ПРИМЕНЕНИЕ МЕТОДОВ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ К ЗАПРОСАМ В БАЗАХ ДАННЫХ

Л.М. Веретенников, Е.В. Станевич, А.В. Табаков

Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)

Аннотация. В данной статье рассматривается проблема производительности больших запросов в базах данных. Кратко рассмотрены способы оптимизации базы данных с их достоинствами и недостатками. Также приводится алгоритм и пример поиска подзапросов, которые могут выполняться параллельно, с применением теории графов и методов параллельных вычислений.

Ключевые слова: параллельный алгоритм, граф, параллельные запросы, параллельные вычисления, база данных, оптимизация.

Введение

На сегодняшний день использование баз данных является неотъемлемой частью функционирования предприятий и организаций. Поскольку растёт объём данных и требования к данным, поэтому проблема сокращения времени обработки данных продолжает оставаться актуальной. Приём и получение данных из базы является одним из узких мест. Таким образом оптимизация запросов позволяет существенно повысить производительность.

Существует множество методов оптимизации для баз данных и запросов: оптимизация сложных запросов, увеличение аппаратной производительности и другие.

Основные способы оптимизации баз данных:

1. Индексация таблиц. Таблица, не имеющая индексов, представляет собой беспорядочный набор строк. Например, для поиска нужной записи по id, необходимо проверить искомое значение на совпадение со всеми строками в таблице. Если таблица проиндексирована, то нет необходимости проверять на совпадения все строки в таблице. Сканирование по индексу возвращает одну строку и так как индексы отсортированы по возрастанию, то при нахождении искомого значения дальнейшее сканирование можно завершить. Индексы значительно ускоряют поиск данных, но также замедляют операции изменения, добавления и удаления, поскольку при каждом изменении необходимо изменять индекс.

2. Использование пользовательских переменных. Пользовательские переменные позволяют сохранять промежуточные результаты и использовать их при дальнейшем выполнении запроса, что может значительно ускорить время выполнения запроса.

Кроме приведенных выше способов оптимизации, существует множество и других способов. Также не все способы играют положительную роль. Каждая база данных требует индивидуальный подход к оптимизации, поэтому проблема оптимизации запросов продолжает оставаться актуальной.

Распараллеливание запросов

Выполнение большого запроса, как правило, выполняется на одном ядре, что может занимать значительное количество времени. Поэтому один из способов распараллеливания запросов – это разбиение одного большого запроса на множество подзапросов, выполняемых параллельно. При этом запросы для параллельного выполнения должны быть независимы друг от друга (т.е. результат одного запроса не должен влиять на формирование другого). Количество потоков обычно определяется количеством ядер.

Ниже приведен алгоритм распараллеливания запросов [1, 2, 3]:

Шаг 1. Создать множества с выходными вершинами. Поместим в множество все выходные вершины из списка.

Шаг 2. Создать множества с входными вершинами. Поместим в множество все входные вершины из списка.

Шаг 3. Удалить вершины из выходного множества все вершины, которые встречаются во входном множестве.

Шаг 4. Если выходное множество содержит хоть одну вершину, то добавить оставшиеся вершины в новую группу расписания, в противном случае – конец алгоритма (решений нет).

Шаг 5. Удалить из списка рёбра, у которых входное значение совпадает со значением из расписания.

Шаг 6. Перейти к шагу 1, пока существуют вершины, которые можно вычеркнуть. В противном случае – переходим к шагу 6.

Шаг 7. Добавить в новую группу расписания вершины, которые остались в выходном множестве.

Шаг 7. Добавить в новую группу расписания вершины, которые остались во входном множестве.

