Вычислительная техника и информационные технологии

УДК 004.4
DOI: 10.12737/24904

В.К. Гулаков, К.В. Гулаков, Ю.А. Сковородников

ИССЛЕДОВАНИЕ МЕТОДОВ И АЛГОРИТМОВ ХЕШИРОВАНИЯ ДВИЖУЩИХСЯ ОБЪЕКТОВ

Предложена идея, основанная на методе хеширования, позволяющая значительно уменьшить количество обновлений базы данных и сделать осуществимой процедуру индексации в задачах реального времени, где объекты имеют пространственные и временные зависимости. Представлены экспериментальная оценка и сравнение эффективности предложенных методов хеширования.

Ключевые слова: пространственно-временные объекты, индексирование, деревья, хеширование.

V.K. Gulakov, K.V. Gulakov, Yu.A. Skovorodnikov

INVESTIGATION OF METHODS AND ALGORITHMS OF MOVING OBJECT HASHING

The paper deals with the study of one of the urgent problems of real time applications in which objects have spatial and temporal dependences. With the development of wireless communication and positioning technologies the problem of storage in a database and indexing a large quantity of moving objects becomes urgent. In this paper there is offered an idea based on the method of hashing allowing the considerable decrease of the quantity of database updates and the fulfillment of an indexing procedure possible. The paper reports the following basic approaches to the solution of the problem mentioned: a hashing; LP-layer; a division of space into blocks; coverings between blocks; the use of a dynamic update of blocks, and also combinations of approaches.

In order to draw conclusions of efficiency of hash-functions offered there was carried out a work on the experimental assessment of methods mentioned. The results of the comparison of methods are presented according to different criteria: productivity, a quantity of database updates, amount of disk memory pages used. The conclusions and recommendations for use are formulated.

Key words: spatial-temporal objects, indexing, trees, hashing.

Введение

Основная проблема заключается в том, что нам необходимо эффективно индексировать большое количество перемещающихся объектов, не генерируя большого объема обновлений базы данных.

В данной статье рассматриваются следующие подходы к решению данных проблем:

- Идея, основанная на методе хеширования. Каждый объект помещают в блок. Только тогда, когда объект выходит на новый блок, в базе данных выполняется обновление. Этот метод значительно уменьшает количество обновлений базы данных, что позволяет системе хранить и индексировать большое число движущихся объектов.
- Структура системы. Между коллектором позиции и базой данных добавляется новый слой под названием «Предварительная обработка расположения» (LP-слой) [4]. При помощи этого слоя можно отфильтровать большую часть запроса обновления базы данных на основе правил.
- Четыре различных метода. Первый метод разделяет пространство на небольшие блоки. Второй использует некоторые перекрытия между блоками, которые уменьшают обновления базы данных, когда рассматриваются

зигзагообразные движения объектов. Третий метод использует динамическое обновление блока, который увеличивает эффективность хранения, при неравномерном распределении объектов. Последний метод сочетает в себе преимущества второго и третьего методов.

Техника хеширования и структура системы

Основное различие между пространственно-временными и статическими объектами в том, что расположение динамических объектов часто меняется. Если мы хотим отслеживать точную информацию о местоположении объектов в базе данных, то неизбежно создание большого объема обновлений базы данных. Таким образом, вводим «нечеткую» идею: мы не обновляем расположение объекта в базе данных, если его текущее местоположение не очень далеко от исходного положения [4]. Для того чтобы решить эту проблему, необходимо использовать совершенно новую структуру. Структура для метода хеширования представлена на рис. 1.

Сначала вводится хеш-функция, которая использует текущее состояние объекта в качестве входных данных [1]. Из этой функции система способна определить, какому блоку каждый объект принадлежит. База данных хранит только информацию блока: сколько объектов в каждом блоке, в каком блоке находится каждый объект в настоящее время. Между базой данных и динамическими объектами добавляется набор фильтров под названием «Предварительная обработка расположения» (LP). Каждый LP контролирует небольшое подмножество объектов и использует массив для хранения последнего состояния этих объектов.

Когда объект меняет свое местоположение и генерирует запрос на обновление, запрос сначала идет к соответствующему LP. LP обновляет состояние объекта локально, применяя хеш-функцию, чтобы увидеть, находится ли объект все в том же блоке. Если это так, то запрос просто игнорируется. Для тех объектов, которые перемещаются в новый блок, применяются запросы на обновление.

На рис. 1 курсивные линии показывают процедуру индексации, а полужирные – процедуру запроса.

Рассмотренная структура является очень гибкой: выбирая различные хеш-функции, мы используем разные методы. В следующем разделе мы рассмотрим четыре различных метода, основанных на той структуре.

Рис. 1. Структура техники хеширования

Хеш-функции

В этом разделе представлены четыре различные хеш-функции и дается детальное описание соответствующих методов.

1. Метод разделения пространства без пересечений

Основная идея первого метода состоит в том, что пространство разбивается на несколько частей. Каждая часть сопоставляется с блоком в базе данных. Только тогда, когда объект покидает одну часть и переходит в другую, в базе данных осуществляется операция обновления.

