Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный университет»

Кафедра информационно-аналитических систем

Шкуратов Илья Андреевич

Параллельное пакетное построение R-деревьев

Курсовая работа

Научный руководитель:
д. ф.-м. н., профессор Новиков Б.А.

Санкт-Петербург
2015

SAINT-PETERSBURG STATE UNIVERSITY

Sub-Department of Analytical Information Systems

Ilia Shkuratov

R-trees parallel bulk-loading methods

Course Work

Scientific supervisor:
Doctor of Physics and Mathematics, Professor Boris Novikov

Saint-Petersburg
2015

Оглавление

Введение

Для обеспечения быстрого доступа к записям, удовлетворяющим заданным условиям, в СУБД используются специальные структуры данных — индексы. R-дерево является пространственным индексом, то есть предназначено для индексирования пространственных данных, к которым относятся многомерные точки и пространственные объекты. Оно используется в таких индустриальных СУБД как PostgreSQL [20], Oracle [12, 13, 15], IBM Informix [11] и MySQL [18], а также в различных геоинформационных системах (MapInfo, TatukGIS, SuperMap).

Базовыми запросами к пространственным данным являются [5]:

1. *Запросы на совпадение отдельных координат* (partial match queries). Фиксируется одна или несколько координат. Необходимо найти все точки, у которых соответствующие координаты совпадают.

2. *Точечные запросы* (point queries). Дана точка в пространстве объектов. Необходимо найти все объекты, которым принадлежит данная точка;

3. *Диапазонные запросы* (range queries). Даны диапазоны одного или нескольких измерений. Необходимо найти все объекты, которые частично или полностью принадлежат области пространства, задаваемой диапазонами;

4. *Поиск k ближайших соседних объектов* (k nearest-neighbor search). Дана точка в пространстве объектов. Необходимо найти k точек, ближайших к данной.

Мы сфокусируем своё внимание на запросах второго и третьего типов.

Кроме того, мы будем считать, что данные являются статическими, то есть изменяются достаточно редко, чтобы мы могли построить индекс заново при обновлении. Таким образом, нам заранее известны все данные для индексирования. Алгоритмы построения, которые используют этот факт, называются «пакетной загрузкой» (bulk-loading). Они позволяют построить дерево быстрее и качественнее других алгоритмов, используя весь объём данных сразу и не прибегая к реорганизации дерева.

Примерами приложений, в которых применяется этот метод индексирования, могут служить [17]:

- геоинформационные системы,

- базы данных переписи населения,
- базы данных, хранящие информацию об окружающей среде: уровень осадков, уровень загрязнения, содержание токсинов в почве и воде.

Несмотря на то, что вопрос пакетного построения R-деревьев изучается уже 30 лет, он всё же не исчерпан. Изменения в оборудования предоставляют новые возможности для решения старых проблем. Стоит отметить две тенденции, оказывающие влияние на развитие современных СУБД. Во-первых, частота процессоров перестала расти примерно в 2005 году и повышение вычислительной мощности достигается за счёт увеличения количества ядер и совершенствования архитектуры. Во-вторых, цена оперативной памяти стабильно падает, а объём с которым может работать отдельный процессор растёт. Поэтому, если раньше естественно было предполагать, что данные хранятся во внешней памяти, то сейчас становится возможным постоянное хранения необходимых данных в основной памяти.

Исходя из этого, некоторые исследователи начали рассматривать возможность организации базы данных в оперативной памяти и отмечают перспективность такой архитектуры [8]. В настоящее время уже существует несколько СУБД использующих данный подход: VoltDB, MemSQL, SAP HANA. Кроме того, в существующие СУБД (MariaDB, SQLite, MS SQL Server 2014, Oracle Database 12c) добавляют возможности размещения таблиц в основной памяти.

Таким образом появляется необходимость в параллельных алгоритмах построения индексов, которые действуют в предположении, что оперативной памяти достаточно, чтобы построить индекс сразу по всем необходимым данным, не прибегая к чтению с диска. В связи с тем, что на данный момент не существует методов, которые были бы ориентированы на работу в таких предположениях, мы рассмотрим имеющиеся алгоритмы и возможность их адаптации для решения поставленной проблемы.

