Московский Государственный Университет им. М.В. Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Автоматизации Систем Вычислительных Комплексов

Шпилевой Владислав Дмитриевич

Разработка и реализация алгоритма отложенного обновления вторичных индексов на LSM-деревьях

Магистерская диссертация

Научный руководитель:
к.ф.-м.н. ассистент
Д.Ю.Волканов
Научный консультант:
К.А.Осипов

Москва, 2018

Аннотация

В настоящее время растет популярность баз данных, хранящих данные на диске не в виде традиционных B-деревьев и их производных, а в виде LSM деревьев. Главное преимущество LSM деревьев в том, что их обновление всегда приводит только к последовательной записи на диск, в отличие от B-деревьев. Это возможно благодаря тому, что LSM дерево способно хранить множество версий одной и той же записи - за счет этого при обновлении дерева не нужно точечно читать и удалять старые данные - это происходит позже во время слияния уровней LSM дерева. Это работает, когда в таблице только один индекс - первичный. При наличии вторичных индексов LSM деревья лишаются преимуществ версионности, так как при обновлении дерева нужно явно читать и удалять старые данные из всех вторичных индексов. В настоящей работе представлен обзор существующих способов решения этой проблемы, а также разработанная модификация LSM дерева, которая позволяет не делать явных чтений и удалений старых данных из вторичных индексов. Проведенное экспериментальное исследование нового LSM дерева показало прирост скорости на порядок при наличии нескольких вторичных индексов.

Содержание

1 Введение

1.1 Цель работы

Целью работы является увеличение скорости обновления базы данных, которая хранит индексы таблиц в LSM деревьях при помощи модификации процедур обновления и слияния уровней LSM дерева.

1.2 Актуальность

LSM (Log-Structured Merge) деревья были разработаны в 1990-х [1] годах для задач с интенсивной записью, и использовались в файловых системах и для резервного копирования. Но их применение в СУБД (Система Управления Базой Данных) было ограничено из-за *скрытых чтений*. Скрытыми называют чтения, которые выполняются СУБД при обновлении данных, чтобы либо найти старые, данные и удалить их, либо чтобы проверить ограничения уникальности при вставке новых данных. Значительная часть чтений в СУБД - скрытая, так как почти любое обновление данных требует проверки различных огрaчений и удаления старых данных, и это почти ничего не стоит в B-деревьях, где для обновления данных в любом случае нужно читать [2]. Но на LSM деревьях скрытые чтения значительно снижают производительность, поскольку лишают их преимуществ версонности данных, когда можно сохранять новые данные не читая и не удаляя старые явно.

С появлением SSD (Solid-State Drive) дисков абсолютная скорость любых чтений и записи возросла на порядки по сравнению со старыми механическими дисками, но существенно увеличился разрыв между скоростями последовательной записи и последовательного чтения [3], буквально на порядки. Благодаря тому, что LSM дерево всегда выполняет запись на диск последовательно, а чтения из него на SSD по скорости мало отличаются от B-дерева, LSM дерево стало одной из стандартных структур данных для СУБД. Например, на момент написания работы LSM деревья уже используются в LevelDB [4], RocksDB [5], Cassandra [6], Tarantool [7], BigTable [6], HBase [6], Riak [8], MySQL [9].

Однако SSD хоть и делает LSM дерево более конкурентноспособным, но не решает проблему существования скрытых чтений на любое обновление при наличии вторичных индексов, что не позволяет использовать все возможности LSM деревьев, когда в таблице в БД (База Данных) больше одного индекса.

Когда в таблице только один индекс на LSM дереве, то любые изменения, не требующие знания старых данных (такие как *REPLACE, DELETE*), возможны без скрытых чтений. Например, в случае *REPLACE* запись просто попадает в дерево с новой версией. Тоже самое при *DELETE* - ключ (набор индексируемых колонок и их значений), по которому производится удаление, попадает в дерево с новой версией и пометкой, что это именно удаление, а не вставка. Скрытых чтений не выполняется. Но при появлении вторичного индекса даже *REPLACE* и *DELETE* вынуждены читать старые данные

из первичного индекса, чтобы узнать, какой у старой записи был вторичный ключ, и вставить его *DELETE* в LSM дерево вторичного индекса.

Это обычная процедура для индексов на B-деревьях, где нет версионности данных, и на классических LSM деревьях ее тоже нельзя избежать. Это происходит из-за того, что удаление старых версий данных в LSM дереве работает так, что записи считаются разными версиями одних и тех же данных, только если они равны по ключу, по которому сортируется дерево. И если некоторый запрос меняет этот ключ в уже существующей записи, не читая и не удаляя ее явно, то новая запись становится никак не связанной со старой, и старая не удалится никогда - LSM дерево видит их как разные ключи.

Таким образом, задача борьбы со скрытыми чтениями в таблицах с индексами на LSM деревьях становится актуальной.

1.3 Постановка задачи

Etiam cursus massa quis condimentum suscipit:

1. Vivamus justo dui, scelerisque non blandit a;
2. Molestie non urna;
3. Pellentesque habitant morbi tristique senectus.

1.4 Подход к решению задачи

Duis tempus diam erat. Donec sit amet nulla volutpat, ullamcorper risus et, fermentum ipsum. Suspendisse non tempor quam. Pellentesque in congue felis, non auctor lacus. Aliquam et metus non turpis placerat cursus. Aenean ac felis a orci porta euismod ut ac enim.

1.5 Структура работы

2 Секция

2.1 Подсекция 1

2.2 Подсекция 2

2.3 Подсекция 3

Список литературы

[1] O'Neil P. et al. The log-structured merge-tree (LSM-tree) //Acta Informatica. – 1996. – T. 33. – No. 4. – C. 351-385.

[2] Comer D. Ubiquitous B-tree //ACM Computing Surveys (CSUR). – 1979. – T. 11. – №. 2. – C. 121-137.

[3] Agrawal N. et al. Design Tradeoffs for SSD Performance //USENIX Annual Technical Conference. – 2008. – T. 8. – C. 57-70.

[4] Wang P. et al. An efficient design and implementation of LSM-tree based key- value store on open-channel SSD //Proceedings of the Ninth European Conference on Computer Systems. – ACM, 2014. – C. 16.

[5] Yang F. et al. Optimizing NoSQL DB on flash: A case study of RocksDB //Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC- ScalCom), 2015 IEEE 12th Intl Conf on. – IEEE, 2015. – C. 1062-1069.

[6] Wang P. et al. An efficient design and implementation of LSM-tree based key- value store on open-channel SSD //Proceedings of the Ninth European Conference on Computer Systems. – ACM, 2014. – C. 16.й

[7] Сайт базы данных Tarantool [Электронный ресурс] : сайт содержит документацию всех выпущенных версий Tarantool. — Режим доступа: https://tarantool.io. - Загл. с экрана.

[8] Lersch L. et al. An analysis of LSM caching in NVRAM //Proceedings of the 13th International Workshop on Data Management on New Hardware. – ACM, 2017. – C. 9.

[9] Dong S. et al. Optimizing Space Amplification in RocksDB //CIDR. – 2017.