Цель первого этапа данного метода состоит в разделении пространства на множество мелких частей. На рис. 2 пред-

ставлено рабочее пространство. Область покрытия каждой части может рассматриваться как блок в базе данных. В идеале каждый сегмент все время содержит одно и то же количество объектов. Тем не менее разделение на области сделано заранее, и у нас нет подсказки о том, как объекты двигаются.

После разделения пространства мы даем каждой части уникальный идентификатор. Хеш-функция теперь имеет вид $f(p)=i$, где p - объект и i - идентификатор блока p .

Когда во время t объект перемещается из одного сегмента в другой (путь I на рис. 2), LP, который контролирует этот объект, отправляет запрос на обновление базы данных. Запрос на обновление имеет вид *query(part_id, old_bucketid, new_bucketid, t)*. Иногда после того, как объект меняет свое местоположение, он все еще остается в том же сегменте (например, путь II на рис. 2). В таком случае в базе данных не происходит никаких изменений.

Часть запроса интуитивна после определения LP. Стоит отметить, что блоки информации в данном случае являются статическими. Это означает, что информация (размер, расположение и т.д.) блоков никогда не изменяется, как только задана хеш-функция f. Это позволяет нам использовать существующие пространственные

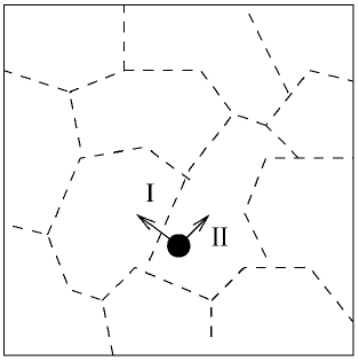
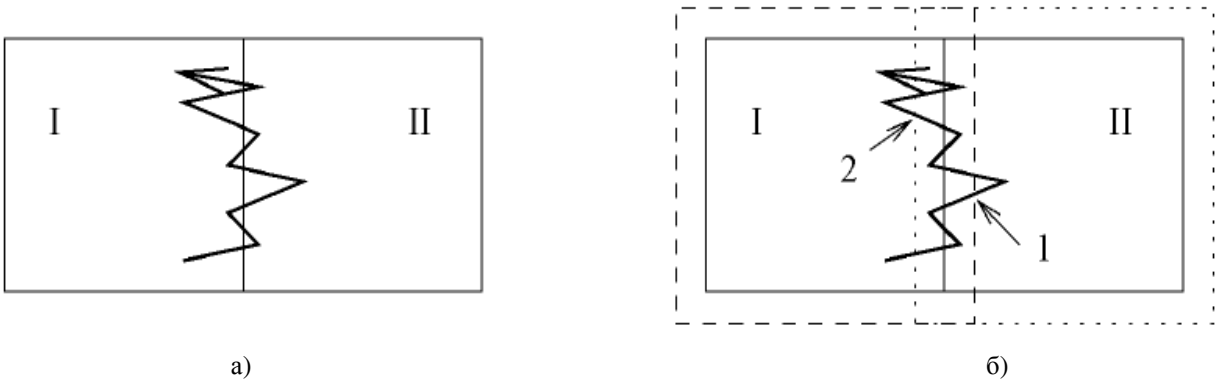


Рис. 2. Пример перемещения объекта

индексные структуры (R*-дерево, Quad-дерево) для управления блоками в базе данных, что значительно ускоряет процедуру запроса к базе данных.

2. Метод разделения пространства расширением

В методе разделения пространства без пересечений объекты, которые хаотично двигаются вдоль границы сегмента, могут принести большие неприятности (рис. 3а). Так как область в рабочем пространстве однозначно сопоставляется с блоком в базе данных, в дальнейшем эти два понятия имеют равное значение.



а) б)

Рис. 3. Движение объекта вдоль границы

Для того чтобы решить эту проблему, мы немного увеличиваем размер каждого сегмента – так, чтобы существовала некоторая область перекрытия между двумя блоками. Запрос на обновление генерируется только тогда, когда объект выходит из области пересечения. Объект начинает движение в блоке I (штриховой квадрат). В точке 1 он оставляет блок I и переходит в блок II (пунктирный квадрат). Он движется в блоке II и в точке 2 переходит обратно

к блоку I. В данном случае сгенерируются два запроса на обновление базы данных.

Метод разделения пространства расширением работает следующим образом. Сначала выполняется разделение пространства без перекрытий. Затем хешируется каждый объект в блоке согласно его исходной позиции. После этого каждый блок делает δ-расширение, которое заключается в том, что центр блока не изменяется, но область покрытия увеличивается на небольшую длину δ. Например, если блок охватывает прямоугольник $[x_0, y_0][x_1, y_1]$, то после δ-расширения он будет охватывать $[x_0 - \delta, y_0 - \delta][x_1 + \delta, y_1 + \delta]$.

