Санкт-Петербургский государственный университет

Математико-механический факультет

Кафедра системного программирования

Реализация уровня изоляции Read Committed для древовидных структур данных

Курсовая работа студента 345 группы

Федотовского Павла Валерьевича

Научный руководитель Чернышев Г.А.,

ассистент кафедры информатики СПбГУ

Санкт-Петербург

2012

Содержание

Введение

Согласно [1], индексом называется структура данных и метод размещения данных в таблице, что индексируется. Данный объект позволяет значительно ускорить время выполнения различных запросов в зависимости от типа индекса. Например, использование B-деревьев позволяет снизить оценку временной сложности поиска элемента в таблице с линейной до логарифмической. Существует два основных подхода к построению индексов [1]: использовать хеш-таблицы или деревья поиска. Данная работа посвящена последнему типу.

Кроме реализации структуры данных и ее интеграции с исходными данными (таблицей), необходимо также обеспечить корректность параллельного доступа для нескольких потоков. Для решения этой задачи применяются различные методы, например для B+ деревьев существует несколько классических алгоритмов [2]. Однако они слабо применимы для случая многомерных индексов и, соответственно, многомерных деревьев поиска. Индекс такого типа строится для набора атрибутов и позволяет осуществлять поиск не только по отдельным значениям, но сразу по кортежам из нескольких значений. Такие индексы широко применяются, например, для построения геоинформационных систем, CAD систем (computer-aided design), мультимедиа СУБД [3].

Корректность при параллельном доступе обеспечивается за счет выбора определенного уровня изоляции. В свою очередь, уровень изоляции может определяться как набор допустимых аномалий транзакционного доступа [2]. Различают четыре классических уровня изоляции: read uncommitted, read committed, repeatable read, serializable (перечислены в порядке усиления требований корректности). Последний из этих уровней обеспечивает полную корректность, однако им не всегда пользуются на практике из-за его “дороговизны” [4]. Вместо него часто используется уровень изоляции read committed, согласно [5] можно получить выигрыш в скорости в 2.5-3 раза.

Одной из популярных моделей обеспечения транзакционного доступа в деревьях для многомерных индексов является GiST (Generalized Search Tree). Данная модель позволяет наладить транзакционный доступ при работе с любым деревом, отвечающим определенным требованиям [6]. Она является достаточно популярным решением и используется, например, в PostgreSQL [11].

В данной работе с помощью построенного прототипа многомерного транзакционного индекса исследуется производительность алгоритма

параллельного доступа с уровнем изолированности read commited. В качестве структуры данных для индекса было использовано R-дерево [3], в индексе отсутствовало журналирование и восстановление, и прототип был ориентирован на нахождение в памяти целиком (in-memory indexing). Замки брались не на страницы, но на отдельные записи.

Данная работа выполнялась в рамках соревнования ACM SIGMOD Programming Contest 2012 [12]. Полный состав команды ‘SPbSU’: Федотовский П.В., Ерохин Г.А., Чередник К.Е.

Общий обзор системы

Данная система многомерного индексирования состоит из нескольких частей:

- Менеджер памяти
- Менеджер блокировок
- Подсистема обработки транзакций
- Древовидная структура данных (в данном случае GiST)

Менеджер памяти необходим для эффективного выделения и освобождения памяти. Практика показывает, что стандартный распределитель не всегда является оптимальным решением, в связи с чем, выгоднее использовать альтернативные решения или написать свой, оптимизированный под нужды конкретной системы. Работа [8] посвящена сравнению различных вариантов распределителей.

Древовидная структура данных служит для непосредственного хранения данных и предоставления доступа к ним. Необходимым требованием к ней являются многопоточный доступ и эффективные алгоритмы поиска и добавления записей. Некоторые детали реализации описываются в работе [9].

Менеджер блокировок и подсистема обработки транзакций обеспечивают поддержку нужного уровня изоляции [10]. Текст данной курсовой работы более подробно рассматривает указанные вопросы и является расширенной версией указанной статьи.

