электронное научно-техническое издание

НАУКА и ОБРАЗОВАНИЕ

Эл № ФС77 - 30569. Государственная регистрация №0421100025. ISSN 1994-0408

Разделение узла при индексировании интервалов

77-30569/307963

 # 01, январь 2012

А.Е. Коротков

УДК 004.65

Национальный исследовательский ядерный университет ”МИФИ”

aekorotkov@gmail.com

1. Введение

Индексирование интервалов — актуальная задача для современных баз данных (БД). Наиболее часто потребность в индексировании интервалов возникает в темпоральных БД [1]. Такие БД имеют дело с временным интервалом, заданным временем начала и временем завершения актуальности записи. Для того, чтобы темпоральная БД могла эффективно извлекать ««срезы»» данных за определенный момент времени, временные интервалы должны быть проиндексированы с использованием соответствующей структуры данных.

Проблема при индексировании интервала состоит в том, что интервал обладает протяженностью. Поэтому, хотя интервалы и могут быть линейно упорядочены, этот порядок сам по себе не позволяет эффективно выполнять поиск, например, поиск всех интервалов, пересекающихся с заданным интервалом. Из-за этого ограничения такие структуры, как B-дерево не могут быть напрямую применены для индексирования интервалов.

Существуют различные подходы к решению проблемы индексирования интервалов. Временно́й индекс [2,3] представляет адаптацию B-дерева для индексирования интервалов, однако эта структура данных требует $O(n^2)$ (n — число индексируемых интервалов) памяти, а время добавления нового интервала в индекс — $O(n)$, что неприемлемо для многих приложений. В работе [4] предлагается другая индексная структура на основе B-дерева, занимающая $O(n \cdot log(n))$ памяти, вставка в которую требует $O(log(n))$ времени. R-дерево [5], изна-

чально предназначенное для доступа к многомерным данным, имеет дело с объектами, обладающими протяженностью. Поэтому одномерное R-дерево может быть применено для индексирования интервалов [6]. R-дерево занимает $O(n)$ памяти, вставка новой записи в R-дерево требует $O(log(n))$ времени.

Производительность поисковых операций в одномерном R-дереве сильно зависит от двух параметров: степени покрытия и степени пересечения. Степень покрытия в R-дереве определяется как общая длина всех интервалов, хранящихся в узлах определенного уровня. Степень пересечения R-дерева определяется как общая длина области, содержащейся как минимум в двух интервалах, расположенных в узлах определенного уровня. Высокая степень покрытия и высокая степень пересечения означают плохую структуру R-дерева, потому что требуется просканировать множество путей для того, чтобы обработать поисковый запрос (многопутевой поиск), а это означает низкую производительность поиска. Очевидно, что минимизация как степени пересечения, так и степени покрытия очень важны для производительности R-дерева [7].

Качество R-дерева сильно зависит от алгоритма разделения узла. Задача разделения узла состоит в том, чтобы разделить записи переполненного узла на две группы, которые затем образуют два новых узла. Алгоритм разделения узла во многом определяет степень пересечения и степень покрытия R-дерева. В свою очередь эти параметры определяют вероятность того, что запрос будет использовать сканирование дерева по нескольким путям. Существуют следующие параметры, с помощью которых можно оценить качество разделения узла:

• степень пересечения ограничивающих интервалов. Меньшая степень пересечения интервалов приводит к меньшей вероятности многопутевого поиска;

• покрытие ограничивающих интервалов. Покрытие разделения — это общая площадь ограничивающих интервалов. В целом меньшее покрытие приводит к меньшей вероятности многопутевого поиска в том случае, когда запрашиваемая площадь относительно большая [8];

• эффективность хранения. В качестве меры эффективности хранения может быть использовано соотношение между числом элементов в меньшей и большей группах. Обычно на этот параметр вводится ограничение в виде минимально числа элементов в меньшей группе m. Ограничение на это параметр

очень уместно, однако этот параметр также может выступать и в качестве цели оптимизации. Лучшее соотношение при разделении ведет к меньшей балансировки дерева в процессе его создания, что, в свою очередь, влияет на его качество.

В данной работе представлен новый алгоритм разделения узла для одномерного R-дерева, основанный на двойной сортировке и позволяющий сократить время поиска по дереву за счет меньшей степени пересечения предикатов узлов. Для демонстрации эффективности предлагаемого алгоритма будет проведено его сравнение со следующими алгоритмами:

- квадратичный алгоритмом Гутмана [5];
- алгоритмами, основанными на сортировке по левой границе, правой границе или середине интервала [6].

