ОРГАНИЗАЦИЯ АСИНХРОННОГО КОНВЕЙЕРА И КУСТОВОГО ПАРАЛЛЕЛИЗМА В СУБД ДЛЯ КЛАСТЕРНЫХ СИСТЕМ НА МНОГОЯДЕРНЫХ ПРОЦЕССОРАХ

А.А. Медведев

Введение

Межоперационный параллелизм в СУБД предполагает параллельное выполнение реляционных операций, принадлежащих одному и тому же плану запроса. Данный вид параллелизма может быть реализован либо в форме горизонтального (кустового) параллелизма, либо в форме вертикального (конвейерного) параллелизма. Горизонтальный параллелизм предполагает параллельное выполнение независимых поддеревьев дерева, представляющего план запроса. Вертикальный параллелизм предполагает организацию параллельного выполнения различных операций плана запроса на базе механизма конвейеризации. Традиционный подход к организации такого конвейерного параллелизма заключается в использовании абстракции итератора для реализации операций в дереве запроса. Подобный подход получил название синхронного конвейера. Основным недостатком синхронного конвейера является блокирующий характер операций конвейерной обработки данных. Это означает, что если некоторая операция задерживается при обработке данных, то она блокирует работу всего конвейера. Таким образом, скорость обработки запроса будет равна скорости самой медленной операции. Для преодоления этого недостатка может быть использован асинхронный конвейер. Асинхронный конвейер организуется таким образом, что поставщик и потребитель работают независимо друг от друга, а данные между ними передаются через некоторый буфер обмена.

Потоковая модель выполнения запроса

Для реализации кустового и конвейерного параллелизма предлагается потоковая модель выполнения запроса [1]. В рамках данной модели запрос представляется в виде отдельного процесса. Каждый процесс имеет одну корневую нить и некоторое количество вычислительных нитей, ассоциированных с соответствующими операциями физической алгебры. Будем называть корневую нить менеджером нитей. Менеджер нитей может явно порождать произвольное количество вычислительных нитей, которые организуются в иерархию нитей. Иерархия нитей представлена на Рис. 1.

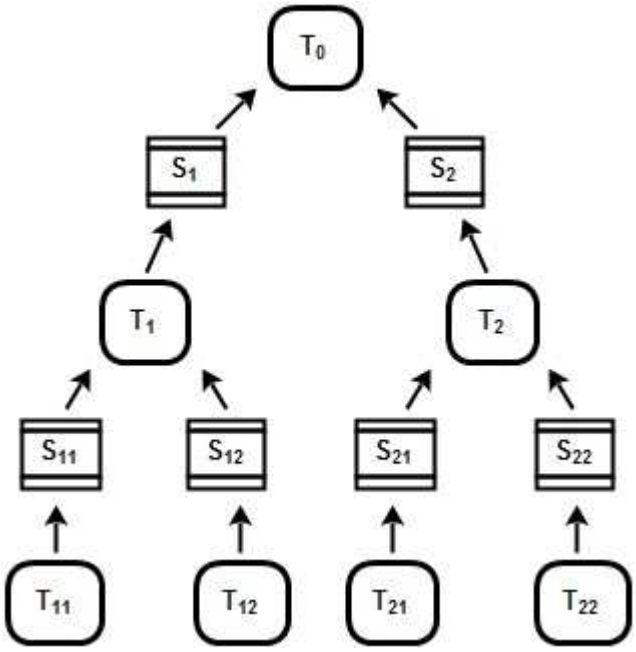

Рис. 1. Иерархия нитей

Потоковая модель выполнения запроса основана на парадигме «производитель-потребитель». В рамках данной парадигмы каждая нить «производит» некоторые данные определенной структуры для вышестоящей в иерархии нити. При этом эта же нить «потребляет» некоторые объекты данных определенной структуры, поставляемые нижестоящими в иерархии нитями. Для организации процесса поставки-потребления с каждой нитью связывается буфер вывода конечной длины, организуемый как очередь (FIFO) и называемый складом.

Операционная поддержка потоковой модели

Операционная поддержка потоковой модели представляет собой прототип ядра СУБД, называемый Flow-Stock Engine (FSE). Исполнитель запроса прототипа FSE реализует механизмы асинхронного конвейера и

кустового параллелизма. На Рис. 2 представлена диаграмма деятельности, описывающая обработку запроса в прототипе FSE.