Постановка задачи

Целью данной работы является создание алгоритма пакетного построения R-дерева в архитектуре с разделяемой памятью (shared memory). При этом мы предполагаем, что:

- построение происходит с использованием нескольких потоков;
- объёма оперативной памяти достаточно, чтобы построить индекс сразу по всем необходимым данным, не прибегая к чтению с диска.

Для достижения этой цели были поставлены следующие задачи:

- провести обзор существующих методов построения статических R-деревьев,
- рассмотреть возможность их применения в обозначенных условиях,
- рассмотреть возникающие при этом проблемы,
- предложить решение обнаруженных проблем.

1. Предварительные сведения

Перед тем как начать обзор имеющихся алгоритмов пакетного построения, необходимо ввести определение R-дерева, а также сказать о том, как с помощью него выполняются запросы.

Приведём определение согласно Антонину Гуттману (Antonin Guttman) [7]. R-дерево — это древовидная структура данных, каждый узел которой представляет минимальный ограничивающий прямоугольник (MBR) своих потомков в d-мерном пространстве. R-дерево порядка (m, M) должно удовлетворять следующим характеристикам:

- Каждый листовой узел, если он не является корнем, может вмещать не более M записей и не менее $m \leq M/2$. Запись представляет собой пару (mbr, oid), где mbr — минимальный ограничивающий прямоугольник пространственного объекта, а oid — его идентификатор.

- Для внутреннего узла ограничение на количество записей является таким же как и для листового. Однако записи имеют вид (mbr, p), где p — указатель на потомка узла, а mbr — MBR этого потомка.

- Корень может содержать минимум 2 записи, если не является листом. В противном случае минимальное количество записей 0 (пустое дерево).

- Все листовые узлы должны располагаться на одном уровне.

Пример небольшого дерева показан на Рис. 1.

Поиск объектов, удовлетворяющих диапазонному запросу состоит из двух этапов. Сначала мы находим все объекты, MBR которых пересекается с запросом. Затем из таблицы извлекаются сами объекты и проводится их фильтрация — проверяется пересечение самих объектов с запросом. Прошедшие фильтрацию объекты выдаются в качестве результата. Аналогично выполняется точечный запрос.

В силу того, что прямоугольники, представляющие узлы имеют пустое пространство, не соответствующее никакому объекту, а также могут пересекаться с соседями, мы вынуждены просматривать записи, которые не релевантны запросу. Например, если на дереве, приведённом в примере (Рис. 1), мы захотим выполнить точечный

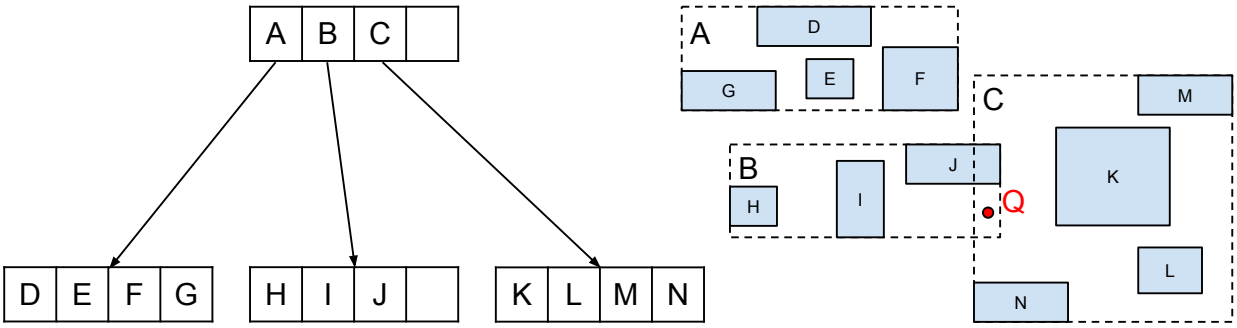


Рис. 1: Пример R-дерева и точечного запроса Q.

запрос Q, то нам необходимо будет просмотреть все записи в узлах B и C. При этом мы не найдём там ни одного объекта, удовлетворяющего запросу. Чтобы уменьшить число просматриваемых записей, алгоритмы пакетного построения стремятся минимизировать суммарный объём узлов и избегать перекрытия соседей друг с другом.

