Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Санкт-Петербургский государственный университет»
Кафедра Системного Программирования

Смирнов Михаил Александрович

Реализация random write cache на основе красно-черных деревьев

Курсовая работа

Научный руководитель:
разработчик RAIDIX Маров А.В.

Санкт-Петербург
2016

Оглавление

Введение

Рост объёмов данных, возросшие требования к надёжности хранения и быстродействию доступа к данным стали причинами возникновения систем хранения данных (СХД). Правильной организацией работы и структуры этой системы можно улучшить её быстродействие.

Целью данной работы была оптимизация работы кэша СХД, с использованием красно-чёрных деревьев. В ходе работы СХД выделяют 2 типа нагрузки: последовательная, когда блоки памяти следуют непосредственно друг за другом, и случайная, когда невозможно предсказать расположение блоков памяти относительно друг друга. Определением типа нагрузки занимаются уже существующие детекторы. Последовательная запись эффективна, вследствие механической природы жёсткого диска, потому появилась задача о преобразовании случайной записи в последовательную.

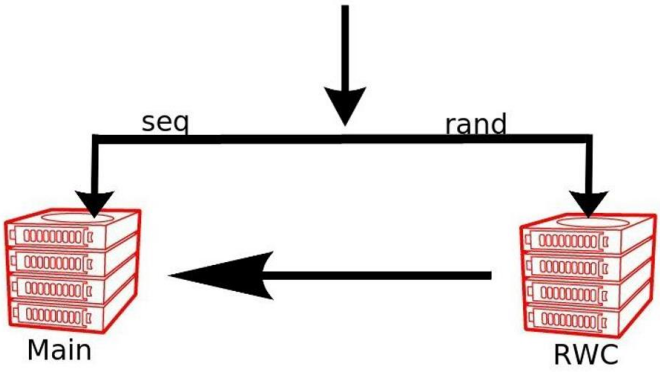

Рисунок 1: Организация работы с последовательными и случайными данными.

На рис 1. видно, что последовательные данные записываются на основной том (Main), а случайные записываем на вспомогательный (Random Write Cache), с сохранением их

исходной позиции на основном томе. Запись продолжается в таком ключе до момента заполнения RWC, либо пока идёт сильная нагрузка. Когда нагрузка ослабевает или вспомогательный том заполнен, данные вытесняются из RWC на Main. В качестве вспомогательного тома может выступать отдельный диск или массив из дисков, объединённых в RAID и рассматривается как блочное устройство.

Постановка задачи.

В рамках данной курсовой работы ставились следующие задачи:

1. Реализовать модуль на языке Си, работающий в пространстве ядра Linux, осуществляющий работу с красно-черными деревьями.
2. Провести сравнение с уже существующим аналогом.

Существующее решение

На данный момент вспомогательный том реализован по типу хэш-таблицы, в которой коллизии разрешаются методом цепочек.

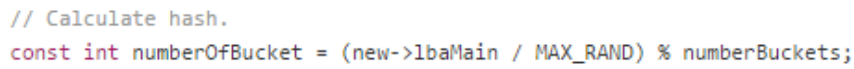

Рисунок 2: хэш-функция.

На рисунке 2 показано вычисление хэша для нового элемента. Для расчёта важны 2 константы: MAX_RAND – максимальный размер случайных данных и numberBuckets – количество ячеек в хэш-таблице. Из RWC на базе хэш-таблицы вытеснение на основной том происходит не последовательно, что противоречит основной идеи работы. Вследствие было принято решение о построении RWC на базе красно-чёрного дерева, что даст нам последовательную запись на всех этапах работы и ускорение всей системы в целом.

Описание алгоритма

Для полноценной работы красно-чёрного дерева понадобятся следующие методы: добавления, вытеснения и корректировки значений.

Добавление элемента в RBTree

Ключом при добавлении в дерево будет адрес блока памяти на основном томе, в процессе могут возникнуть четыре ситуации пересечения данных:

1. Значение нового элемента дерева пересекается с существующим слева.

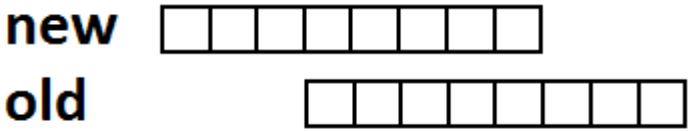

Рисунок 3: Добавление в дерево нового блока (1 случай)

Реализация: Удаляем неактуальные данные у старого элемента и продолжаем добавлять в RBTRee.

2. Значение нового элемента дерева пересекается с существующим справа.