2. Предлагаемый алгоритм

Определения. В одномерном алгоритме разделения входные элементы содержат множество I интервалов x_i: $I = \{x_i\}$. Интервал x_i определяется своими верхней и нижней границами: $x_i = (l_i, u_i)$. Общая нижняя граница это $l = \min\{l_i\}$, а общая верхняя граница — $u = \max\{u_i\}$. Вначале мы ограничим рассмотрение теми разделениями, где одна из групп содержит общую нижнюю границу, а другая — общую верхнюю границу. Для этого класса разделений назовем пару $\langle a, b\rangle$ разделяющей парой, если любой интервал из I содержится либо в интервале (l, a), либо в интервале (b, u): $\forall x(x \in I \Rightarrow (x \subseteq (l, a)) \vee (x \subseteq (b, u))$. Другими словами, a и b — это соответственно верхняя и нижняя границы групп в некотором разделении из рассматриваемого класса.

Будем называть разделяющую пару $\langle a, b\rangle$ угловой разделяющей парой, если

$$(a\in\{u_i\})\wedge(b\in\{l_i\})\wedge\forall t\big((t<a)\Rightarrow\exists x(x\in I\Rightarrow(x\not\subseteq(l,t)\wedge(x\not\subseteq(b,u))))\big)\vee$$
$$\vee\,\forall t\big((t>b)\Rightarrow\exists x(x\in I\Rightarrow(x\not\subseteq(l,\,a))\wedge(x\not\subseteq(t,\,u)))\big).$$

Другими словами a — это верхняя граница некоторого интервала, а b — нижняя граница некоторого интервала. При этом a не может уменьшится или b не может увеличится с сохранением свойства разделяющей пары. Это предположение выглядит уместным, так как иначе существовало бы другое разделение, у

которого была бы и степень пересечения меньше чем у данного, а минимально число элементов в группе не больше, т.е. оно было бы лучше тем текущее с точки зрения цели оптимизации данного алгоритма.

Случаи превосходства. Рассмотрим некоторые примеры, которые иллюстрируют случаи, где рассмотрение всех угловых разделяющих пар может иметь превосходство над другими алгоритмами разделения, которые могут быть применены к одномерному случаю. Пример 1 иллюстрирует случай превосходства над алгоритмом, основанном на сортировке по середине. Разделение, для которого есть угловая разделяющая пара представлено на рис. 1. Это разделение сочетает в себе оптимальное соотношение числа элементов и хорошую степень пересечения. Разделение с оптимальным соотношением числа элементов, которое может быть получено с помощью алгоритма, основанного на сортировке по середине, показано на рис. 2. Здесь степень пересечения хуже, чем в предыдущем разделении, потому что алгоритм сортировки по середине не может поместить интервал 5 в правую группу и одновременно поместить интервал 4 в левую группу, так как середина интервала 5 правее, чем середина интервала 4. Разделение с хорошей степенью пересечения, которое может быть получено с помощью алгоритма сортировки по середине, представлено на рис. 3. Здесь степень пересечения хорошая, потому что интервал 4 помещен в левую группу, но соотношение числа элементов не оптимально, потому что интервал 5 помещен в левую группу.