В этом методе вводится атрибут previous_bucketid (id предыдущего блока). Этот атрибут запоминает, в каком блоке ранее находился объект. Оставшаяся часть индексации и процедуры запроса такая же, как и в методе разделения пространства без пересечений.

3. Метод хеширования Quad-деревьями

Другим недостатком метода разделения пространства без пересечений является то, что его блоки не имеют возможности изменять свой размер или местоположение после первоначального определения. В определенных моментах это может вызвать некоторые проблемы (например, при направленном движении). Для того чтобы найти решение для этой проблемы, мы введем некоторые динамические структуры. Основная идея состоит в том, чтобы динамически изменять область обзора каждого блока. Когда блок содержит слишком много объектов, мы разделяем его на несколько более мелких и делаем перераспределение объектов в старом блоке. С другой стороны, если в нескольких блоках мало объектов, мы объединяем их в один большой блок и ставим объекты вместе [3].

В базе данных мы создаем новую часть под названием «Блок управления» (БУ). БУ использует пространственную структуру индекса управления блоками. Любое изменение в блоках (например, добавление или удаление объекта в блоке) вызовет действие в БУ. БУ проверяет изменяющиеся блоки и решает, какая необходима операция, стоит ли разделить (объединить) блок. В качестве пространственной структуры индекса мы используем Quad-дерево, поскольку оно имеет простую структуру и алгоритмы разделения и слияния. Мы не поддерживаем использование R-дерева, так как каждый внутренний узел в Quad-дереве имеет точно четыре дочерних узла и не допускается никакого дублирования между узлами.

Когда количество объектов в одном листовом узле превышает допустимое значение, тогда и только тогда узел разделяется. Определим M – максимальное количество объектов, которые могут храниться в одной дисковой странице. Алгоритм разделения сначала создает четыре дочерних узла, каждый из которых охватывает четверть первоначальной площади покрытия. Это сообщается в базу данных для генерации четырех блоков. Объекты в старых блоках затем проверяются и повторно устанавливаются в новых генерируемых блоках. После этого в базе данных удаляются старые блоки, а также сообщается количество объектов в новых блоках Quad-дерева, которые сохраняются в новых конечных узлах.

Условие запуска операции слияния является более сложным. Если конечный узел содержит менее m объектов, нам все еще нужно проверить, как много объектов содержится в родственных узлах. Вполне возможно, что один из его соседних узлов еще полон. Кроме того, поскольку операция слияния узла очень дорогая, мы не хотим вновь генерировать узел, а должны разбить его очень быстро. Так, в нашем алгоритме мы определяем состояние «конечный узел имеет менее m объектов и количество объектов в родительском узле меньше $\frac{3M}{4}$».

Когда узел удовлетворяет этому условию, его родительский узел становится новым листовым узлом. База данных генерирует новый блок. Объекты из ста-

рых блоков перемещаются на новый блок, а старые просто удаляются.

Так как блоковая структура теперь является динамической, LP должны определять текущую структуру, чтобы должным образом фильтровать запрос на обновление. БУ будет транслировать структуру индекса к LP после каждого изменения. Этот метод является возможным только тогда, когда блоковая структура изменяется не часто или же мы имеем небольшое количество LP в системе. Рис. 4 показывает, как выглядят блоки при использовании структуры динамических блоков.

Последний метод сочетает в себе идеи Quad-деревьев метода хеширования и метода разделения пространства расширением. На этот раз мы немного дополним каждый узел Quad-дерева.

4. Расширенный метод хеширования Quad-деревьями

Как и в методе разделения пространства расширением, EQ-дерево использует в индексации расширенные узлы. Алгоритмы индексации, вставка и удаление почти такие же, как и в методе хеширования Quad-деревьями.

Экспериментальная часть

Для того чтобы сделать выводы об эффективности предложенных хеш-функций, была выполнена определенная подготовка к экспериментальной оценке различных методов.

