позволяющая создавать сложные сценарии тестирования, имеющая несколько режимов нагрузки (случайное чтение, случайная запись и их комбинация). В процессе и конце работы fio показывает количество операций в секунду и задержку выполнения операций. Fio позволяет игнорировать файловую систему и обращаться напрямую к диску. Также для тестирования использовалась утилита dd, для отправки единовременных запросов к дискам, и команда iostat для проверки процентной загруженности дисков в системе.

Проверка корректности

Для проверки корректности работы алгоритма в течение часа нагружается область размером 100 ГБ на HDD-RAID и ожидается, что все данные переедут на SSD-RAID. Миграция данных происходит раз в минуту.

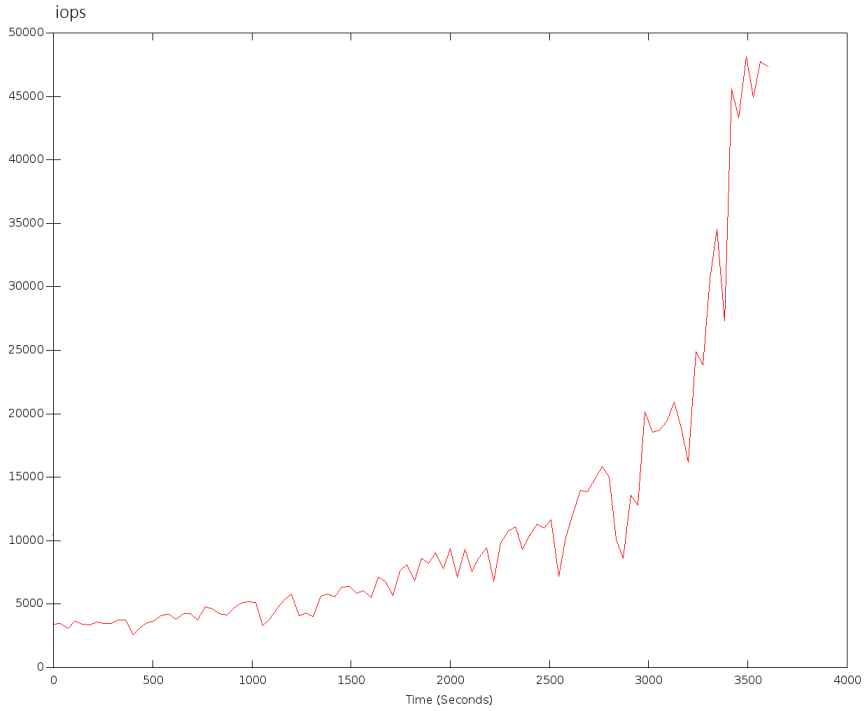

Рисунок 2: Показатели iops в тесте на корректность работы

Скорость работы сильно возросла под конец теста (см. Рис. 2), при этом нагрузка на HDD диски была равна нулю, а на SSD диски выше 99% .

Тестирование размера Bloom Filter

В рамках данного теста нагружалась область размером 100 ГБ в течение 1 часа, но при каждом запуске теста менялся размер Bloom Filter. Количество ячеек данных равно 8136, а размер менялся от 4000 до 16000. Миграция происходила раз в минуту.