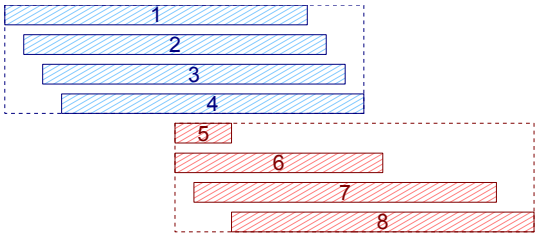

Рис. 1. Пример 1, разделение на основе угловой разделяющей пары

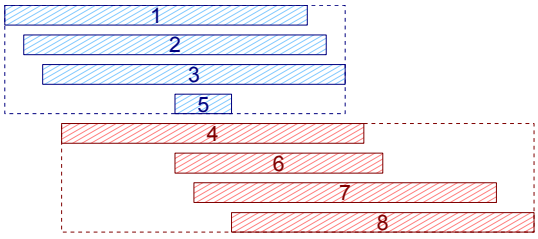

Рис. 2. Пример 1, сортировка по середине, наилучшее соотношение

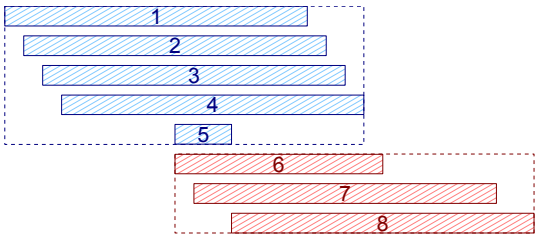

Рис. 3. Пример 1, сортировка по середине, наилучшая степень пересечения

Пример 2 иллюстрирует случай превосходства над алгоритмом, основанным на сортировке по левой границе. На рис. 4 представлено разделение, для которого есть угловая разделяющая пара. Это разделение сочетает в себе оптимальное соотношение числа элементов и хорошую степень пересечения. Разделение с оптимальным соотношением числа элементов, которое может быть получено с помощью алгоритма, основанного на сортировке по левому краю, показано на рис. 5. Здесь степень пересечения хуже, чем в предыдущем разделении, потому что алгоритм сортировки по левой границе не может поместить интервал 5 в правую группу и одновременно поместить интервалы 3 и 4 в левую группу, так как левая граница интервала 5 правее, чем левые границы интервалов 3 и 4. Разделение с хорошей степенью пересечения, которое может быть получено с помощью алгоритма сортировки по левой границе, представлено на рис. 6. Здесь степень пересечения хорошая, потому что интервал 5 помещен в правую группу, но соотношение числа элементов не оптимально, потому что интервалы 3 и 4 также помещены в правую группу. Нет смысла отдельно рас-

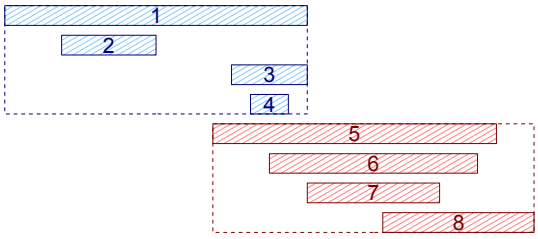

Рис. 4. Пример 2, разделение на основе угловой разделяющей пары

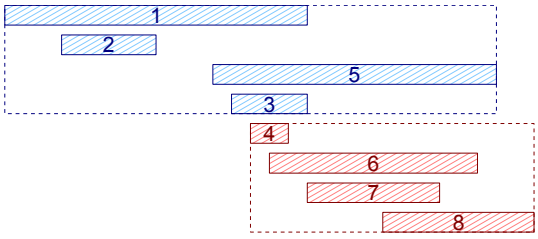

Рис. 5. Пример 2, разделение на основе угловой разделяющей пары

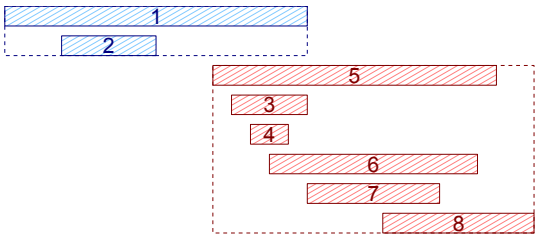

Рис. 6. Пример 2, сортировка по левой границе, наилучшая степень пересечения

сматривать алгоритм на основе сортировки по верхней границе, поскольку он симметричен алгоритму на основе сортировки по нижней границе.

Пример 3 иллюстрирует случай превосходства над Гуттмановским алгоритмом разделения. На рис. 7 представлено разделение, для которого есть угловая разделяющая пара. Это разделение сочетает в себе оптимальное соотношение числа элементов и хорошую степень пересечения. На рис. 8 представлено разделение, которое может быть получено с помощью Гуттмановского алгоритма с $m = 3$. Как соотношение числа элементов, так и степень пересечения не оптимальны. Интервалы 1 и 8 были выбраны семенами. После этого, интервалы 2, 3, 4, 5 были добавлены к левой группе, потому что для них значения разницы в приросте размеров групп было максимальным. После этого остальные интервалы были добавлены к правой группе для того, чтобы сделать минимальное число интервалов в группе не меньше m.

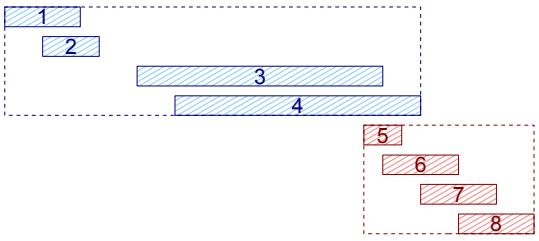

Рис. 7. Пример 3, разделение на основе угловой разделяющей пары

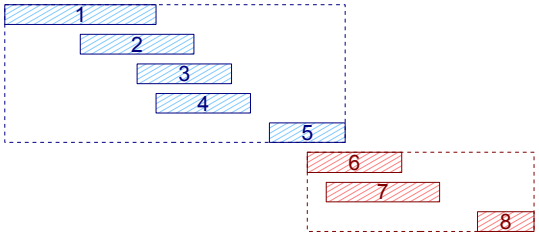

Рис. 8. Пример 3, Гутмановское разделение ($m = 3$)