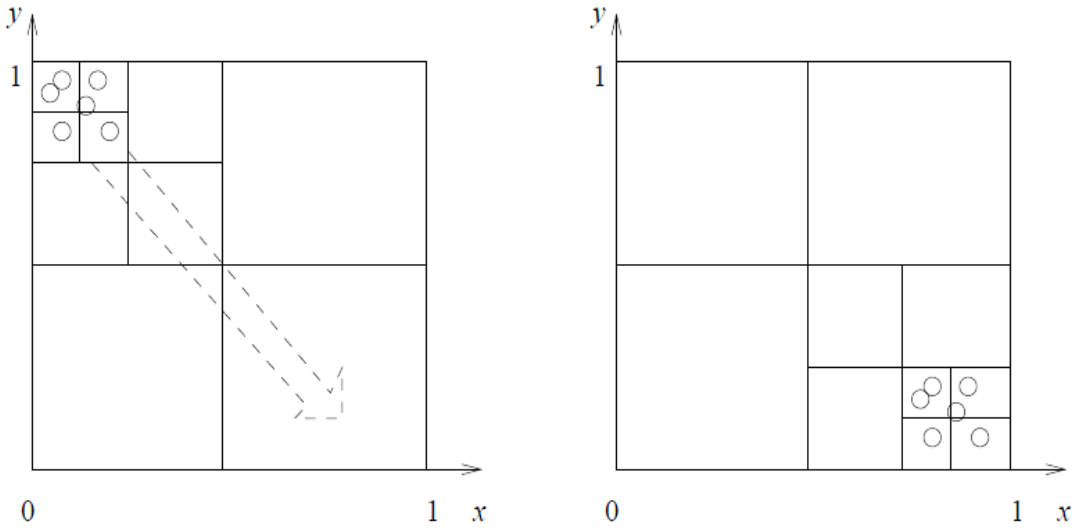


Рис. 4. Динамические блоки структуры

1. Описание и начальные установки системы

Мы используем симулятор данных на основе широко применяемого сравнительного анализа среды под названием «Генерация пространственно-временных данных» (GSTD) [3]. Как и в GSTD, наш симулятор данных поддерживает начальные распределения объектов: *равномерное и гауссовское*. В нашем эксперименте мы только проиндексируем и запросим текущие состояния движущихся объектов.

В нашем эксперименте мы используем язык Java из-за его хорошей поддержки потоков. После определенного периода времени каждый перемещающийся объект вычисляет свое новое место расположения. Есть два LP-слоя в системе, каждый контролирует половину объектов. После определенного периода времени LP проверяет последний статус перемещающихся объектов и отправляет отфильтрованную информацию в БУ. БУ собирает информацию и сообщает результаты эксперимента.

Используется 20 байт для представления каждого двухмерного объекта (2

doubles для расположения и 1 целое число для ID). Размер страницы составляет 4K и позволяет разместить 204 2-мерных объекта в одной странице. В обоих методах Quad-деревьев мы выбираем M=200 и m=50.

Большинство методов также индексируют хронологическую информацию для каждого объекта. Их параметры делают невозможной индексацию большого числа объектов. Таким образом, несправедливо сравнивать наши методы с ними. Кроме наших четырех методов мы также применяем метод R-дерева. Метод R-дерева использует традиционный статический метод в проблеме перемещающихся объектов.

Обозначения. Обозначения, используемые в данном разделе, для удобства были обобщены и представлены в табл. 1. Аббревиатуры алгоритмов, применяемых в этой части, находятся в табл. 2.

Затраты на индексацию и запросы. Прежде чем перейти к эксперименту, нам сначала необходимо изучить факторы, которые имеют значительное влияние на общую производительность. Общая стоимость индекса C_{index} включает в себя главным образом три составляющие: обновления базы данных, изменения структуры блока и связи между LP и базой данных. Другие факторы, такие как последняя позиция коллекции или LP-обновления, не являются сильно затратными из-за структуры параллельной системы. Поэтому

$$C_{index} = \mathrm{DU\#}*\mathrm{C_u} + \mathrm{C_b} + \mathrm{C_c}.$$

Таблица 1

Обозначения, используемые в эксперименте

v	Скорость объекта
$\bar{v}$	Средняя скорость всех объектов
$\sigma(v)$	Стандартное отклонение скорости
S	Размер блока
DU#	Количество обновлений базы данных
$\sigma(D)$	Стандартное отклонение начального распределения
$\mathrm{C_u}$	Стоимость одного обновления базы данных, которая включает операцию удаления и вставки
$\mathrm{C_b}$	Стоимость изменения структуры блока
$\mathrm{C_c}$	Стоимость связи между LP и базой данных
$\mathrm{C_q}$	Стоимость запроса к базе данных

Таблица 2

Обозначения алгоритмов

RT	R-tree метод
SP	Метод разделения пространства без пересечений
ASP	Метод разделения пространства с расширением
QH	Метод хеширования Quad-деревьями
EQH	Расширенный метод хеширования Quad-деревьями

Набор данных. Набор данных состоит из 100 000 объектов в рабочем пространстве. Мы рассматриваем производительность различных методов при помощи двух начальных распределений объектов и двух типов движения, которые описаны ниже.

- *Два вида начальных распределений.* Первый – равномерное распределение. В этом случае объекты

равномерно распределены в рабочем пространстве. Вторым видом является гауссовское распределение. На этот раз объекты сгруппированы вокруг одной или нескольких центральных точек. Если мы используем распределение Гаусса, то устанавливаем $\sigma(D)$ равным 0,1 в единичном пространстве.

- *Два типа движения.* Метод GSTD является очень мощным в описании движения объектов. В нашей программе мы заимствовали эти идеи. Полностью определены два типа движения для нашего эксперимента. Первым типом является случайное движение. Возможные значения представлены в табл. 3. Детали значения каждого параметра можно найти в [3]. Второй тип движения можно назвать «направленное движение». Используемые значения представлены в табл. 4. Этот тип движения полезен тогда, когда, например, мы рассматриваем движение автомобиля.