2. Обзор существующих подходов

2.1. Традиционные методы пакетного построения

Прежде всего мы в общем виде рассмотрим алгоритмы которые предназначаются для выполнения на одиночной машине и не рассчитаны на параллельное исполнение. Такие алгоритмы можно разбить на две группы по направлению построения дерева: «снизу-вверх» и «сверху-вниз».

Методы «снизу-вверх» Методы типа «снизу-вверх» также можно называть «основанными на сортировке». Их общую схему можно описать следующим образом:

1. Отсортировать прямоугольники в выбранном заранее порядке.
2. Последовательно сгруппировать прямоугольники в узлы по k штук.
3. Найти минимальный ограничивающий прямоугольник для каждой группы и создать запись, состоящую из этого прямоугольника и указателя на группу.
4. Рекурсивно применять шаги 1-3 до тех пор, пока не останется один узел — корень.

К методам этого типа относятся Packed R-tree [21], Hilbert Sort R-tree (HS R-tree) и Z-Sort R-tree (ZS R-tree) [14], которые используют для сортировка кривые, заполняющие пространство, а также Sort-Tile-Recoursive (STR) [16], которые производит поочерёдно сортировку по всем размерностям.

Методы «сверху-вниз» Мотивация алгоритмов данного типа состоит в следующем: можно ожидать, что взаимное расположение и размер узлов на верхних уровнях дерева сильнее влияют на производительность индекса, чем соответствующие характеристики листовых узлов. С этой идеей 1998 году Y. Garcia et al. [6] предложили алгоритм Top-Down Gready Split (TGS), который строит дерево «сверху-вниз».

Так же как и Packed R-tree, он заполняет узлы наперёд заданным количеством записей. Однако каждый раз рассматривается несколько разбиений и выбирается то, которое минимизирует заданную функцию стоимости. Например, суммарный объём получающихся узлов. Такой подход позволяет уменьшить ожидаемое количество вершин, которые нужно посетить при выполнении запроса [14, 23].

По сути, это первый алгоритм пакетного построения, который строит дерево основываясь формальной модели стоимости. Этим можно объяснить впечатляющие результаты, которые он показал в сравнении с Hilbert Sort и STR на нескольких наборах реальных и синтетических данных. Как и до этого, метрикой качества выступало среднее количество обращений к диску во время выполнения запроса. TGS превзошёл своих конкурентов на всех видах данных и запросов, участвующих в эксперименте. Особенно хорошо он показал себя на данных со смещением, то есть когда сильно варьируется плотность прямоугольников, их размер и соотношение сторон. В некоторых случаях количество обращений было в 2-2.7 раз меньше, чем у Hilbert Sort и STR.

Вместе с тем стоит отметить, что время построения индекса с помощью TGS в несколько раз превышает время, необходимое методам, основанным на сортировке [1]. Должно быть, именно поэтому в индустриальных СУБД отдаётся предпочтение вторым [22].

2.2. Алгоритмы для архитектуры с разделённой памятью

2.2.1. Обобщённый алгоритм построения

Проблема параллельного построения пространственного индекса стала привлекать исследователей с начала 2000-х. Впервые, обобщённый алгоритм для построения пространственного индекса в архитектуре с разделённой памятью (shared-nothing architechture) был предложен A. Papadopoulus et al. [19]. Затем, на фоне возрастающей популярности Apache Hadoop, появилась его адаптация для Map-Reduce [4] архитектуры. Так как нам важна лишь идея данного подхода к построению индекса, мы опустим несущественные для нас детали и опишем алгоритм в виде трёх этапов.

1. *Перераспределение данных.* Данные между рабочими перераспределяются таким образом, чтобы у каждого оказались пространственно близкие друг к другу объекты. При этом мы стремимся распределить данные поровну.

2. *Построение локального индекса.* Каждый рабочий строит индекс для локальных данных любым удобным алгоритмом.

3. *Построение глобального индекса.* Рабочие пересылают все узлы индекса, кроме листовых, мастеру. Мастер строит из них глобальный индекс.

Нетрудно видеть, что данный алгоритм можно использовать и в архитектуре с разделяемой памятью. При этом в качестве мастера и рабочих будут выступать потоки. Преимуществом такого подхода является гибкость — мы можем использовать любой алгоритм для построения локальных индексов.