Постановка задачи

Передо мной были поставлены следующие задачи:

- Реализация подсистемы обработки транзакций
- Реализация менеджера блокировок
- Реализация древовидной структуры данных (в данном случае обобщенного дерева поиска)

Предложенный алгоритм

Как известно, для обеспечения уровня изоляции read committed необходимо предотвратить две аномалии: грязное чтение и потерянное обновление [2]. Для этого используется менеджер блокировок и специальный механизм меток, гарантирующий, что транзакция не “увидит” незафиксированных данных.

Основное отличие от алгоритма, описанного в [7], состоит в том, что замки берутся только на сами данные. Каждая запись хранит идентификатор текущей WR транзакции и состояние. Всего состояний пять:

- ProcessInsert – запись вставляется
- ProcessDelete – запись удаляется
- ProcessUpdate – запись обновляется (изменяется только значение, ключ остается прежним)
- Deleted – запись удалена
- Valid – запись в текущий момент не изменяется

Алгоритм удаления - логический, то есть выставляется флаг “запись удалена”. Рассмотрим алгоритмы исполнения основных операций:

- Получить записи (GetRecords)

  Этот метод получает итератор на записи, которые удовлетворяют запросу (в данном случае подходят все записи, содержащиеся в n-мерном прямоугольнике). Сначала происходит поиск необходимых записей с помощью древовидной структуры данных (при этом не учитывается текущее состояние вершины). Затем пользователь последовательно вызывает метод GetNext для получения очередной записи. При этом проверяется ее текущее состояние. Опишем дальнейшие действия в зависимости от состояния:

  - ProcessInsert – запись пропускается, так как транзакция, ее вставляющая, еще не была зафиксирована. Происходит переход к следующей подходящей под запрос записи (если ее нет, то пользователь информируется об этом).
  - ProcessDelete – пользователь получает запись, поскольку изменяющая ее транзакция еще не была зафиксирована или отменена (иначе состояние было бы Deleted или Valid соответственно).

    - ProcessUpdate – возвращается старое значение записи
    - Deleted – запись была удалена, происходит переход к следующей записи (см. пункт ProcessInsert).
    - Valid – пользователь получает запись, так как в текущий момент ее не изменяет ни одна транзакция, и она не удалена.

Ниже приводится краткое описание остальных методов.

- Вставка (Insert)
    - Запись помечается состоянием ProcessInsert и текущим идентификатором транзакции
    - Непосредственная вставка в древовидную структуру данных
- Обновление/удаление (Update/Delete)
    - Нахождение в древовидной структуре нужных записей
    - Получение блокировок на вершины у менеджера блокировок
    - Изменение состояние вершины

Данная работа является работой практического типа, корректность алгоритма подтверждалась unit-тестами, однако ниже кратко объясняется отсутствие аномалий многопоточного доступа.

- Потерянное обновление (Lost Update)

  Возникает в случае, когда две транзакции изменяют одну и ту же запись. Это предотвращается с помощью менеджера блокировок. Каждый раз при изменении записи транзакция запрашивает блокировку, что гарантирует изменение записи только одной транзакцией.
- “Грязное” чтение (Dirty Read)

  Возникает в случае, если транзакция читает данные, которые еще не были зафиксированы. Для борьбы с данной аномалией используется предложенный механизм маркировки записей. Каждый раз при чтении записи анализируется ее состояние. Если происходит изменение, то пользователь получает старую версию записи, которая уже была зафиксирована. Чуть более детально этот процесс описан выше при рассмотрении операции “получить записи”.

О корректной работе с деревом в условиях нескольких потоков

Для корректной работы дерева в условиях нескольких потоков используются:

- Порядковый номер вершины
- Защелки на вершины дерева

Защелка – это особый вид блокировки, отличающийся от нее тем, что она берется на короткое время, и не нужно следить за возникновением тупиков. С точки зрения реализации это блокировка чтения-записи, которая гарантирует то, что одну и ту же вершину в каждый момент времени может изменять только один поток. В то же время, читать данные из вершины несколько потоков могут параллельно.