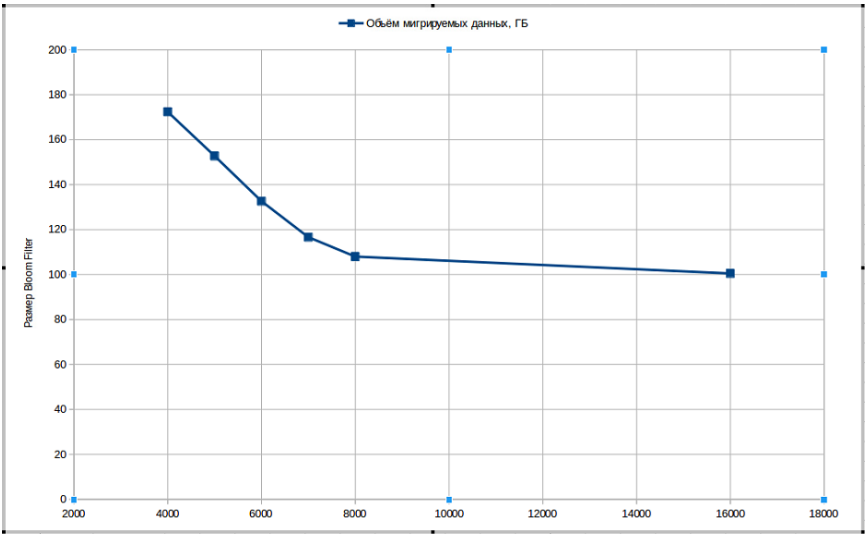

Рисунок 3: Объём мигрируемых данных, в зависимости от размера Bloom Filter

Для Bloom Filter размером 4000 было перенесено 173 ГБ данных (см. Рис. 3), для фильтра размером 16000 перенесли 100,5 ГБ информации. Для успешной работы тира приемлемыми являются размеры от 7300, где экономия памяти на статистических данных равна 10%, и погрешность переноса примерно равна 10%.

Тестирование количества хэш-функций

Чтобы понять какое количество хэш-функций использовать была проведена серия тестов, в которых нагружалась область объёмом 100 ГБ и смотрели количество данных, которое было перенесено. Общее количество ячеек данных равно 8136, размер Bloom Filter был выбран равным 8000. В случае двух хэш-функций было мигрировано 140 ГБ данных, в случае трёх 115 ГБ. Что помимо уменьшения количества бесполезной работы дало сравнительно быстрый рост производительности (~2000 сек против ~1300 во втором случае, т.е. 11,5 мин выигрыша). Затем было принято решение протестировать для четырёх хэш-функций (см. Рис. 4).

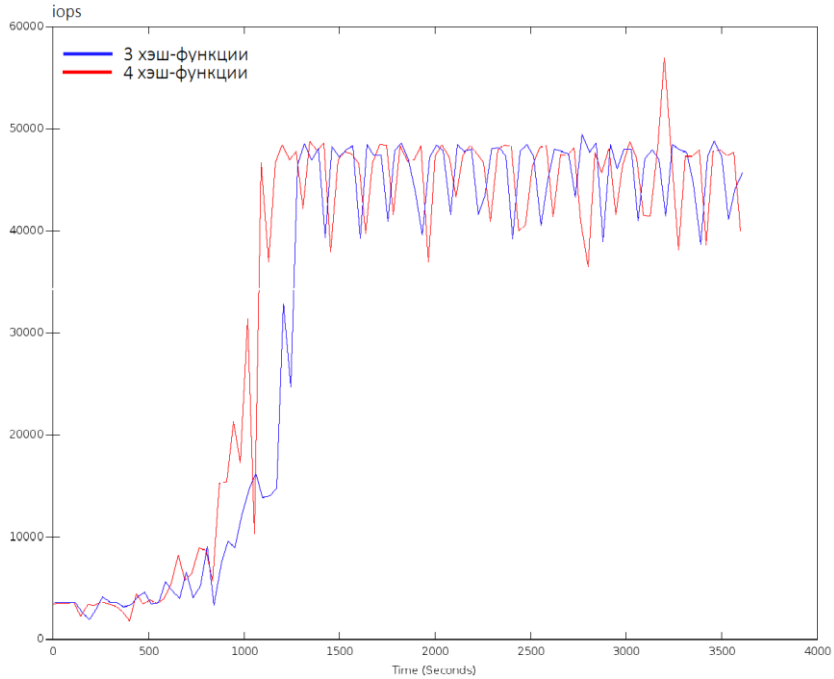

Рисунок 4: Показатели iops для тестов с различным количеством хэш-функций

Для четырёх хэш функций было перенесено 108 ГБ против 115 ГБ у трёх. Рост производительности также был быстрее (~1200 сек против ~1300).