Однако при использовании данного подхода возникает проблема балансировки глобального индекса, так как высоты локальных индексов могут различаться. Это может негативно сказаться на качестве получающегося дерева.

2.2.2. Построение «уровень за уровнем»

Не так давно Achakeev et al. [22] предложили модификацию алгоритма, которая избавлена от этого недостатка, но предполагает использование методов пакетного построения, основанных на сортировке. Идея состоит в том, чтобы строить дерево «уровень за уровнем».

Опишем данный алгоритм в виде последовательности шагов.

1. *Создание функции разбиения.*
   (a) Рабочие делают случайную выборку и посылают её мастеру.
   (b) Мастер создаёт функцию разбиения на основе полученных выборок и рассылает её рабочим.
2. *Построение листового уровня.* Каждый рабочий сортирует выделенную ему часть прямоугольников и упаковывает в листовые узлы.
3. *Обновление функции разбиения.* Мастер обновляет функцию разбиения, в соответствии с количеством узлов упакованных на последнем шаге.
4. *Построение промежуточного уровня.* Каждый рабочий сортирует, при необходимости, выделенную ему часть прямоугольников и упаковывает её в узлы.
5. Шаги 3 и 4 повторяются до тех пор, пока не получится один узел — корень.

Таким образом, мы избавляемся от проблемы балансировки глобального индекса за счёт дополнительных точек синхронизации после построения каждого уровня и увеличенного объёма передаваемой информации.

Стоить отметить, что если мы будем использовать этот подход для реализации параллельного алгоритма в архитектуре с разделяемой памятью, необходимость в пересылке данных отпадает. Таким образом мы можем получить качественный индекс, не значительно увеличив время построения.

2.3. Алгоритмы для архитектуры с разделяемой памятью

Не смотря на то, что исследование алгоритмов и структур данных для СУБД, ориентированных на работу в оперативной памяти ведутся с начала 2000-х, специальных алгоритмов для параллельного пакетного построения индекса в архитектуре с разделяемой памятью нет. Однако, как уже отмечалось выше, мы можем адаптировать существующие алгоритмы, ориентированные на разделённую память.

Ввиду того, что доступ к данным других рабочих-потоков не требует накладных расходов, мы предлагаем использовать подход «уровень за уровнем». Это позволит сохранить преимущества пакетной загрузки и избавиться от проблемы балансировки дерева.

Стоит обратить внимание и на то, что по нашим предположениям все данные помещаются в оперативную память. Поэтому у нас нет необходимости использовать метод вероятностного разбиения (Probabilistic Splitting) [3] для сортировки объектов, который применялся в оригинальном алгоритме [22]. Вместо него мы можем задействовать стандартные алгоритмы параллельной сортировки, такие как Parallel Merge Sort [2] или Parallel Quicksort [9]. Это позволит нам избавиться от дополнительного шага, на котором делалась случайная выборка. Кроме того, мы сможем более равномерно распределять нагрузку между рабочими.

Полученный адаптированный алгоритм можно описать следующим образом.

1. Рабочие параллельно сортируют прямоугольники.
2. Мастер создаёт функцию разбиения, которая распределяет данные по рабочим.
3. Каждый рабочий упаковывает выделенную ему часть прямоугольников в листовые узлы.
4. Мастер обновляет функцию разбиения.

5. Каждый рабочий сортирует, при необходимости, выделенную ему часть прямоугольников и упаковывает во внутренние узлы.

6. Шаги 4-5 повторяются до тех пор, пока не останется один узел — корень.

Выбирая тот или иной способ сортировки, мы можем получить параллельные варианты Packed R-tree, Hilbert Sort или STR.

3. Упаковка узлов

Теперь рассмотрим немного подробнее вопрос упаковки узлов. Обратим внимание, что все методы основанные на сортировке заполняют узлы фиксированным количеством записей k ($m \leq k \leq M$), которое задаётся пользователем. Это позволяет быстро сформировать узлы и при этом контролировать размер индекса. Последнее было важным преимуществом данных алгоритмов, когда проблема экономии дискового пространства стояла более остро. Сейчас же мы можем позволить использовать больше памяти, чтобы повысить производительность индекса.