Таблица 3

Значения для случайного движения объектов

$\bar{v}$	$\sigma(v)$	$\max_x(speed)$	$\max_y(speed)$	$\min_x(speed)$	$\min_y(speed)$
0	0,005	0,005	0,005	-0,005	-0,005

Таблица 4

Значения для направленного движения объектов

$\bar{v}$	$\sigma(v)$	$\max_x(speed)$	$\max_y(speed)$	$\min_x(speed)$	$\min_y(speed)$
0,005	0,005	0,01	0,01	0	0

Набор запросов. Набор запросов состоит из 1000 прямоугольников, лежащих в пределах рабочего пространства. Мы выбираем центры прямоугольников случайно. Размер прямоугольника в запросе составляет 1% от общей площади. В конце каждого периода времени один запрос выбирается случайным образом.

2. *Экспериментальные результаты*

Эксперимент 1. Влияние размера блока.

В этом разделе нам необходимо изучить производительность индекса при различных S и $\bar{v}$ в методе SP. Определим хеш-функцию $f(x, y) = i \cdot \text{int}(y \cdot i) + \text{int}(x \cdot i)$. Изменяем значения i; $i \times i$ у нас будет равным размеру квадратных блоков. Тогда мы фиксируем $\bar{v}$ равным 0,005 и записываем значения производительности. Результаты отображены на рис. 5.

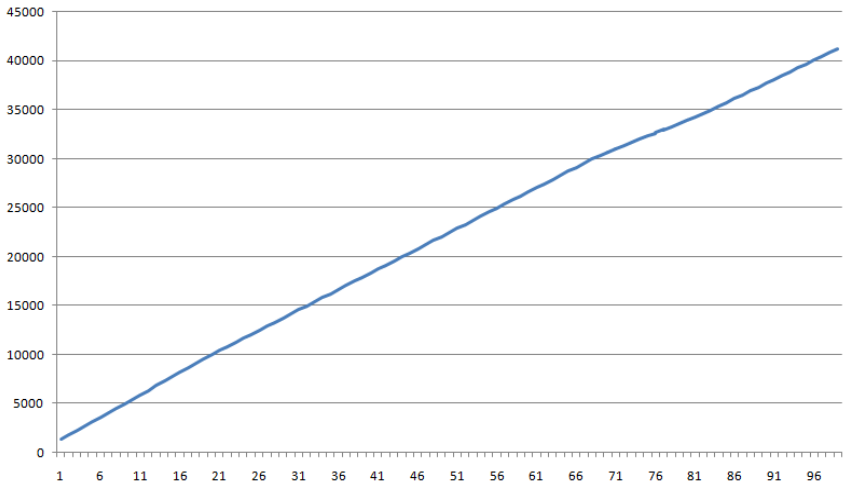


Рис. 5. Производительность относительно S, $\bar{v}$ =0,005

Они свидетельствуют о том, что производительность пропорциональна i. На

рис. 6 мы исправили S и узнали, что производительность также пропорциональна $\bar{v}$.

Таким образом, главный вывод в этом эксперименте:

$$DU\#\infty\frac{\bar{v}}{\sqrt{S}}.$$

Эксперимент 2. Исследование производительности индекса.

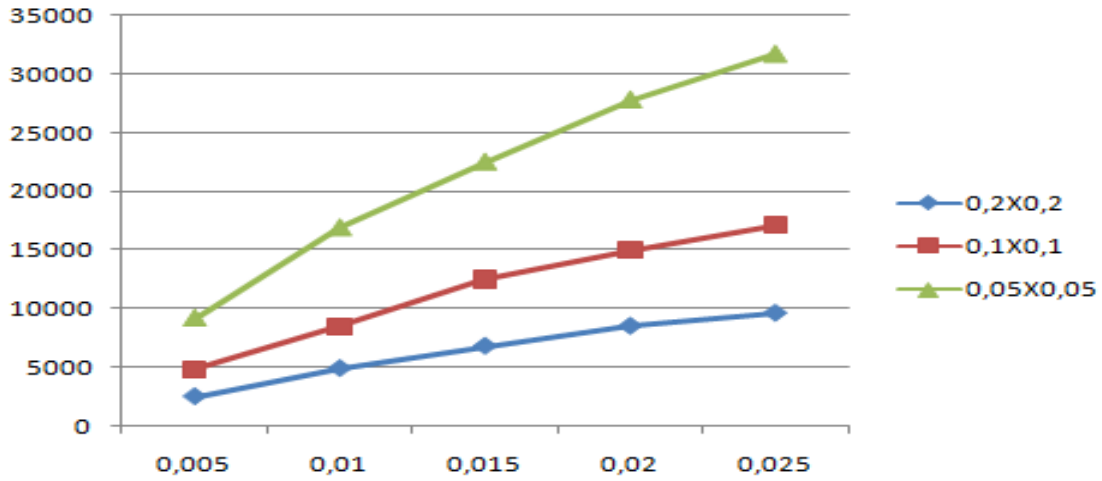


Рис. 6. Производительность относительно $\bar{v}$

В этом эксперименте мы хотим определить производительность индекса методов при различных начальных распределениях и типах движения. Мы подсчитываем количество обновлений базы данных и дисковых страниц, используемых для хранения всех данных. При этом количество обновлений базы данных состоит из двух частей: запроса обновления базы данных, генерируемого LP; операции обновления (в процессе объединения и разделения блока). На рис. 7 и 8 представлены результаты.