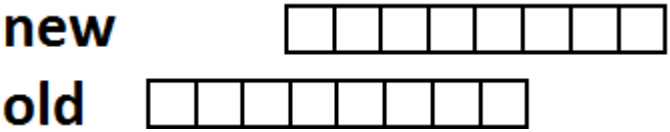

Рисунок 4: Добавление в дерево нового блока (2 случай)

Реализация: Удаляем неактуальные данные у старого элемента и продолжаем добавлять в RBTRee.

3. Значение нового элемента дерева включается в существующее.

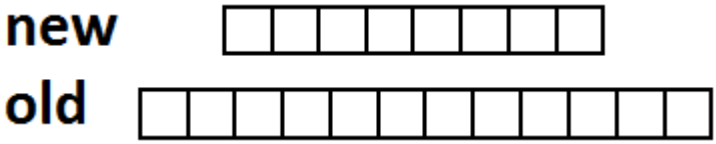

Рисунок 5: Добавление в дерево нового блока (3 случай)

Реализация: Разделяем старый элемент на два, исключая неактуальные данные. Левую часть оставляем на прежней позиции, новый элемент и правую часть продолжаем добавлять в RBTRee.

4. Значение старого элемента дерева включается в новый.

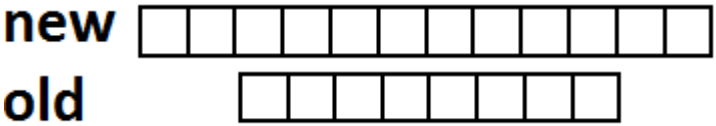

Рисунок 6: Добавление в дерево нового блока (4 случай)

Реализация: Удаляем старый элемент дерева. Новый элемент ставим на данную позицию и корректируем данные в RBTRee.

Вытеснение элемента из RBTree

В данном подходе использовался существующий метод получения первого по ключу элемента из Red-black Tree. Так как ключом при добавлении была позиция блока на основном томе, то для него вытеснение будет последовательным.

Корректировка значений в RBTree

Корректировка значений используется как при случайной нагрузке, так и при последовательной. Новые данные могут пересекаться с существующими, и, в случае пересечения, необходимо удалить или обновить устаревшую информацию в дереве. Корректировка значений во многом схожа с добавлением. Единственное отличие – отсутствие непосредственного добавления в красно-чёрное дерево.

Реализация и тестирование

Описанные алгоритмы реализованы на C в виде модуля ядра Linux, чтобы максимально приблизить полученные замеры к реальному производственному решению. В данной работе использовалась уже существующая структура red-black tree.

В качестве входных данных использовался лог запросов:

1) Генерируемых случайным и последовательным потоком.
2) Только случайным потоком.

В запросе указывается: его время, дата, номер, тип команды (Write или Read), последовательная запись или случайная, адрес на основном томе, длина.

Тестирование скорости проводилось на ноутбуке с процессором Intel® Core™ i3 2,27Ггц и с 4 Гб оперативной памяти. Операционная система Ubuntu. Отклонение в измерениях в среднем составляет 5-10%.

Для измерения времени использовались функции ядра Linux, позволяющие измерить время с точностью до наносекунд.

Сравнение

Для сравнения эффективности работы красно-чёрного дерева и существующей хэш-таблицы был проанализирован набор из 800 000 записей и взята хэш-таблица, имеющая 200 000 ячеек, чтобы количество коллизий было невысоким. Сравнение с существующим решением проводилось по следующим метрикам: скорость добавления, скорость вытеснения данных, скорость корректировки значений и общее время работы.

Для улучшения восприятия информации суммировалось время ста добавлений и десяти вытеснений.

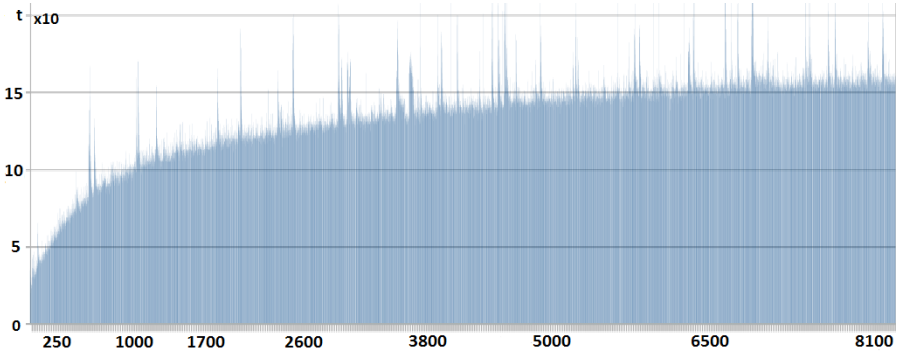

Рисунок 7: Добавление в RBTRee.