В связи с этим Achakeev et al. [1] предложили отказаться от требования на фиксированный уровень заполнения узлов. В этом случае у нас появляется пространство разбиений множества записей на узлы. Теперь мы можем задать некоторую функцию стоимости на этом пространстве и выбрать разбиение, минимизирующее её. Авторы воспользовались для этого известной моделью стоимости для R-дерева, которая оценивает количество посещённых узлов для данного профиля запросов [14, 23].

Профиль запросов представляет собой прямоугольник, центр которого равномерно распределён на пространстве объектов, а размер равен ожидаемому размеру диапазонного запроса. Кроме этого модель предполагает, что пространство объектов является единичным гиперкубом. Тогда ожидаемое количество посещённых узлов в двумерном случае выражается формулой:

$$cost(\text{R-tree}) = \sum_{i=1}^{M} (dx_i + sx) \cdot (dy_i - sy) = \sum_{i=1}^{M} Area_{WQ}(r_i) \tag{1}$$

где dx_i, dy_i — длины узлов, а sx, sy — длины сторон профиля запросов

Так как разбиение по сути есть набор прямоугольников, то функцию стоимости можно без изменения перенести на пространство разбиений:

$$cost(P) = \sum_{r \in P} (rx + sx) \cdot (ry + sy) \tag{2}$$

где P — разбиение.

3.1. Алгоритм поиска оптимального разбиения

Следует отметить, что в общем случае задача нахождение оптимального разбиения относительно функции стоимости $cost(P)$ является NP-трудной [1]. Однако, если мы зафиксируем порядок следования прямоугольников, то данную задачу можно решить с помощью динамического программирования за линейное время [1]. Получающееся разбиение получило название $gopt^*$-partition.

Прежде чем представить идею алгоритма [1], введём обозначения. Пусть $\mathcal{R} = p_1, p_2, ..., p_N$ — упорядоченное множество прямоугольников, а $p_{i,j} = MBR(p_i, p_{i+1}, \ldots, p_j)$ — минимальный ограничивающий прямоугольник для отрезка. Кроме этого обозначим разбиение оптимальное на отрезке $(i : j)$ как $P_{i,j}$.

Обратим внимание, что $\forall P$ и $\forall P_{left}, P_{right}$, т.ч. $P = P_{left} \cup P_{right}$ и $P_{left} \cup P_{right} = \emptyset$

$$cost(P) = cost(P_{left}) + cost(P_{right}) \tag{3}$$

В частности, это будет выполнятся для оптимального разбиения P_{opt}, при этом P_{left} и P_{right} будут оптимальными для соответствующих им подмножеств прямоугольников. То есть разбиение, оптимальное на всём отрезке, можно представить в виде объединения разбиений оптимальных на его частях.

Тогда мы можем рекурсивно свести задачу нахождения оптимального разбиения для первых i прямоугольников к подзадаче:

$$gopt^*(i) = \min_{b \le j \le B} \{gopt^*(i-j) + cost(p_{i-j+1,i})\}$$

Таким образом, чтобы найти $P_{1,m}$ нам нужно последовательно вычислять

$$gopt(1), gopt(2), \ldots, gopt(N)$$

и запоминать соответствующие индексы $j_1, j_2, \ldots, j_m$. По запомненным индексам, мы сможем получить оптимальное разбиение.

Авторы реализовали STR, Hilbert Sort и Z-Sort алгоритмы с применение $gopt^*$-partition и сравнили производительность получающихся индексов с оригиналами (STR R-tree, HS R-tree и ZS R-tree), а также TGS. Обширные эксперименты показали, что деревья, построенные с помощью оптимизации разбиения превосходят свои оригиналы почти на всех видах тестовых данных и запросов и лишь на некоторых пока-