Алгоритм. Алгоритм 2 EnumerateCornerSplitPairs перечисляет все угловые разделяющие пары. Алгоритм основан на использовании двух массивов:

первый содержит входные элементы, отсортированные по нижней границе, а второй содержит их же, но отсортированных по верхней границе. В главном цикле данного алгоритма делаются итерации по двум массивам одновременно так, что свойство разделяющей пары сохраняется. Когда угловая разделяющая пара найдена, вызывается алгоритм 3 ConsiderSplit. Этот алгоритм принимает ограничивающие интервалы групп и максимальное число элементов, которое может быть размещено в группы, в соответсвии с заданными ограничивающими интервалами, в качестве входных параметров. Максимальное число элементов, которые могут быть размещены в группах определяется по индексам в отсортированных массивах, откуда извлекаются границы разделяющих пар. Алгоритм ConsiderSplit ищет разделения с наименьшей степенью пересечения групп, где минимальное число элементов в группе больше или равно m. Когда разделение с нулевым пересечением возможно, то алгоритм выбирает разделение, где расстояние между ограничивающими интервалами групп наибольшее. Это свойство достигается за счёт того, что переменной overlap разрешается принимать отрицательные значения. Заметим, что если некоторые элементы могут быть размещены в обе группы, то алгоритм ConsiderSplit рассматривает разделение, где распределение элементов по группам наиболее близко к равномерному.

Алгоритм 1 DoubleSortSplit осуществляет разделение в целом. На первом шаге он вызывает алгоритм EnumerateCornerSplitPairs для того, чтобы найти угловую разделяющую пару с наименьшим пересечением. На втором шаге он распределяет элементы, которые могут быть однозначно распределены. После этого, остаток элементов сортируется по серединам их интервалов и распределяется таким образом, чтобы распределение интервалов между группами было наиболее равномерным.

Алгоритм 1 DoubleSortSplit

Вход: Переполненный узел

Выход: Два новых узла, в каждом из которых не менее m элементов

1: Вызвать EnumerateSplitPairs для того, чтобы найти угловую разделяющую пару с наименьшим пересечением.

2: Распределить между группами те интервалы, которые могут быть размещены только в одну группу.

3: Отсортировать остальные элементы по серединам их интервалов.

4: Распределить первые k из этих элементов в первую группу, а остальные распределить во вторую группу, таким образом, чтобы распределение элементов между группами было наиболее равномерным среди всех возможных k.

Алгоритм 2 EnumerateCornerSplitPairs

Вход: Набор интервалов

Выход: Перечисление всех разделений, которые могут быть получены с помощью угловых разделяющих пар, с помощью вызова ConsiderSplit

1: Отсортировать интервалы по нижней границе, записать результат в массив a

2: Отсортировать интервалы по верхней границе, записать результат в массив b

3: $s1 \Leftarrow (a[0].l, b[0].u)$

4: $s2 \Leftarrow (a[0].l, b[n-1].u)$

5: $i \Leftarrow 0$

6: $j \Leftarrow 0$

7: –Делать итерации до тех пор, пока не встретится первое разделение, полученное с помощью угловой разделяющей пары.″

8: **while** $b[j].u = s1.u$ **and** $j < n$ **do**

9: $\quad j \Leftarrow j + 1$

10: **end while**

11: considerSplit $(s1, j, s2, n - i)$

12: **while** $i < n$ **do**

13: $\quad prev\_s2\_l \Leftarrow s2.l$

14: $\quad next\_s1\_u \Leftarrow s1.u$

15: $\quad next\_i \Leftarrow i$

16: $\quad$ –Найти следующее значение верхней границы s1 и соответствующее значение нижней границы s2, которое образует с ним угловую разделяющую пару.″

17: $\quad$ **while** $next\_i < n$ **and** $next\_s2\_l = s2.l$ **do**

18: $\quad\quad next\_s1\_u \Leftarrow max\{next\_s1\_u, a[next\_i].u''$

19: $next\_i \Leftarrow next\_i + 1$

20: **if** $next\_i \geq n$ **then**

21: **break**

22: **end if**

23: $next\_s2\_l \Leftarrow a[next\_i].l$

24: **end while**

25: **if** $next\_i \geq n$ **and** $next\_s1\_u = s1.u$ **then**

26: **break**

27: **end if**

28: –Все промежуточные значения нижней границы s2 образуют угловую разделяющую пару с предыдущим значением верхней границы s1.″