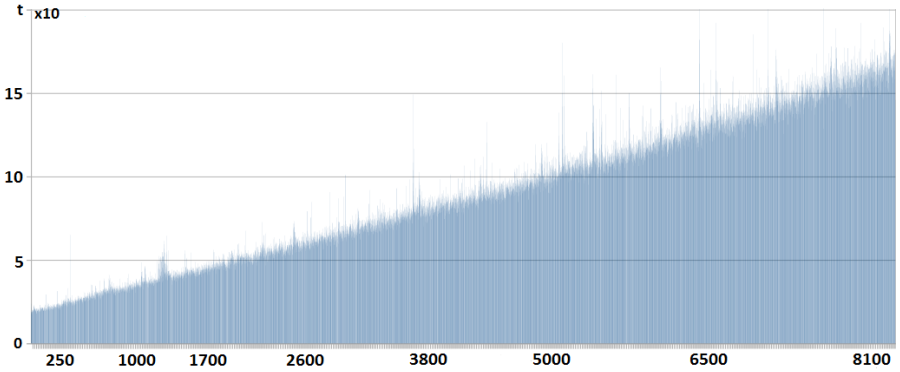

Рисунок 8: Добавление в хэш-таблицу.

Как видно на рисунках 6 и 7, на добавление ста элементов в

красно-чёрное дерево затрачивается 50 000 наносекунд уже примерно при наличии 25 000 элементов, в то время как хэш-таблица тоже время начинает тратить на добавление только при наличии примерно 200 000 элементов. Но время добавления в хэш-таблицу возрастает линейно, а в RBTree по логарифму и под конец хэш-таблице необходимо примерно 170 000 наносекунд на добавление ста элементов, а красно-чёрному дереву чуть больше 150 000 и разница будет увеличиваться.

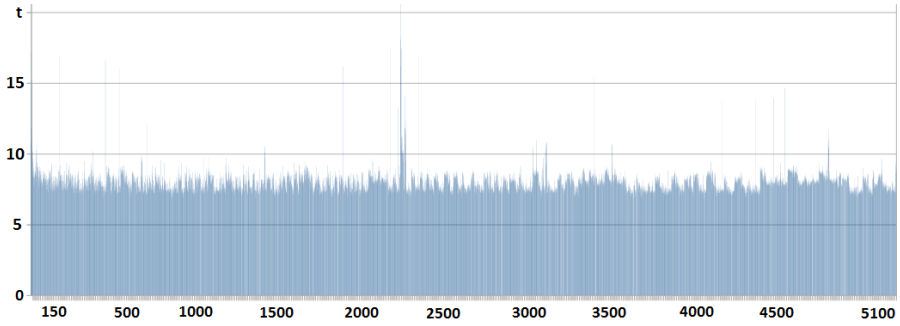

Рисунок 9: Вытеснение из RBTree.

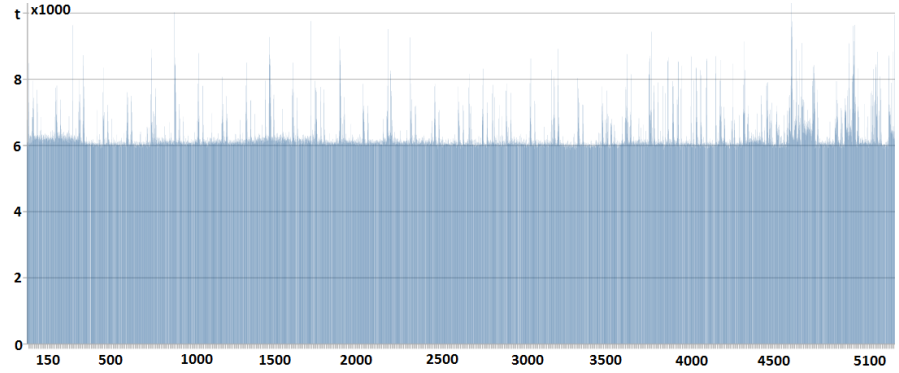

Рисунок 10: Вытеснение из хэш-таблицы.

Из рисунков 8 и 9 следует, что вытеснение из красно-чёрного дерева происходит примерно в 600 раз быстрее. Это вызвано необходимостью сортировать данные в хэш-таблице перед вытеснением.

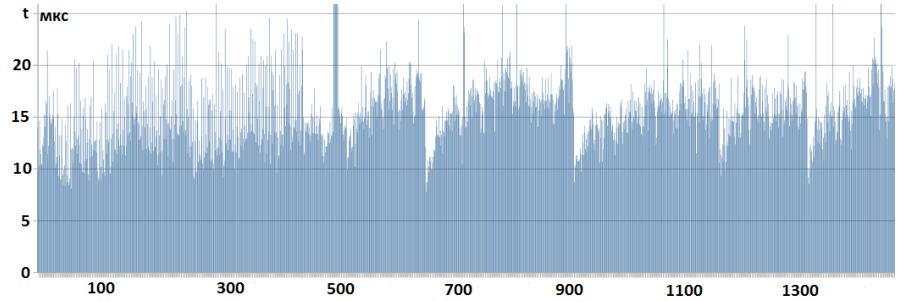

Рисунок 11: Корректировка значений в RBTree.