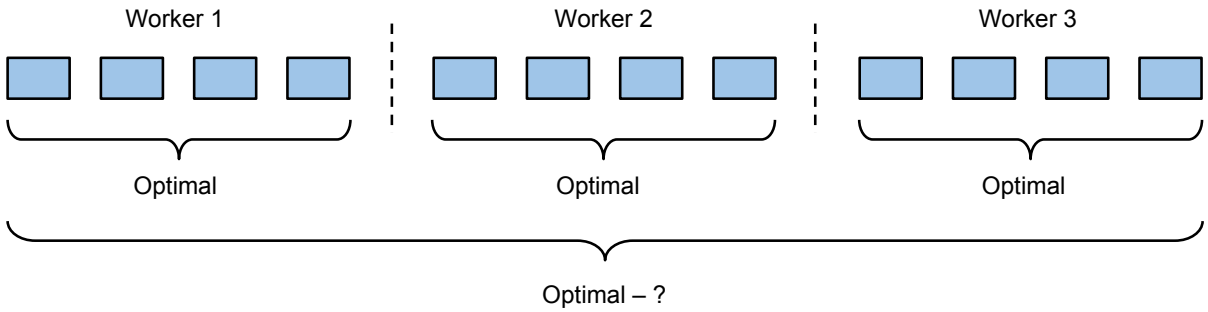


Рис. 2: Иллюстрация нахождения *gopt**-partition при параллельном построении индекса.

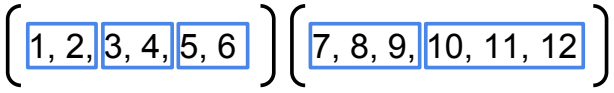
зывают равную с ними производительность. Также стоит отметить, что в условиях, удовлетворяющих модели стоимости, количество обращений к листьям для HS R-tree и ZS R-tree удалось уменьшить на 25-35% в зависимости от набора данных. Кроме этого, на реальных данных, индексы, построенные с помощью оптимизации разбиения, показывают производительность близкую к TGS, при этом время их построения в несколько раз меньше.

Таким образом использование функции стоимости для выбора разбиения позволяет повысить качество индекса, при этом не значительно пожертвовав временем построения.

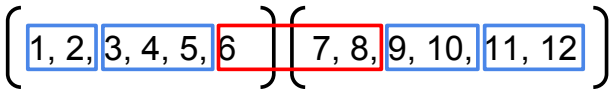
3.2. Поиск разбиения при параллельном построении

Обратим внимание, что при использовании параллельного алгоритма построения с *gopt**-partition, каждый рабочий находит разбиение, оптимальное лишь на своём отрезке (Рис. 2). Это может негативно сказаться на производительности индекса, ведь чем большее количество рабочих мы хотим использовать, тем менее оптимальным может оказаться получающееся разбиение.

D. Achakeev et al. [1] исследовали данную проблему для индекса во внешней памяти и экспериментально выяснили, что если размер отрезка, над которым работает один поток, равен M^2, то качество получающегося разбиение практически не отличается от оптимального (как если бы мы находили разбиение на всём отрезке сразу). Однако, справедливо ли это для индексов в основной памяти, они не уточняют. Если окажется, что размер отрезка, при котором качество индекса сохраняется, должен быть больше, чем M/T, где T – количество потоков, то имеет смысл искать более



(a) Рассмотренное разбиение



(b) Упущенное разбиение

Рис. 3: Примеры возможных разбиений упорядоченного набора прямоугольников

выгодное разбиение на объединённом множестве отрезков.

Не трудно заметить, что мы упускаем из рассмотрения разбиения с узлами, которые включают в себя прямоугольники из разных отрезков (Рис. 3b). Однако, мы можем воспользоваться тем, что поток имеет доступ к прямоугольникам соседнего отрезка и учесть упущенные разбиения.

Рассмотрим объединение двух соседних отрезков. Пусть в каждом из них по 6 прямоугольников, занумерованных от 1 до 12, $m = 2$, а $M = 3$. Каким бы ни было оптимальное разбиение на объединённом отрезке, мы можем найти его рассмотрев объединение оптимальных разбиений на отрезках:

$$P_{1,12} = P_{1,opt} \cup P_{opt+1,12}, \text{где}$$

$$opt = \arg\min_{6 \le i \le 8}(cost(P_{1,i}) + cost(P_{i+1,12})),$$

где $P_{i,j}$ – разбиение оптимальное на отрезке от i до j. Это следует из равенстве 3 и того, что $M = 3$.

Описанный подход позволяет нам найти оптимальное разбиение для соседних отрезков. Таким образом, при использовании T потоков, мы получаем такое же разбиение, как если бы мы использовали $T/2$ потоков (Рис. 4). Кроме того, количество прямоугольников в отрезках как правило намного больше M, а значит мы незначительно увеличиваем накладные расходы на вычисление $gopt^*$-partition.