29: **while** $j < n$ **and** $b[j].u \leq next\_s1\_u$ **do**

30: **if** $b[j].u > s1.u$ **and** $b[j].u < next\_s1\_u$ **then**

31: $s1.u \Leftarrow b[j]$

32: considerSplit $(s1, j + 1, s2, n - i)$

33: **else**

34: $s1.u \Leftarrow b[j]$

35: **end if**

36: $j \Leftarrow j + 1$

37: **end while**

38: –Переход к следующим значениям верхней границы s1 и нижней границы s2.″

39: $s1.u \Leftarrow next\_s1\_u$

40: $s2.l \Leftarrow next\_s2\_l$

41: **if** $next\_i < n$ **then**

42: $i \Leftarrow next\_i$

43: considerSplit $(s1, j, s2, n - i)$

44: **else**

45: considerSplit $(s1, j, s2, n - i)$

46: **break**

47: **end if**

48: **end while**

Алгоритм 3 ConsiderSplit

Вход: Ограничивающие интервалы групп $s1$ и $s2$, числа $n1$ и $n2$, которые представляют собой максимальные числа элементов, которые могут быть размещены в каждой из групп.

Выход: Обновленная информацию о том, какое оптимальное разбиение найдено на данный момент

1: $overlap \Leftarrow (s1.u - s2.l)/(s2.u - s1.l)$
2: **if** $n1 \geq m$ **and** $n2 \geq m$ **and** $overlap < best\_overlap$ **then**
3: $best\_overlap1 \Leftarrow overlap$
4: $best\_s1 \Leftarrow s1$
5: $best\_s2 \Leftarrow s2$
6: $best\_n1 \Leftarrow n1$
7: $best\_n2 \Leftarrow n2$
8: **end if**

3. Тесты производительности

Экспериментальный стенд. Все тесты проводились на компьютере с процессором Core 2 Duo 3 GHz с 2 GB оперативной памяти и ОС Ubuntu 10.10 32bit. Для реализации R-дерева с различными алгоритмами разделения узлов был использован фреймворк GiST [9] в СУБД PostgreSQL. GiST обобщает различные поисковые деревья, включая R-дерево.

Наборы данных. Каждый набор данных содержит 10^6 случайно сгенерированных интервалов. Размер интервалов подчиняется нормальному закону распределения с нулевым математическим ожиданием и среднеквадратичным отклонением, которое обеспечивает требуемую степень пересечения интервалов. Под степенью пересечения интервалов подразумевается среднее число интервалов, которые будут содержать случайную точку в диапазоне $[0; 1)$. Степень пересечения интервалов варьировался экспоненциально от 1 до 10^4. Распределение середины интервалов зависит от типа набора данных следующим образом.

• Равномерный набор данных. Середины интервалов подчиняются равномерному закону распределения на интервале $[0; 1)$.

• Нормальный набор данных. Середины интервалов подчиняются стандартному нормальному закону распределения.

• Равномерный набор данных с кластерами. Вначале генерируется 500 середины кластеров, которые подчиняются равномерному закону распределения на интервале $[0; 1)$. После этого для каждой середины кластера генерируется 2000 середин интервалов, сдвиг которых относительно середины кластера подчиняется равномерному закону распределения на интервале $[0; 6 * 10^{-4})$.

• Нормальный набор данных с кластерами. Вначале генерируется 500 середин кластеров, которые подчиняются стандартному нормальному закону распределения. После этого для каждой середины генерируется 2000 середин интервалов, сдвиг которых относительно середины кластера подчиняется нормальному закону распределения с нулевым математическим ожиданием и среднеквадратичным отклонением $6 * 10^{-4}$.

Результаты тестирования. Тесты показывают, что все алгоритмы, основанные на сортировке, работают почти одинаково. Поэтому здесь представлен только один подобный алгоритм, основанный на сортировке по середине интервала. Следующие алгоритмы разделения узлов были включены в тесты для одномерного случая:

• квадратичный гуттмановский алгоритм [5];

• алгоритм, основанный на сортировке по середине интервала [6];

• предложенный алгоритм, основанный на двойной сортировке.