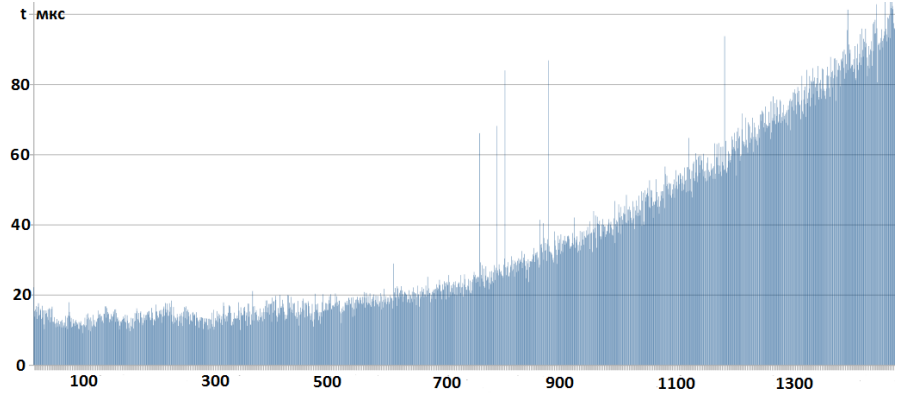

Рисунок 12: Корректировка значений в хэш-таблице.

На рисунках 11 и 12 видно, что для исходных данных скорость корректировки значений в красно-чёрном дереве остаётся практически неизменной, в то время как для хэш-таблицы наблюдается линейный рост. И, в итоге, разница скорости достигает примерно 5 раз.

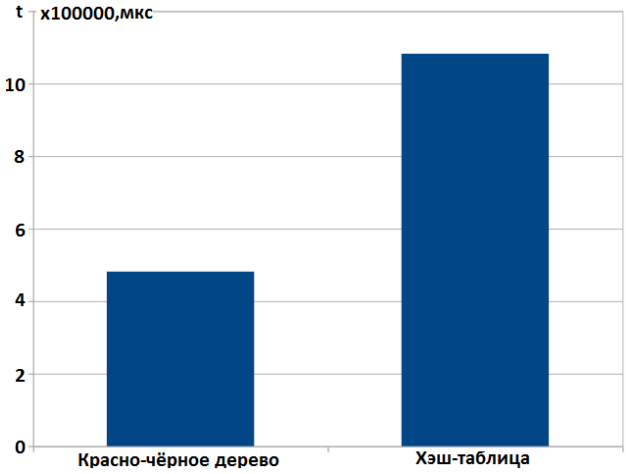

Рисунок 13: Сравнение общего времени работы

Для сравнения общего времени работы были взяты данные, генерируемые случайным потоком. Суммировалась скорость добавления элемента, вытеснение объёма данных и корректировки значений. В итоге, красно-чёрное дерево закончило работу с исходными данными примерно в 2,5 раза быстрее, чем хэш-таблица.

Заключение

В документе были представлены два варианта реализации вспомогательного тома для случайных записей: на основе хэш-таблицы и красно-чёрного дерева. Было проведено сравнение по следующим метрикам: скорость добавления, скорость вытеснения определённого количества элементов, скорость корректировки значений. В результате оказалось, что для RWC основанного на RBTree, для исходных данных, скорость вытеснения примерно в 600 раз быстрее, чем в хэш-таблице, скорость добавления выше, когда количество элементов становится достаточно большим, разница в скорости корректировки, в итоге, различается примерно в 5 раз в пользу красно-чёрного дерева и общее время работы различается в 2,5 раза в пользу RWC, основанного на красно-чёрном дереве.

Список литературы

[1] Цилюрик Олег. Программирование модулей ядра Linux.

[2] Linux Cross Reference. URL: http://lxr.free-electrons.com/ (дата обращения: 21.04.2016)

[3] Т. Кормен, Ч. Лейзер. Алгоритмы. Построение и анализ.

[4] Ning Liu, Jason Cope, Philip Carns, Christopher Carothers, Robert Ross, Gary Grider, Adam Crume, Carlos Maltzahn. On the Role of Burst Buffers in Leadership-Class Storage Systems. URL: http://www.mcs.anl.gov/papers/P2070-0312.pdf (дата обращения: 21.04.2016)

[5] Information storage and management: Storing, Managing, and Protecting Digital Information.