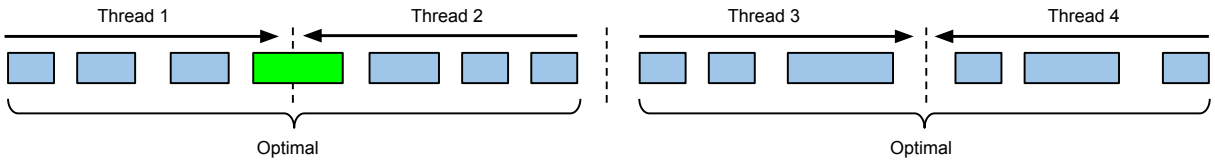


Рис. 4: Нахождение оптимального разбиения в архитектуре с разделяемой памятью

4. Алгоритм параллельного пакетного построения для архитектуры с разделяемой памятью

Принимая во внимание предложенный в предыдущем разделе способ нахождения разбиения, мы приходим к следующему параллельному алгоритму построения для архитектуры с разделяемой памятью.

1. Рабочие параллельно сортируют прямоугольники.
2. Мастер разбивает отсортированный набор прямоугольников на пересекающиеся отрезки. Каждый нечётный отрезок, пересекается со стоящим справа от него отрезком на $M - 1$ прямоугольник.
3. Создание узлов.
    - Рабочие находят $gopt^*$-partition для своих отрезков.
    - Мастер использует разбиения соседних отрезков, чтобы найти разбиение оптимальное на объединённом отрезке. После этого он сообщает рабочим как нужно упаковывать узлы.
    - Каждый рабочий упаковывает в узлы выделенный ему отрезок в соответствии с оптимальным разбиением.
4. Шаги 2 и 3 повторяются до тех пор, пока не останется один узел — корень.

Стоит отметить, что данный алгоритм можно использовать на этапе построения локального индекса при использовании метода, описанного в разделе 2.2. Таким образом мы можем добиться параллелизации построения R-дерева как на уровне кластера, так и на уровне отдельных узлов.

Заключение

Таким образом на основе ранее предложенных методов решения задачи пакетного построения дерева мы получили целое семейство алгоритмов, которые можно применять для параллельного пакетного построения в архитектуре с разделяемой памятью. Данные алгоритмы могут использоваться в СУБД для индексации таблиц с пространственными данными, которые постоянно хранятся в основной памяти. Кроме того, их можно использовать и в качестве алгоритма построения локального индекса при распределённом построении.

В дальнейшем планируется исследовать, на сколько частей мы можем разбивать множество прямоугольников при параллельном построении с помощью $gopt^*$-partition, чтобы сохранить качество индекса. Если количество таких частей не велико (меньше, чем количество потоков, которое мы хотим использовать для построения), то планируется исследовать возможность применения предложенного алгоритма и качество получающегося индекса.

Список литературы

[1] Achakeev Daniar, Seeger Bernhard, Widmayer Peter. Sort-based query-adaptive loading of R-trees // Proceedings of the 21st ACM international conference on Information and knowledge management - CIKM '12. — 2012. — P. 2080–2084. — URL: http://dl.acm.org/citation.cfm?doid=2396761.2398577.

[2] Akl S G. The Design and Analysis of Parallel Algorithms. — 1989. — ISBN: 0132000563.

[3] DeWitt D.J., Naughton J.F., Schneider D.a. Parallel sorting on a shared-nothing architecture using probabilistic splitting // [1991] Proceedings of the First International Conference on Parallel and Distributed Information Systems. — 1991.

[4] Experiences on Processing Spatial Data with Using MapReduce in Practice / Ariel Cary, Zhengguo Sun, Vagelis Hristidis, Naphtali Rishe // 21st International Conference on Scientific and Statistical Database Management. — 2009. — P. 302 – 319.

[5] Garcia-Molina Hector, Ullman Jeffrey D, Widom Jennifer. Database Systems: The Complete Book. — 2008. — P. 1248. — ISBN: 813170842X. — URL: http://www.worldcat.org/isbn/813170842X.