Некоторые факты, которые можно наблюдать на данных рисунках:

- Метод RT обновляет расположение всех объектов после каждого периода времени. Таким образом, число обновлений базы данных совпадает с количеством объектов. Остальные четыре метода имеют лучший показатель, чем метод RT.

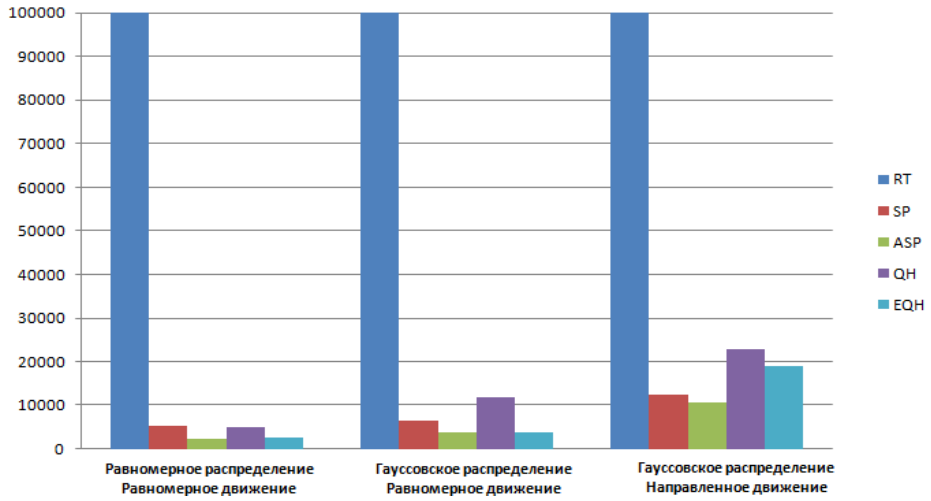


Рис. 7. Количество обновлений базы данных

- Когда начальное распределение объектов равномерно, производительность QH совпадает с SP. Это нормально, так как при равномерном распределении объектов

Quad-дерево очень сбалансировано и его листовые узлы имеют одинаковую высоту. Таким образом, конечные узлы (блоки) имеют равный размер. Вся структура превращается в одноразмерное пространственное разбиение по методу SP. По той же причине производительность ASP и EQH одинакова.