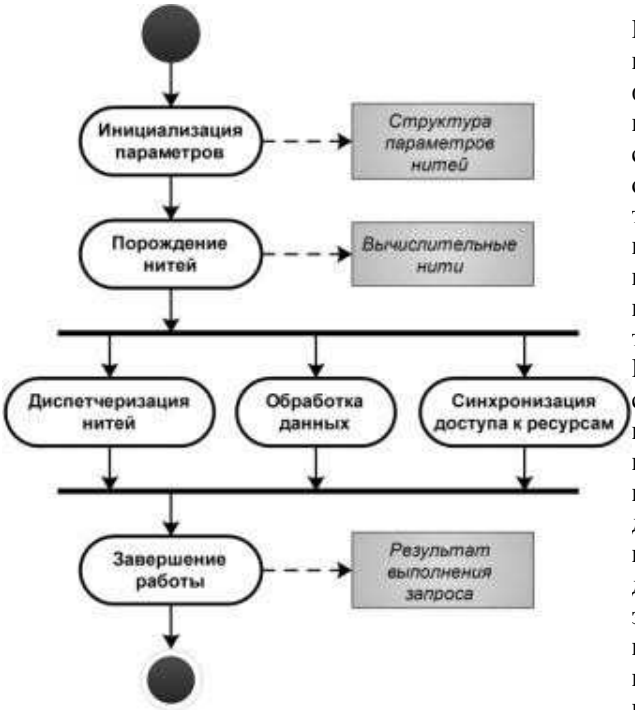

Рис. 2. Обработка запроса

Обработка запроса проходит в четыре этапа. Первый этап обработки запроса – инициализация параметров нитей. На данном этапе генератор формирует дерево запроса. На втором этапе исполнитель производит порождение вычислительных нитей в соответствии с параметрами. В рамках третьего этапа обработки запроса исполнитель выполняет параллельно три подзадачи. Во-первых, диспетчеризация нитей, необходимая для изменения приоритетов процессорного времени вычислительных нитей. Диспетчеризация необходима для того, чтобы нивелировать разницу трудоемкости различных операций физической алгебры. Во-вторых, обработка данных. В-третьих, синхронизация доступа к ресурсам, являющаяся необходимым условием корректной обработки запроса прототипом FSE. В рамках данной задачи каждая вычислительная нить производит синхронизацию доступа к складу, так как склад одновременно является выходным буфером для нити-производителя и входным – для нити-потребителя. Четвертый этап обработки запроса – завершение работы. На данном этапе исполнитель получает результат выполнения запроса от нити, которая является корнем дерева запроса, производит уничтожение вычислительных нитей и выдает результат обработки запроса.

Реализация прототипа Flow-Stock Engine

Для реализации прототипа FSE был использован язык программирования C++ и библиотека POSIX Threads [2, 3]. Прототип FSE состоит из двух основных подсистем, которые обеспечивают анализ запроса, написанного на языке RQL [4], построение и выполнение дерева запроса. Модульная структура прототипа представлена на Рис. 3.

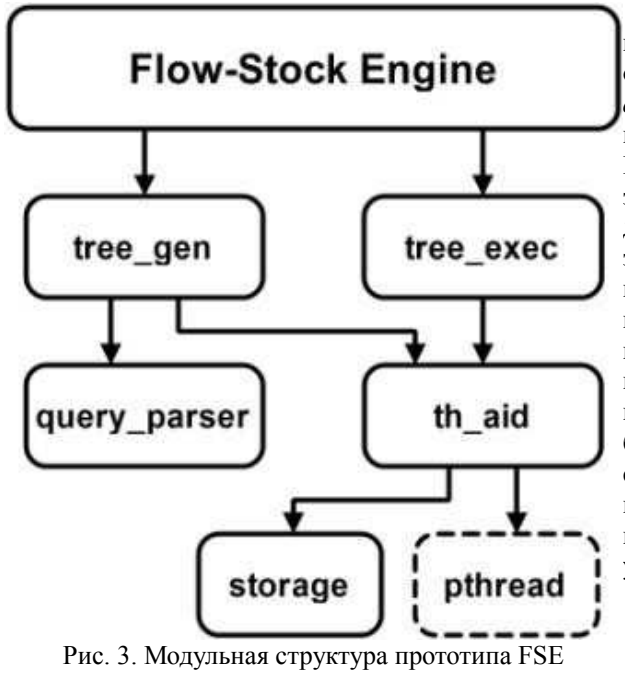

Рис. 3. Модульная структура прототипа FSE

Рассмотрим подробно назначение каждой подсистемы и библиотеки. Подсистема *tree_gen* обеспечивает генерацию дерева запроса. Подсистема *query_parser* обеспечивает распознавание символьной цепочки, которая представляет собой запрос на языке RQL. Подсистема *tree_exec* является исполнителем запросов. Исполнитель производит запуск и диспетчеризацию вычислительных нитей. После завершения работы вычислительных нитей, исполнитель производит вывод результатов запроса. Библиотека *th_aid* предоставляет основные функции и структуры, необходимые для организации потоковой модели выполнения запроса. Библиотека *storage* предоставляет класс «склад». Данный класс содержит реализацию буфера обмена, используемого при передаче данных в соответствии с парадигмой «производитель-потребитель». Библиотека pthread определяет интерфейс прикладного программирования для создания и управления нитями.