Пример работы метода

Пусть есть запрос вида:

```
SELECT * FROM worker WHERE worker.skill IN
   (SELECT name FROM skills WHERE spec_id =
      (SELECT id FROM specializations WHERE specializations.id IN
         (SELECT id FROM faculties WHERE faculties.university_id IN
            (SELECT id FROM universities WHERE universities.city_id =
               (SELECT id FROM cities WHERE cities.name = 'Saint Petersburg')))
         AND specializations.name = 'IT'))
   AND (worker.departament_id =
        (SELECT id FROM departaments WHERE departaments.id IN
           (SELECT departament_id FROM organizations WHERE (organizations.city_id =
              (SELECT id FROM cities WHERE cities.name = 'Saint Petersburg'))
           AND (departaments.spec_id =
                (SELECT id FROM specializations WHERE specializations.name = 'IT'))))));
```

Рис. 1. Пример подзапроса

Представим подзапросы в виде графа зависимостей (вершины пронумерованы в произвольном порядке), где каждая вершина – это отдельно выполняемый подзапрос.

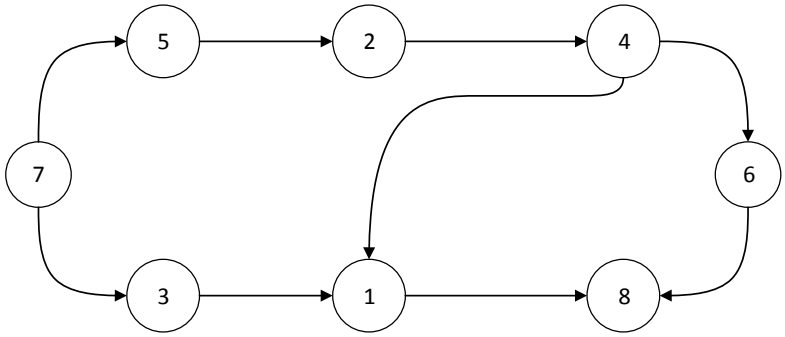

Рис. 2. Исходный граф

Представим всё рёбра графа в виде списка («выход» – «вход»).

{(3 - 1), (4 - 1), (5 - 2), (7 - 3), (2 - 4), (7 - 5), (4 - 6), (1 - 8), (6 – 8)}

Создадим множество с выходными вершинами: {1, 2, 3, 4, 5, 6, 7}.

Создадим множество с входными вершинами: {1, 2, 3, 4, 5, 6, 8}.

Удалим вершины из выходного множества все вершины, которые встречаются во входном множестве. Получим: {7}.

Добавим оставшиеся вершины в новую группу расписания: [{7}].

Удалить из списка рёбра, у которых входное значение совпадает со значением из расписания. Получим: {(3 - 1), (4 - 1), (5 - 2), (2 - 4), (4 - 6), (1 - 8), (6 - 8)}.

Снова создадим множество с выходными вершинами: {1, 2, 3, 4, 5, 6}.

Создадим множество с входными вершинами: {1, 2, 4, 6, 8}.

Удалим вершины из выходного множества все вершины, которые встречаются во входном множестве. Получим: {3, 5}.

Добавим оставшиеся вершины в новую группу расписания: [{7}, {3, 5}].

Удалить из списка рёбра, у которых входное значение совпадает со значением из расписания. Получим: {(4 - 1), (2 - 4), (4 - 6), (1 - 8), (6 - 8)}.

Снова создадим множество с выходными вершинами: {1, 2, 4, 6}.

Создадим множество с входными вершинами: {1, 4, 6, 8}.

Удалим вершины из выходного множества все вершины, которые встречаются во входном множестве. Получим: {2}.

Добавим оставшиеся вершины в новую группу расписания: [{7}, {3, 5}, {2}].

Удалить из списка рёбра, у которых входное значение совпадает со значением из расписания. Получим: {(4 - 1), (4 - 6), (1 - 8), (6 - 8)}.

Снова создадим множество с выходными вершинами: {1, 6, 4}.

Создадим множество с входными вершинами: {1, 6, 8}.

Удалим вершины из выходного множества все вершины, которые встречаются во входном множестве. Получим: {4}.

Добавим оставшиеся вершины в новую группу расписания: [{7}, {3, 5}, {2}, {4}].