Для того чтобы сравнить эффективность индексных структур, полученных с помощью различных алгоритмов разделения узла, измерялось число доступа к узлам при выполнении запроса. Для тестирования было сгенерировано 100 маленьких интервалов размером 10^{-5}, и измерялось число доступов к узлам дерева для извлечения всех интервалов из тестового набора данных, которые пересекаются с данным интервалом. На рисунке 9 представлено сравнение среднего числа доступов к узлам. Для более наглядного сравнения, представлено не абсолютное значение, а отношения значения для конкретного алгоритма к среднему значению по всем рассматриваемым алгоритмам. Измерения были проделаны для 4-х наборов данных, представленных выше, с различными уровнями пересечения. На рисунке 10 приведено сравнение времени создания дерева. Данными представлены тем же способом, что и число доступов к узлам:

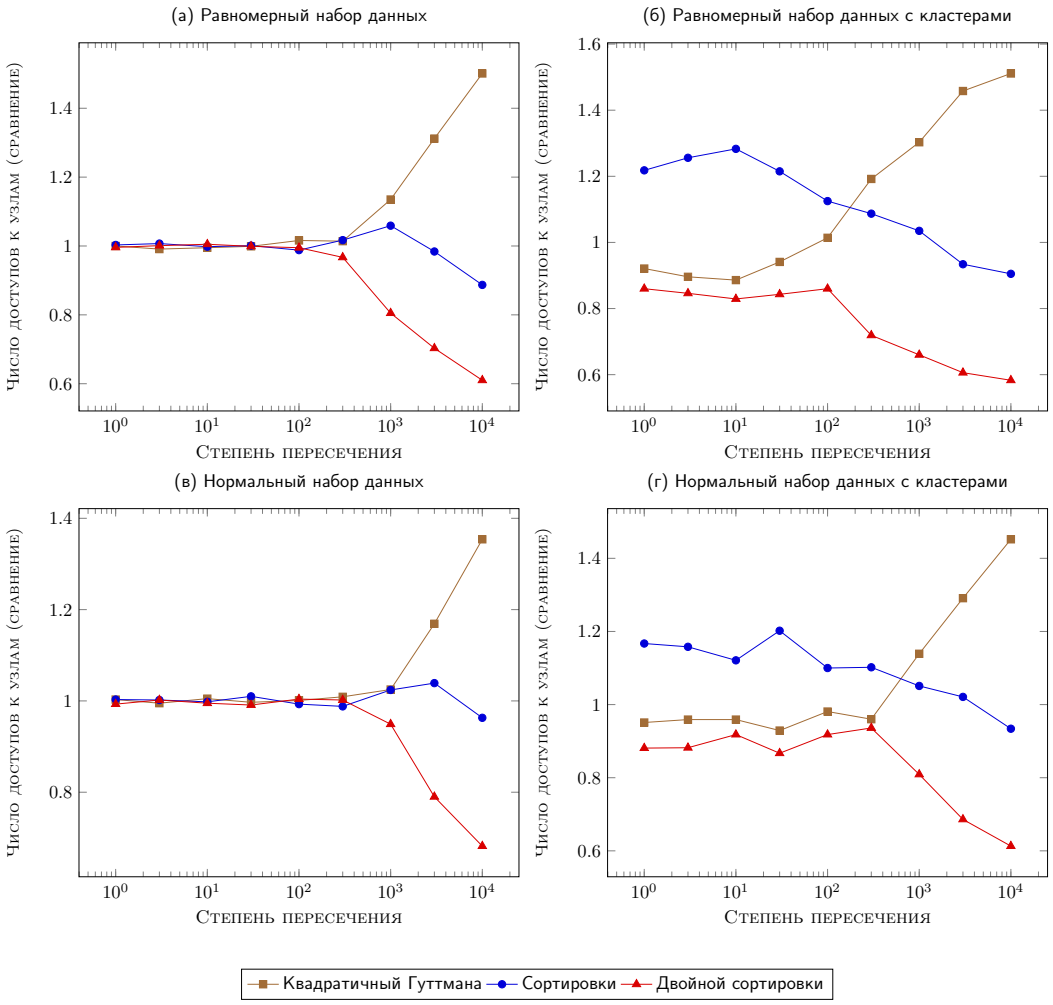

Рис. 9. Сравнение числа доступов к узлам при поиске для различных одномерных алгоритмов разделения

отношение времени создания для конкретного алгоритма к среднему времени создания.