Вычислительные эксперименты

В соответствии с концепцией потоковой модели выполнения запроса был проведен ряд вычислительных экспериментов для того, чтобы исследовать эффективность работы прототипа FSE в зависимости от параметров системы. В качестве аппаратной платформы для проведения экспериментов был использован вычислительный узел суперкомпьютера «СКИФ Урал». Вычислительный узел суперкомпьютера «СКИФ Урал» содержит два четырехъядерных процессора Intel Xeon E5472 3.0 GHz [5].

В качестве запросов были использованы мультисоединения, представленные в виде левонаправленных и сбалансированных деревьев. Пусть количество операций JOIN, используемых в запросе, будет равно N, тогда глубина левонаправленного дерева будет равна *N*, а глубина сбалансированного – *log2(N+1)*. Были проведены измерения скорости выполнения запросов и при различных значениях размера склада и количества операций соединения.

Результаты экспериментов при *N=3* и *N=15*, представленные на Рис. 4 и Рис. 5 соответственно, показывают, что при *N=3* ускорение для случаев с левонаправленным и сбалансированным деревьями одинаково. Однако при дальнейшем увеличении *N* до 15 в случае с левонаправленным деревом ускорение выше, но его максимум достигается быстрее, и максимальная скорость выполнения запроса меньше, чем для случая со сбалансированным деревом.