Удалить из списка рёбра, у которых входное значение совпадает со значением из расписания. Получим: {(1 - 8), (6 - 8)}.

Добавить в новую группу расписания вершины, которые остались в выходном множестве: [{7}, {3, 5}, {2}, {4}, {1, 6}].

Добавить в новую группу расписания вершины, которые остались в выходном множестве: [{7}, {3, 5}, {2}, {4}, {1, 6}, {8}].

Получим итоговое расписание:

T1: {7}, T2: {3, 5}, T3: {2}, T4: {4}, T5: {1, 6}, T6: {8}

В результате алгоритм отрабатывает за 6 единиц времени, что на 2 единиц меньше, если бы все подзапросы выполнялись последовательно.

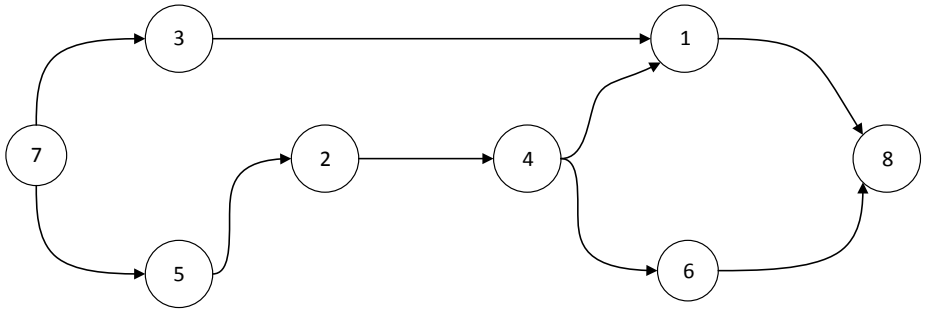

Рис. 3. Граф в параллельной форме

Заключение

В статье были кратко затронуты различные способы оптимизации базы данных, такие как индексация таблиц, использование пользовательских переменных и т.д. Также приведены их достоинства и недостатки.

Описанный метод оптимизации основанный на теории графов, показывает возможность оптимизации больших запросов.

Разбиение одного большого запроса на подзапросы позволяет сократить время их обработки. К таким запросам могут быть применены методы параллельных вычислений с применением теории графов, что позволяет максимально задействовать аппаратные ресурсы и снизить требования к аппаратуре.

Список литературы

1. Шичкина Ю.А. Применение списков следования для оптимизации информационного графа по высоте. Системы. Методы. Технологии. № 9. 2011. С. 68-77.
2. Шичкина Ю.А. Взвешенные графы в параллельных вычислениях. Современные наукоемкие технологии. 2009. № 11. С. 93-97.
3. Шичкина Ю.А. Комбинированный метод многопараметрической оптимизации взвешенного информационного графа. Системы. Методы. Технологии. № 7. 2010. С. 76-82.
4. Моргунов Е.П. Язык SQL. Базовый курс
5. Гроff. Дж.Р., Вайнберг П.Н., Оппель Э.Дж. SQL. Полное руководство

ПОТОКОБЕЗОПАСНЫЕ СТРУКТУРЫ ДАННЫХ С ОСЛАБЛЕННОЙ СЕМАНТИКОЙ ВЫПОЛНЕНИЯ ОПЕРАЦИЙ

А. В. Табаков, Л. М. Веретенников

Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)

Аннотация. Структуры с ослабленной семантикой выполнения операций представляют большой интерес в области многопоточных программ. Существующие на данный момент потокобезопасные структуры данных (на базе блокировок или без использования блокировок) имеют узкие места, так как всем потокам приходится вынужденно ожидать своей очереди обращения к данным. Потокобезопасные структуры с ослабленной семантикой доступа позволяют обращаться с данными без их блокировки, жертвуя точностью, но выигрывая в производительности.

Ключевые слова: структура данных, ослабленная семантика доступа, потокобезопасные структуры

Многоядерные вычислительные системы сейчас представляют особенный интерес, так как разработчики процессоров стали уделять большее внимание количеству физических и логических ядер, нежели частоте одно ядра. Таким образом возникает проблема одновременного (параллельного) доступа к общим данным.