Мы можем видеть, что число доступов к узлам, необходимое для поиска в дереве, построенном с использованием разделения узла на основе двойной сортировки, почти никогда не оказывается больше, чем при использовании других алгоритмов. При высокой степени пересечения данных наблюдается существенное превосходство алгоритма на основе двойной сортировки, до 50% в сравнении с алгоритмом, основанном на сортировке, и до 2 раз по сравнению с Гуттмановским квадратичным алгоритмом. Мы можем видеть, что время создания дерева при использовании алгоритма, основанного на двойной сортировке, меньше, чем при использовании Гуттмановского квадратичного алгоритма, но,

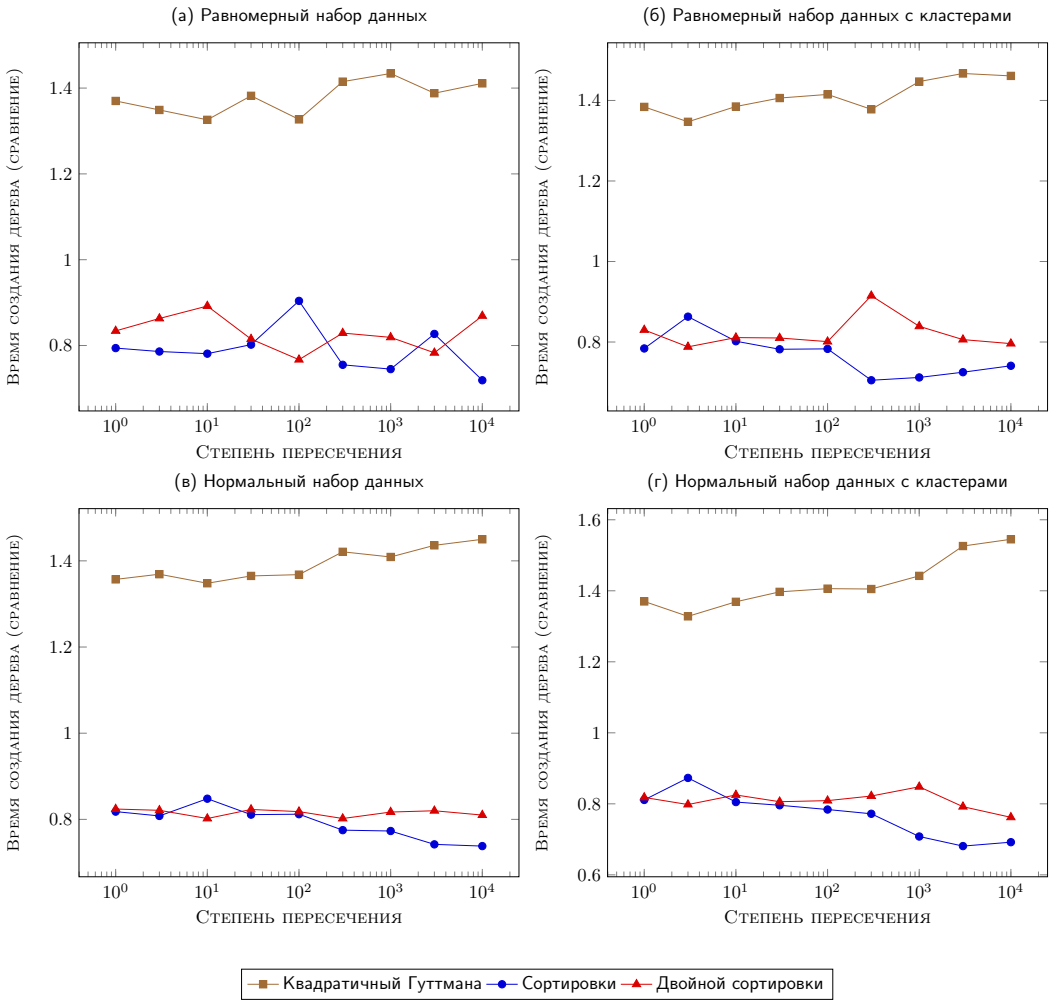

Рис. 10. Сравнение времени создания дерева для различных одномерных алгоритмов разделения

в среднем, несколько выше, чем при использовании алгоритма, основанного на сортировке.

4. Заключение

В данной работе предлагается новый алгоритм разделения узлов для одномерного R-дерева, основанный на двойной сортировке. Этот алгоритм позволяет лучше обрабатывать сложные случаи. Тесты показывают превосходство предложенного алгоритма в отношении числа доступов к узлам дерева по сравнению с квадратичным Гуттмановским алгоритмом и с алгоритмом, основанном на одной сортировке. Наибольшее превосходство достигается при сильном пересечении исходных данных, благодаря способности предложенного алгоритма лучше обрабатывать сложные случаи.