[6] García R Yván J., López Mario a., Leutenegger Scott T. A greedy algorithm for bulk loading r-trees // Proceedings of the sixth ACM international symposium on Advances in geographic information systems - GIS '98. — 1998. — P. 163–164. — URL: http://portal.acm.org/citation.cfm?doid=288692.288723.

[7] Guttman Antonin. R-trees: A Dynamic Index Structure for Spatial Searching // Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data - SIGMOD '84. — 1984. — P. 47–57. — ISBN 0-89791-128-8.

[8] Harizopoulos Stavros, Abadi Dj. OLTP through the looking glass, and what we found there // Proceedings of the 2008 .... — 2008. — Vol. pages. — P. 981. — URL: http://dl.acm.org/citation.cfm?id=1376713.

[9] Heidelberger Philip, Norton Alan, Robinson John T. Parallel quicksort using fetch-and-add // IEEE Transactions on Computers. — 1990. — Vol. 39, no. 1. — P. 133–138.

[10] Hennessy John L, Patterson David a. Computer Architecture, Fourth Edition: A Quantitative Approach. No. 0. — 2006. — P. 704. — ISBN: 0123704901. — URL: http://portal.acm.org/citation.cfm?id=1200662.

[11] IBM Informix R-Tree Index User's Guide. — URL: http://www-01.ibm.com/support/knowledgecenter/SSGU8G_12.1.0/com.ibm.rtree.doc/rtree.htm (дата обращения: 26.05.2015).

[12] Improving performance with bulk-inserts in Oracle R-trees / Ning An, R. Kanth, V. Kothuri, Siva Ravada // Proceedings of the 29th international conference on Very large data bases-Volume 29. — 2003. — P. 948–951. — URL: http://portal.acm.org/citation.cfm?id=1315451.1315532.

[13] Indexing medium-dimensionality data in Oracle / K. V. Ravi Kanth, Siva Ravada, Jayant Sharma, Jay Banerjee // ACM SIGMOD Record. — 1999. — Vol. 28, no. 2. — P. 521–522.

[14] Kamel Ibrahim, Faloutsos Christos. On Packing R-trees // International Conference on Information and Knowledge Management (CIKM). — 1993. — P. 490–499.

[15] Kothuri Ravi Kanth V., Ravada Siva, Abugov Daniel. Quadtree and R-tree indexes in oracle spatial: a comparison using GIS data // the 2002 ACM SIGMOD international conference on Management of data. — 2002. — P. 546–557. — URL: http://dl.acm.org/citation.cfm?id=564755.

[16] Leutenegger S.T., Lopez M.a., Edgington J. STR: a simple and efficient algorithm for R-tree packing // Proceedings 13th International Conference on Data Engineering. — 1997. — P. 497–506.

[17] Manolopoulos, Y., Nanopoulos, A., Papadopoulos, A.N., Theodoridis Y. R-Trees: Theory and Applications. — 2006. — P. 194. — ISBN: 9781852339777.

[18] MySQL 5.6 Reference Manual. — URL: http://dev.mysql.com/doc/refman/5.6/en/ (дата обращения: 26.05.2015).

[19] Papadopoulos Apostolos, Manolopoulos Yannis. Parallel bulk-loading of spatial data // Parallel Computing. — 2003. — Vol. 29, no. April 2002. — P. 1419–1444.

[20] The PostgreSQL Comprehensive Manual. — URL: http://www.postgresql.org/files/documentation/pdf/9.4/postgresql-9.4-A4.pdf (дата обращения: 26.05.2015).

[21] Roussopoulos Nick, Leifker Daniel. Direct spatial search on pictorial databases using packed R-trees // ACM SIGMOD Record. — 1985. — Vol. 14, no. 4. — P. 17–31.

[22] Sort-based parallel loading of R-trees / D Achakeev, M Seidemann, M Schmidt, B Seeger // Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data, BigSpatial 2012. — 2012. — Vol. 1. — P. 8.

[23] Towards an Analysis in Spatial of Range Data Query Performance Structures / Heinrich Toben, Peter Widmayer, Bernd-uwe Pagel, Hans-werner Six // PODS '93 Proceedings of the twelfth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems. — 1993. — Vol. 5. — P. 214–221.