Традиционные структуры данных: массивы, списки, деревья и другие, не приспособлены для работы в многопоточной системе. Для работы с общей структурой в разных потоках возникает необходимость в синхронизации доступа к данным, с использованием блокировок, что является серьёзной проблемой при распараллеливании программ, как с точки зрения времени выполнения, так и с точки зрения консистентности данных. Синхронизация доступа, к данным структуры, необходима, так как разные потоки могут одновременной обращаться к одной и той же структуре и соответственно изменять либо читать данные из неё. Параллельное выполнение этих операций невозможно, из-за возникающей гонки данных (data race), когда операция со структурой начинается до того, как данные были записаны или изменены.

Структуры с ослабленной семантикой доступа решают поставленную проблему следующим образом: создаётся количество структур большее или равное количеству потоков, при изменении или чтении данных выбирается случайный поток, который в данный момент не выполняет действий со структурами (не заблокирован), и выполняет действие.

В статье [1], предлагается улучшение структуры SkipList [2] (описание данной структуры выходит за рамки данной статьи), под названием SprayList [1]. В отличие от SkipList'a, SprayList предполагает не линейный поиск сверху вниз и из начала в конец, переходя по одной ссылке связанного списка, а изначальное перемещение случайным образом по вертикальным уровням и далее уже поиск элемента списка со случайным смещением по горизонтальному уровню. Если поиск не дал результатов или элемент оказался заблокированным другим потоком, алгоритм начинается с начала. После нахождения нужного элемента операции со списком выполняются также, как и в SkipList.

В работе [3] был разработан алгоритм операции удаления элемента с минимальным ключом из очереди с приоритетами и выдачи его в качестве результата функции. Суть данного подхода в том, что на каждый поток необходимо иметь больше чем одну очередь. Операция вставка элемента осуществляется в случайную незаблокированную очередь. Операция удаление элемента с минимальным ключом осуществляется следующим образом: выбираются две случайные очереди, у них находятся элементы с минимальными ключами и сравниваются между собой, после нахождения минимального, этот элемент удаляется из очереди с наименьшим из найденных ключом. Как можно было заметить этот элемент не всегда будет минимальным из вставленных, однако принимается, что он довольно близок к минимальному и данной погрешностью можно пренебречь.

В статье [4] была разработана k-LSM (Log-Structured Merge tree) структура, входящая в подкласс структур с ослабленной семантикой выполнения операций. В качестве базовой структуры в [4] используется LSM дерево. Оно представляет собой указатели на сортированные массивы, называемые блоками, где каждый блок находится на определённом уровне L дерева и может содержать N элементов ($2^{L-1} < N \leq 2^L$). Структура k-LSM состоит из двух других: общей k-LSM и распределённой LSM очередей с приоритетом. Общая k-LSM очередь представляет собой массив указателей на очереди с приоритетами, сортированные по уровню. Все потоки могут обращаться к данной структуре по единому указателю. У общей k-LSM структуры существует два узких места: а) операция вставки элемента, однако обычно используется массовая вставка, таким образом сокращается количество обращений к данной структуре, но возрастает средний размер блока и уменьшается количество операций слияния, б) операция удаления минимального элемента, также вызывает блокировку, однако в данной статье предполагается, использование ослабленной семантики доступа, таким образом из общей

k-LSM структуры удаляется не элемент с самым минимальным ключом, а минимальный из k+1 случайно выбранных ключей. Распределённая LSM разрабатывалась на основе идеи work-stealing [5], каждый поток имеет собственную очередь приоритетов и работает исключительно с ней, но, если очередь пуста, а требуется операция отличительная от операции вставки, начинается попытка доступа к чужим очередям с приоритетами и, если они не заблокированы, выполняется операция с ними. Таким образом, объединив общую k-LSM и распределённую LSM структуры, была получена k-LSM структура. При операции вставки, поток сохраняет элемент в собственной распределённой LSM структуре, если размер данной структуры превышает заданный, то данная распределённая LSM осуществляет слияние с общей k-LSM структурой. При операции удаления используется поиск наименьшего ключа в собственной и общей LSM структурах и оба элемента удаляются, если данные структуры пусты, то осуществляется поиск среди структур у других потоков.