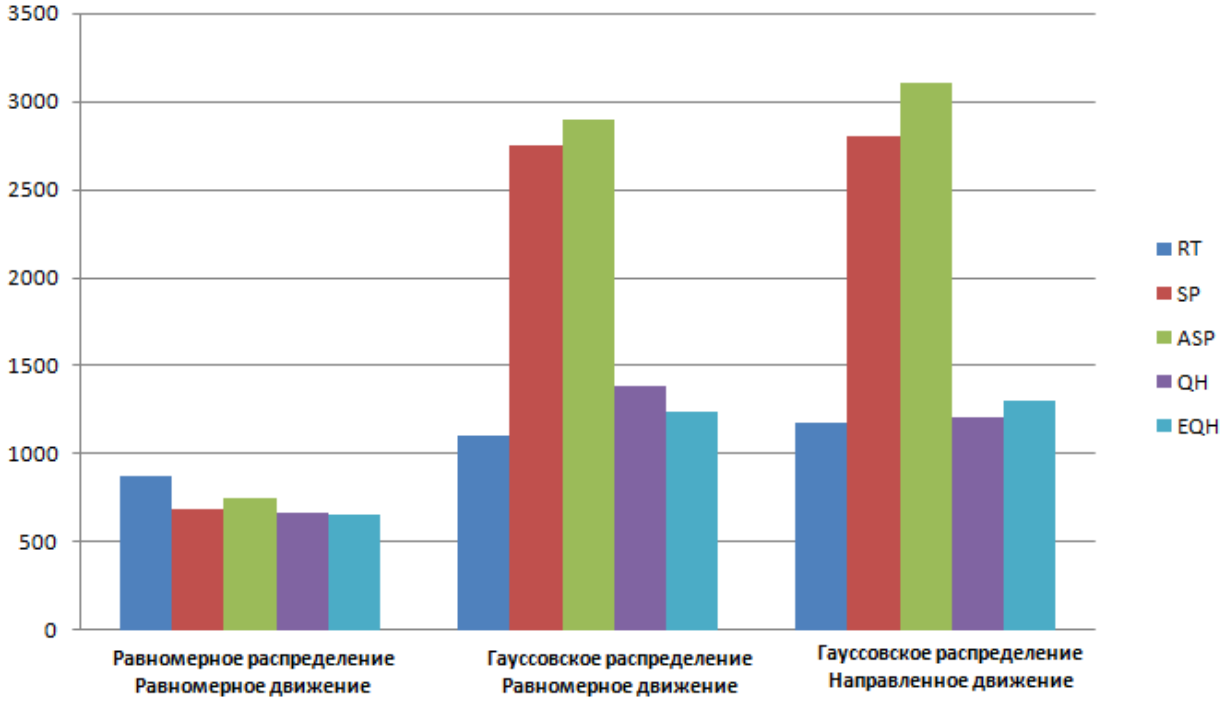


Рис. 8. Количество использованных дисковых страниц

- Общее обновление базы данных в ASP-методе составляет около 60 - 70% от SP-метода. Усовершенствование очень велико.
- Когда объекты находятся в гауссовском распределении, количество обновлений базы данных QH больше, чем в SP. Объекты в этих блоках, скорее всего, пересекут границу блока и сгенерируют обновление в базе данных. Обновление блока генерирует дополнительное обновление базы данных (хотя и не слишком значительное в этом эксперименте). По той же причине EQH генерирует больше обновлений базы данных, чем ASP.
- Выгода QH и EQH – их эффективность хранения. На рис. 8 мы можем обнаружить, что при равномерном распределении дисковая страница, используемая во всех четырех методах, почти одинакова. Однако когда начальное распределение искажено, QH и EQH используют около половины дисковой страницы от SP и ASP, потому что в QH и EQH есть часть управления блоками, которая объединяет блоки с небольшим количеством объектов.
- Если тип движения собирается быть направленным, система вызывает большое количество операций объединения и разделения в QH и EQH. Это производит дополнительные обновления в базе данных. Однако эффективность хранения по-прежнему сохраняется.

Эксперимент 3. Сравнение по производительности запросов.

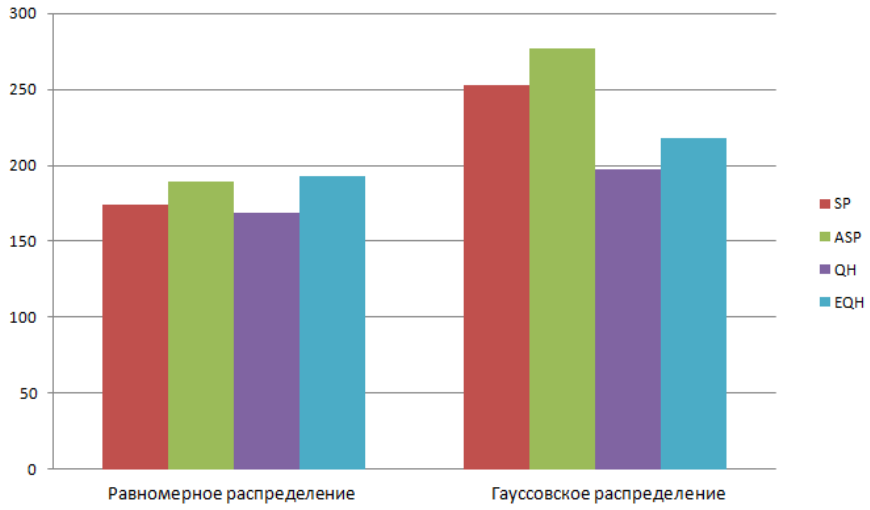


Рис. 9. Количество проверенных дисковых страниц

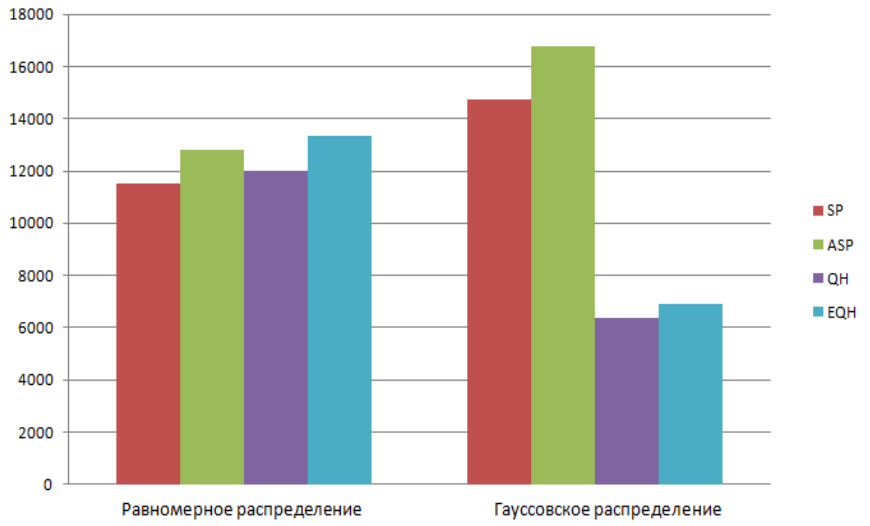


Рис. 10. Количество проверенных объектов

В этом эксперименте нам необходимо проверить производительность запросов всех наших четырех методов. Мы получаем два набора данных в эксперименте: доступ к дисковым страницам и объекты, проверенные в LP. Первый набор данных используется для того, чтобы оценить производительность метода, а второй – для оценки стоимости связи.