При переполнении вершины происходит ее разделение: создается новая вершина, и часть записей разделяемой вершины переносится в нее. При этом может возникнуть проблема многопоточного доступа, которую можно описать следующим образом:

Пусть в вершине ‘X’ находятся записи ‘1’ и ‘2’. Поток ‘A’ ищет в дереве запись ‘1’, и в процессе поиска он находит вершину ‘X’ и хочет получить на нее блокировку. В это время поток ‘B’ расщепляет вершину ‘X’ и перемещает запись ‘1’ в новую вершину ‘Y’. Затем поток ‘A’ получает блокировку на вершину ‘X’, но нужная ему запись была перемещена, в связи с чем поток A не найдет запись ‘1’.

Для предотвращения этой проблемы в каждой вершине хранится ее номер, с помощью которого можно определить, разделялась ли вершина с момента последнего просмотра. Также хранится глобальный номер, который относится ко всему дереву в целом. При каждом разделении вершины происходит следующее:

1) Создается новая вершина, в качестве ее порядкового номера используется порядковый номер разделяемой вершины;
2) увеличивается на 1 глобальный номер;
3) порядковый номер разделяемой вершины становится равным глобальному номеру.

Перед получением блокировки на вершину поток запоминает ее порядковый номер. После того, как блокировка получена, происходит сравнение запомненного номера и текущего, что хранится на данный момент в вершине. Если последний номер больше, это означает, что вершина разделялась, и необходимо дополнительно проверить новую вершину. Для того чтобы иметь доступ к вершине, которая была создана при расщеплении, хранится ссылка на нее в вершине, которая разделялась.

Об измерениях

Схема индексов, запросы и распределения были взяты из постановки задачи на соревновании ACM SIGMOD Programming contest 2012 [12]. Части испытательного стенда (генераторы данных и сборщик статистик) были предоставлены организаторами. Эксперименты проводились на сгенерированных данных различных объемов (32, 64 и 128 Мб) размерности 2, 4 и 8, с равномерным распределением. Одновременно исполнялось 8 потоков (по количеству логических процессоров), все измерения проводились на уже построенном индексе. В каждом случае проводилось по 3 испытания и вычислялось среднее значение.

В своих измерениях мы не используем шаблоны, то есть запросы, содержащие весь диапазон по какому-либо измерению.

Конфигурация тестового оборудования представлена в Таблице 1, результаты экспериментов отображает Диаграмма 1.

Процессор	Intel Core i7-2630QM 2.00 GHz
Hyper-Threading	Enabled
ОЗУ	6 Гб
Размер L3 кэша	6 Мб
Размер L2 кэша	256 Кб на ядро
Размер L1 кэша	(32 + 32) Кб на ядро
Операционная система	GNU/Linux, kernel 2.6.38-8-generic