Список литературы

1. Д. Алистер [D. Alistarh], Дж. Копински [J. Kopinsky], Дж. Ли [J. Li]], Н. Шэвит [N. Shavit] The SprayList: A Scalable Relaxed Priority Queue [Электронный ресурс] //URL: http://www.mit.edu/~jerryzli/SprayList-CR.pdf
2. В. Пуг [W. Pugh] Skip Lists: A Probabilistic Alternative to Balanced Trees [Электронный ресурс] //URL: http://epaperpress.com/sortsearch/download/skiplist.pdf
3. Х. Рихани [H. Rihani], П. Сандерс [P. Sanders], Р. Дементьев [R. Dementiev] MultiQueues: Simpler, Faster, and Better Relaxed Concurrent Priority Queues [Электронный ресурс] //URL: https://arxiv.org/pdf/1411.1209.pdf
4. М. Виммер [M. Wimmer], Дж. Грабер [J. Gruber], Дж. Л. Траф [J. L. Traf], Ф. Тсигас [P. Tsigas] The Lock-free k-LSM Relaxed Priority Queue [Электронный ресурс] //URL: https://arxiv.org/pdf/1411.1209.pdf
5. Р. Блюмо [R. Blumofe], Ч. Лейзерсон [C. Leiserson] Scheduling Multithreaded Computations by Work Stealing [Электронный ресурс] //URL: http://supertech.csail.mit.edu/papers/steal.pdf

УПРАВЛЕНИЕ ВОСПРИЯТИЕМ ИНТЕЛЛЕКТУАЛЬНОГО АГЕНТА В СРЕДЕ ВИРТУАЛЬНОГО ФУТБОЛА

А. В. Табаков

Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)

Аннотация. Рассматривается проблема управления восприятием интеллектуального агента, функционирующего в открытой динамической многоагентной среде, и подход к ее решению в рамках концепции опережающего итеративного планирования. Представлен конкретный пример алгоритма управления восприятием агента-футболиста, функционирующего в среде виртуального футбола.

Ключевые слова: интеллектуальные агенты, многоагентные системы, управление восприятием

Интеллектуальные агенты (ИА) – быстроразвивающийся класс систем искусственного интеллекта (ИИ), характеризующихся способностью к автономному целенаправленному поведению в открытых динамических многоагентных средах [1].

Подсистема восприятия (ПВсп) ИА состоит из множества сенсоров, посредством которых агент получает информацию о состоянии внешней среды и, в некоторых случаях, о состоянии своих внутренних ресурсов. Конкретный состав и характеристики сенсоров ИА определяются конкретным типом агента и его целевым назначением. Одной из важных особенностей ПВсп ИА является ограниченные возможности восприятия, т.е. невозможность получения в каждом сенсорном такте исчерпывающе полной и точной информации о состоянии внешней среды. Это обуславливает необходимость для агента управлять соб-

ственной ПВсп (множеством сенсоров) в интересах наиболее эффективной организации восприятия.

В [1-3] предложена концепция и разработана формальная модель опережающего итеративного планирования (ОИП) как основа построения ИА реального времени, функционирующих в открытых динамических многоагентных средах.

В соответствии с ОИП в каждом сенсорном такте (интервале обновления сенсорной информации) ПВсп ИА должна обеспечивать агента информацией для решения двух задач: а) реализации текущего действия и б) оценки и прогнозирования ситуации в интересах планирования (выбора) следующего действия.

Для успешного завершения текущего действия агенту необходимо иметь актуальную информацию только об объектах внешнего мира, способных повлиять на исход данного действия. Такие объекты могут быть статическими (состояние которых можно считать неизменным на интервале реализации текущего действия), динамическими нецеленаправленными (меняющими состояние под действием известных физических сил) и целенаправленными (агентами).