При равномерном распределении, как в исследовании индекса, производительность SP и QH почти совпадает. Также почти равны ASP и EQH. Тем не менее в распределении Гаусса методы QH и EQH используют приблизительно на 30% меньше дисковых страниц (рис. 9), а стоимость связи составляет менее половины того, что наблюдается в методах SP и ASP (рис. 10).

EQH-метод показывает немного худшие результаты, чем QH. Это связано с тем, что после расширения узла дерева индекса возможность пересечения каждого узла с площадью запроса немного увеличивается. Это увеличивает стоимость запроса. Однако различие невелико.

3. Выводы и выбор метода

Первый вывод состоит в том, что необходимо решить, нужно ли нам расширение блоков как использование ASP вместо SP или EQH вместо QH. Наш ответ будет положительным. Преимущество в данном случае огромно: фильтруется около 30 - 40% запросов на обновление базы данных. Стоимость запроса невелика, хотя он выполняется немного медленнее. Идеальный размер области наложения должен быть такой же, как и $\bar{v}$, который в большинстве случаев является очень маленьким.

Зачем нам необходимо динамически изменять структуру блока? Это зависит от многих факторов. Статическую структуру блока (в методах SP и ASP) можно легко реализовать. Каждому LP нужно только помнить хеш-функцию, которая была задана вначале. Кроме того, структура хорошо работает в однородных случаях. Динамическая структура (в методах QH и

EQH) является более сложной в реализации. Дополнительные расходы динамических структур блока включают в себя: древовидную структуру, которая хранится в памяти, дополнительные затраты на связь между LP-слоем и базой данных и т.д. Тем не менее они достаточно эффективны.

Наш выбор заключается в использовании метода ASP, если распределение объектов не сильно искажено и нагрузка запроса не является высокой. В противном случае предполагается использование метода EQH.

СПИСОК ЛИТЕРАТУРЫ

1. Гулаков, В.К. Введение в хеширование: монография / В.К.Гулаков, К.В.Гулаков. – Брянск: БГТУ, 2011. – 129 с.
2. Гулаков, В.К. Пространственно-временные структуры данных / В.К.Гулаков, Е.О.Трубаков, А.О.Трубаков. – Брянск: БГТУ, 2013. – 214 с.
3. Nascimento, M.A. Evaluation of Access Structures for Discretely Moving Points Intl. Workshop on Spatio-Temporal Database Management / M.A.Nascimento, J.R.Silva, Y.Theodoridi. - Edinburgh, UK, 2002.
4. Song, Z. Hashing moving objects / Z.Song, N.Roussopoulos; Department of Computer Science University of Maryland College Park. - 2004.

1. Gulakov, V.K. *Introduction into Hashing:* monograph / V.K.Gulakov, K.V.Gulakov. – Bryansk: BSTU, 2011. – pp. 129.
2. Gulakov, V.K. *Spatial-Temporal Structure of Data* / V.K.Gulakov, E.O.Trubakov, A.O.Trubakov. – Bryansk: BSTU, 2013. – pp. 214.
3. Nascimento, M.A. Evaluation of Access Structures for Discretely Moving Points Intl. Workshop on Spatio-Temporal Database Management / M.A.Nascimento, J.R.Silva, Y.Theodoridi. - Edinburgh, UK, 2002.
4. Song, Z. Hashing moving objects / Z.Song, N.Roussopoulos; Department of Computer Science University of Maryland College Park. - 2004.

Статья поступила в редколлегию 24.05.2016.
Рецензент: к.т..н., профессор Брянского государственного технического университета Подвесовский А.Г.

Сведения об авторах:

Гулаков Василий Константинович, к.т.н., профессор кафедры «Информатика и программное обеспечение» Брянского государственного технического университета, e-mail: gvk10@yandex.ru.
Гулаков Константин Васильевич, к.т.н., доцент кафедры «Информатика и программное обеспечение» Брянского государственного технического университета, e-mail: gulakov32@yandex.ru.
Сковородников Юрий Александрович, аспирант кафедры «Информатика и программное обеспечение» Брянского государственного технического университета, e-mail:yurbelios@gmail.com.

Gulakov Vasily Konstantinovich, Can. Eng., Prof. of the Dep. “Informatics and Software”, Bryansk State Technical University, e-mail: gvk10@yandex.ru.
Gulakov Konstantin Vasilievich, Can. Eng., Assistant Prof of the Dep. “Informatics and Software”, Bryansk State Technical University, e-mail: gulakov32@yandex.ru.
Skovorodnikov Yury Alexandrovich, Past graduate student of the Dep. “Informatics and Software”, Bryansk State Technical University, e-mail:yurbelios@gmail.com.