Таблица 1: Конфигурация тестового оборудования

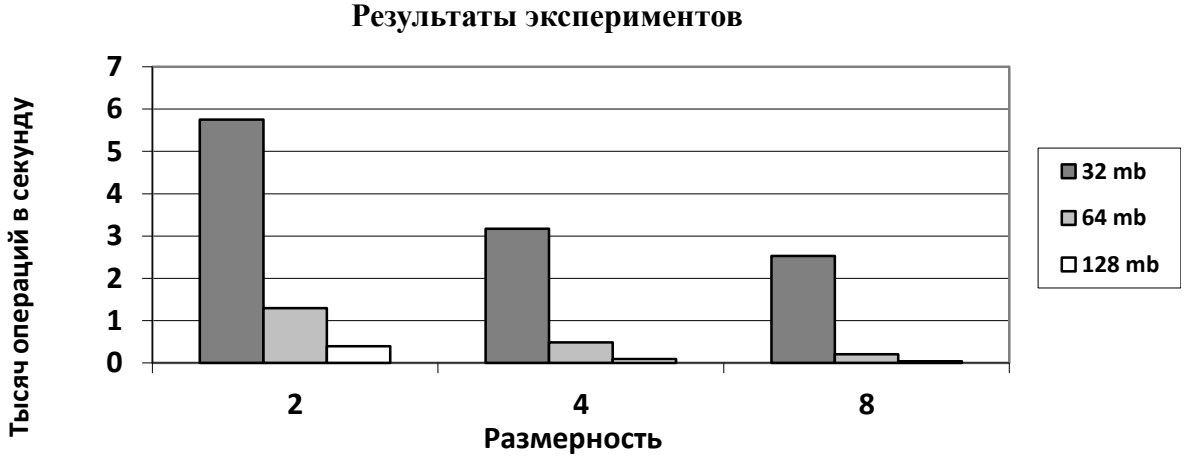

Диаграмма 1: Результаты экспериментов

Можно заметить, что производительность системы сильно падает при увеличении как размера, так и размерности индекса. Также этот график демонстрирует, что текущий прототип справляется с задачей многомерного индексирования, в дальнейшем планируются работы над его улучшением.

Результаты

В данной работе был представлен алгоритм для обеспечения уровня изоляции read committed. С его помощью был построен прототип многомерного транзакционного индекса с данным уровнем изоляции. Прохождение unit-тестов организаторов соревнования ACM SIGMOD Programming Contest 2012 [12] свидетельствует в пользу корректности текущей реализации. Это первая стадия в исследовании, предварительный результат которой – корректный алгоритм и функционирующая реализация. На текущий момент она занимает 5 место по результатам тестирования, основанного на публичных тестах. В дальнейшем планируется повышать производительность системы, а так же представить формальное доказательство корректности применяемого алгоритма. Результаты работы были представлены на конференции СПИСОК [10].

Литература

1. Ling Liu, M. Tamer Özsu (Eds.): Encyclopedia of Database Systems. Index Tuning. 1433-1435. Springer US 2009, ISBN 978-0-387-35544-3, 978-0-387-39940-9.
2. Gerhard Weikum and Gottfried Vossen. 2001. Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
3. Ling Liu, M. Tamer Özsu (Eds.): Encyclopedia of Database Systems. R-Tree (and Family). 2453-2459. Springer US 2009, ISBN 978-0-387-35544-3,978-0-387-39940-9.
4. Ling Liu, M. Tamer Özsu (Eds.): Encyclopedia of Database Systems. SQL Isolation Levels. 2761-2762. Springer US 2009, ISBN 978-0-387-35544-3,978-0-387-39940-9.
5. Paul M. Bober and Michael J. Carey. 1992. On Mixing Queries and Transactions via Multiversion Locking. In Proceedings of the Eighth International Conference on Data Engineering, Forouzan Golshani (Ed.). IEEE Computer Society, Washington, DC, USA, 535-545.
6. Joseph M. Hellerstein, Jeffrey F. Naughton, and Avi Pfeffer. 1995. Generalized Search Trees for Database Systems. In Proceedings of the 21th International Conference on Very Large Data Bases (VLDB '95), Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 562-573.
7. Marcel Kornacker, C. Mohan, and Joseph M. Hellerstein. 1997. Concurrency and recovery in generalized search trees. In Proceedings of the 1997 ACM SIGMOD international conference on Management of data (SIGMOD '97), Joan M. Peckman, Sudha Ram, and Michael Franklin (Eds.). ACM, New York, NY, USA, 62-72.
8. Чередник К.Е, Смирнов К.К. 2012. Динамическое распределение памяти в многопоточном обработчике транзакций. Конференция СПИСОК-2012. (работа принята к публикации).
9. Ерохин Г.А., Чернышев Г.А. 2012. Экспериментальное сравнение алгоритмов разделения вершин в R-дереве на различных данных. Конференция СПИСОК-2012. (работа принята к публикации).

10. Федотовский П.В., Чернышев Г.А., Смирнов К.К. 2012. Реализация уровня изоляции Read Committed для древовидных структур данных. Конференция СПИСОК-2012. (работа принята к публикации).
11. http://www.postgresql.org/docs/8.1/static/gist.html [дата просмотра 23.04.2012]
12. http://wwwdb.inf.tu-dresden.de/sigmod2012contest/#task-overview [дата просмотра 23.04.11]