Сравнение с тиром без миграции

Нагружаем весь тир случайным чтением. И, по очереди, разными потоками нагружаем области HDD-RAID, следующие друг за другом. Тот же самый тест запустим позже, но отключим миграцию данных (это будет, по сути, обычная система)

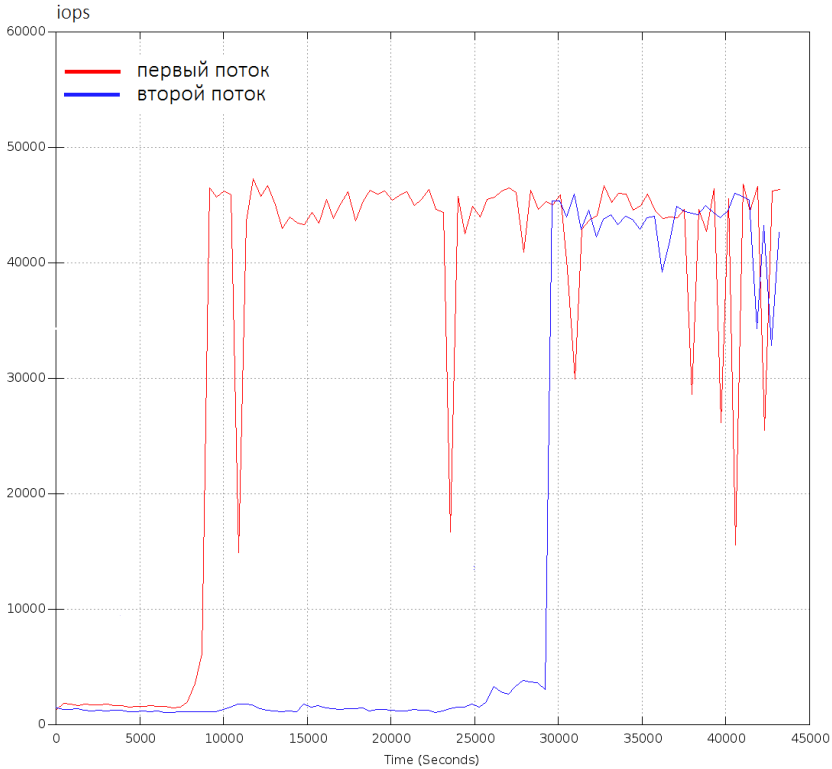

Рисунок 5: Показатели iops для потоков по двум областям HDD-RAID

Естественно, что первая область переехала быстрее, но, в итоге, они достигли одинаковой производительности. В таблице 1 показано сравнение производительности одних и тех же потоков с миграцией и без.

Параметр сравнения	С миграцией	Без миграции	С миграцией лучше?
Средняя скорость потока по всему тиру, iops	2300	1800	+
Средняя скорость первого потока по области, iops	34000	1800	+
Средняя скорость второго и последующих потоков по области, iops	15000	1800	+
Объём мигрируемых данных, ГБ	360	0	-

Таблица 1: Сравнение показателей системы с миграцией и без

Заключение

В документе был представлен и описан алгоритм идентификации “горячих” данных, описано создание и принцип работы тира. Было проведено тестирование параметров: размера Bloom Filter и количества хэш-функций. Представлено сравнение системы с тиром и без. В результате оказалось, что рост производительности составил 1800% для первого потока и 800% для последующих, а производительность потока по всему тиру увеличилась на 27%, но при этом было перенесено 360 ГБ данных. В результате рост производительности окупает затраты на миграцию.

Литература

1. EMC. Information storage and management: Storing, Managing, and Protecting Digital Information. / Под ред. EMC Education Services G. Somasundaram, Alok Shrivastava. — Wiley Publishing, Inc, 2009.
2. Jen-Wei Hsieh Tei-Wei Kuo, Chang Li-Pin. Efficient Identification of Hot Data for Flash Memory Storage Systems. — ACM Transactions on Storage, 2006.
3. Linux Cross Reference. — URL: http://lxr.free-electrons.com/ [дата просмотра: 10.04.2017].
4. Linux по-русски. Изучаем команды Linux: dd // Виртуальная энциклопедия. — URL: http://rus-linux.net/ [дата просмотра: 11.04.2017].
5. Park Dongchul. Hot and Cold Data Identification: Applications to

Storage Devices and Systems.
6. Олег Цилюрик. Программирование модулей ядра Linux.
7. fio - Linux man page. — URL: https://linux.die.net/man/1/fio [дата просмотра: 10.04.2017].