Оценка и прогнозирование ситуации в интересах планирования следующего действия предполагает учет более широкого контекста, т.е. сбора информации о всех объектах, потенциально способных повлиять на выбор следующего действия.

В данном докладе проблема управления восприятием рассматривается на примере агента-футболиста, функционирующего в многоагентной среде виртуального футбола. Визуальный сенсор такого агента имеет 6 режимов работы, отличающихся шириной взгляда (3 значения) и качеством (точностью) получаемой информации (2 значения). Кроме того, агент может в каждом такте произвольно менять направление взгляда (в пределах заданных ограничений на угол поворота головы относительно направления тела).

Алгоритмы управления восприятием определяются ситуационным контекстом, текущей ролью агента и в общем случае варьируются в широких пределах. Например, для агента, пробросившего себе мяч на ход, текущим действием является «догнать мяч». Агент должен сделать это за заданное время (число тактов) и при этом быть первым на мяче. Для реализации этого действия необходимо отслеживать движение мяча (подверженное возмущениям), а также поведение других агентов, потенциально способных добежать до мяча первыми.

Вместе с тем, для выбора следующего действия агенту необходима информация обо всех агентах своей и противостоящей команды, способных влиять на любой из возможных вариантов очередного действия агента. Множество таких вариантов, в свою очередь, определяется текущим ситуационным контекстом. Для случая ведения мяча в центре поля обобщенный алгоритм управления восприятием включает следующие шаги:

1. Расчёт числа N тактов моделирования до момента окончания текущего действия.
2. Расчёт числа сенсорных тактов за N тактов моделирования при номинальном режиме восприятия.
3. Расчёт максимально возможного угла обзора (с учетом ограничений -120°÷120° относительно положения тела) и количества тактов T на получение всех сенсорных сообщений от сервера в данном максимальном угле обзора при нормальном режиме восприятия.
4. Расчёт числа K тактов, в которых мяч не попадает в видимую область агента.
5. Оценка числа M тактов, в которых мяч может не попадать в видимую агентом область при гарантии его достижения за заданное число тактов (на основе данных, полученных в пп. 3 и 4).

6. Получение сенсорной информации в соответствии с углами обзора, рассчитанными в п. 3 и скорректированными в п. 5; вычисление количества тактов для оценки принятой информации.

7. Оценка сенсорной информации, полученной в п. 6, запись углов обзора с оценкой полезности выше заданного значения; вычисление числа оставшихся тактов моделирования до достижения агентом мяча.

8. Переключение сенсорного режима в узкий угол обзора с высоким качеством принимаемой информации с использованием только «полезных» углов обзора, относительно положения тела, полученных в ходе оценки на шаге 7.

В настоящее время ведется работа по программной реализации и экспериментальному исследованию разработанных алгоритмов. Вместе с тем, важным направлением дальнейших исследований является разработка алгоритмов управления восприятием для других ролей (партнеров владеющего мячом игрока, игроков противостоящей команды на разных позициях) и более широкого класса ситуаций.

Список литературы

1. Пантелеев М.Г., Пузанков Д.В. Интеллектуальные агенты и многоагентные системы. – СПб: Изд-во СПбГЭТУ «ЛЭТИ», 2015, 216 с.

2. Пантелеев М.Г. Формальная модель опережающего итеративного планирования действий интеллектуальных агентов реального времени // Труды 14-й нац. конф. по искусственному интеллекту с международным участием КИИ-2014. – Казань: Физматлит, 2014. Т. 1. С. 323–334.

3. Пантелеев М.Г. Концепция построения интеллектуальных агентов реального времени на основе модели опережающего итеративного планирования// Труды 13-ой нац. конф. по искусственному интеллекту с международным участием КИИ-2012. – Белгород: Изд-во БГТУ, 2012. Т 3. С. 25–33.