Список литературы

1. Temporal databases: theory, design, and implementation / Ed. by A. U. Tansel, J. Clifford, S. Gadia et al. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1993.
2. *Elmasri R.* The time index: An access structure for temporal data / R. Elmasri, G. T. J. Wuu, Y.-J. Kim // Proc. 16th Int. Conf. on Very Large Data Bases. VLDB '90. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990. P. 1–12.
3. *Ang C.-H.* The interval b-tree / C.-H. Ang, K.-P. Tan // Inf. Process. Lett. 1995. January. V. 53. P. 85–89.
4. *Nascimento M.A.* Indexing valid time databases via b+-trees / M.A. Nascimento, M.H. Dunham // IEEE Trans. on Knowl. and Data Eng. 1999. November. V. 11. P. 929–947.
5. *Guttman A.* R-trees: a dynamic index structure for spatial searching / A. Guttman // *SIGMOD Rec.* "— 1984. "— June. "— Vol. 14. "— Pp. 47–57. .
6. *Kolovson C.P.* Segment indexes: Dynamic indexing techniques for multi-dimensional interval data / C.P. Kolovson, M. Stonebraker // SIGMOD Conference / Ed. by J. Clifford, R. King. ACM Press, 1991. P. 138–147.
7. *Brakatsoulas S.* Revisiting r-tree construction principles / S. Brakatsoulas, D. Pfoser, Y. Theodoridis // Proc. 6th East European Conf. on Advances in Databases and Information Systems. ADBIS '02. London: Springer-Verlag, 2002. P. 149–162.
8. *Al-Badarneh A.F.* A new enhancement to the r-tree node splitting / A.F. Al-Badarneh, Q. Yaseen, I. Hmeidi // J. Information Science. 2010. V. 36, No 1. P. 3–18.
9. *Hellerstein J.M.* Generalized search trees for database systems / J.M. Hellerstein, J.F. Naughton, A. Pfeffer // Proc. 21th Int. Conf. on Very Large Data Bases. VLDB '95. San Francisco: Morgan Kaufmann Publishers Inc., 1995. P. 562–573.

electronic scientific and technical periodical

SCIENCE and EDUCATION

El № FS77 - 30569. №0421100025. ISSN 1994-0408

Node Splitting on Intervals Indexing

77-30569/307963

 # 01, January 2012

A. E. Korotkov

National research nuclear university ”MEPhI”

aekorotkov@gmail.com

Indexing of intervals is important task for modern databases. There are various index types for intervals indexing, while one-dimensional R-tree is one of them. The main problem of R-tree is overlap of its node predicates. High level of such overlap leads to search query performance slowdown. This paper presents new algorithm for one-dimensional R-tree node splitting, which allows decreasing overlap of node predicates. This algorithm is based on using two sorting simultaneously, which makes it handle some sophisticated cases better.

References

1. Temporal databases: theory, design, and implementation / Ed. by A. U. Tansel, J. Clifford, S. Gadia et al. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1993.
2. *Elmasri R.* The time index: An access structure for temporal data / R. Elmasri, G. T. J. Wuu, Y.-J. Kim // Proc. 16th Int. Conf. on Very Large Data Bases. VLDB ’90. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990. P. 1–12.
3. *Ang C.-H.* The interval b-tree / C.-H. Ang, K.-P. Tan // Inf. Process. Lett. 1995. January. V. 53. P. 85–89.
4. *Nascimento M.A.* Indexing valid time databases via b+-trees / M.A. Nascimento, M.H. Dunham // IEEE Trans. on Knowl. and Data Eng. 1999. November. V. 11. P. 929–947.

5. *Guttman A.* R-trees: a dynamic index structure for spatial searching / A. Guttman // *SIGMOD Rec.* ”— 1984. ”— June. ”— Vol. 14. ”— Pp. 47–57. .
6. *Kolovson C.P.* Segment indexes: Dynamic indexing techniques for multi-dimensional interval data / C.P. Kolovson, M. Stonebraker // SIGMOD Conference / Ed. by J. Clifford, R. King. ACM Press, 1991. P. 138–147.
7. *Brakatsoulas S.* Revisiting r-tree construction principles / S. Brakatsoulas, D. Pfoser, Y. Theodoridis // Proc. 6th East European Conf. on Advances in Databases and Information Systems. ADBIS ’02. London: Springer-Verlag, 2002. P. 149–162.
8. *Al-Badarneh A.F.* A new enhancement to the r-tree node splitting / A.F. Al-Badarneh, Q. Yaseen, I. Hmeidi // J. Information Science. 2010. V. 36, No 1. P. 3–18.
9. *Hellerstein J.M.* Generalized search trees for database systems / J.M. Hellerstein, J.F. Naughton, A. Pfeffer // Proc. 21th Int. Conf. on Very Large Data Bases. VLDB ’95. San Francisco: Morgan Kaufmann Publishers Inc., 1995. P. 562–573.