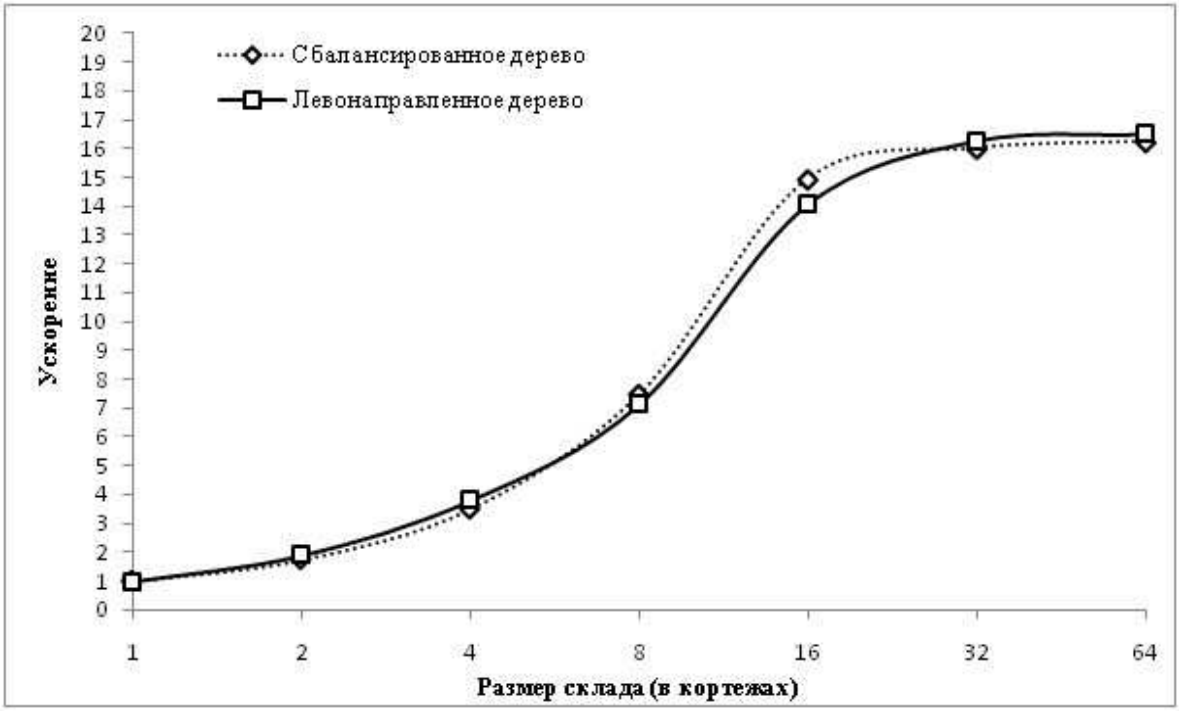

Рис. 4. Зависимость ускорения от размера склада при N=3

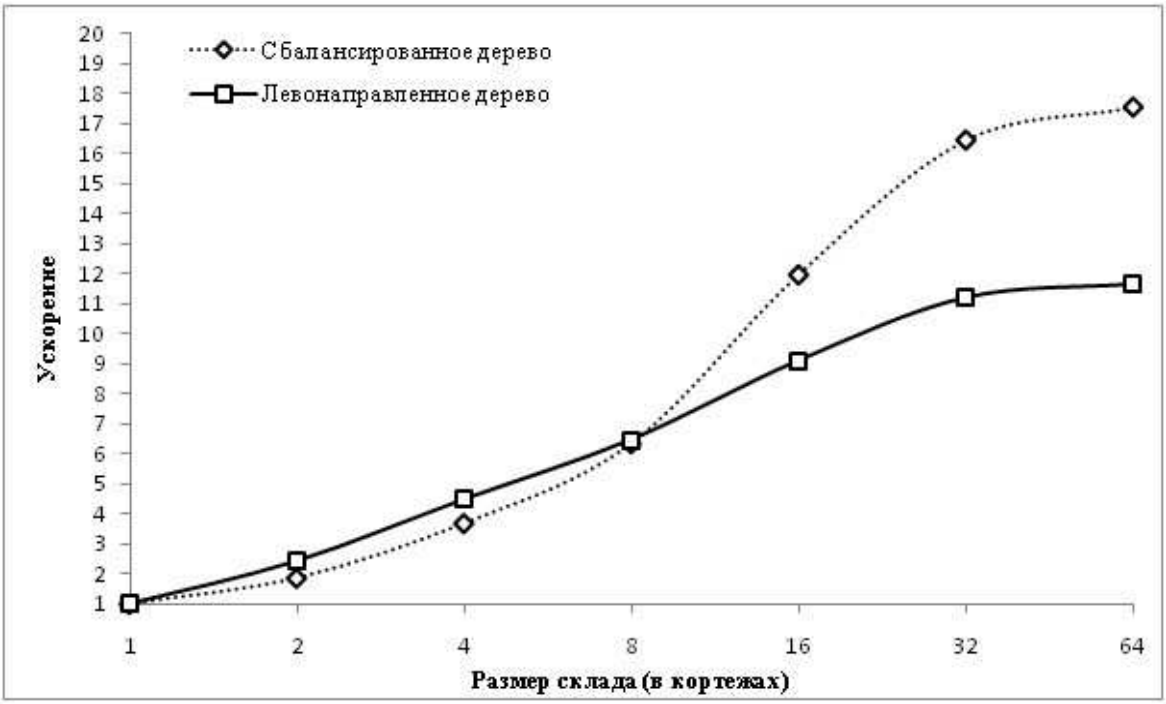

Рис. 5. Зависимость ускорения от размера склада при N=15

При этом максимум ускорения для левонаправленного дерева меньше. Это связано с тем, что левонаправленное дерево хорошо согласуется с конвейерной моделью, но имеет большую глубину. Проведенные эксперименты подтверждают эффективность параллельных алгоритмов, используемых в прототипе FSE.

Заключение

В ходе работы был спроектирован и реализован прототип ядра СУБД, называемый Flow-Stock Engine (FSE), который использует подсистему асинхронного конвейера и кустового параллелизма для параллельной межоперационной обработки запросов. Представлены результаты вычислительных экспериментов, подтверждающих ффективность алгоритмов, используемых в прототипе. Выявлены оптимальные параметры системы для обработки запросов.

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (проект 09-07-00241-а).

ЛИТЕРАТУРА:

1. Соколинский Л.Б. Организация параллельного выполнения запросов в многопроцессорной машине баз данных с иерархической архитектурой // Программирование, 2001. №. 6. С. 13–29.
2. Konstantinos K., Cintra M., Viglas D.S. Multithread query execution on multicore processors. // Proceeding of the VLDB, Lyon, France, 24–28 August, 2009.
3. Parkhurst J., Darringer J., Grundmann B. From single core to multi-core: preparing for a new exponential // ACM international Conference on Computer-Aided Design, San Jose, California, 5–9 November, 2006. ACM, 2006. P. 67–72.
4. Костенецкий П.С., Лепихов А.В., Соколинский Л.Б. Технологии параллельных систем баз данных для иерархических многопроцессорных сред // Автоматика и телемеханика, 2007. № 5. С. 112–125.
5. Вычислительный кластер «СКИФ Урал». URL: http://supercomputer.susu.ru/computers/ckif_ural/ (дата обращения: 10.03.2010).