Параллельная СУБД для кластерных вычислительных систем на базе PostgreSQL

А.В. Лепихов

Южно-Уральский государственный университет

Введение

В связи с взрывным ростом хранимой в электронном виде информации на системы баз данных ложится значительная вычислительная нагрузка. При этом система управления базами данных (СУБД) зачастую является узким местом информационной системы. В связи с этим возникает необходимость увеличения производительности СУБД для обеспечения адекватного времени обработки запросов в условиях непрерывного роста объемов хранимых данных и количества пользователей.

Эффективным решением данной проблемы является использование параллельных СУБД для кластерных вычислительных систем. В настоящее время известен ряд коммерческих параллельных СУБД, например, DB2 Parallel Edition [1], Non Stop SQL [2] и Teradata [3]. Однако высокая цена и алгоритмы обработки запросов, ориентированные на специализированное оборудование ограничивают сферу применения таких СУБД. В данной работе предлагается метод организации параллельной обработки запросов в кластерных вычислительных системах на базе СУБД PostgreSQL.

На сегодняшний день известно несколько проектов, обеспечивающих обработку запросов в кластерных вычислительных системах на базе СУБД PostgreSQL. В рамках научного проекта ParGRES [4] разрабатывается параллельная СУБД, предназначенная для обработки OLAP-запросов. В работе [5] описана разработка компании Hitachi обеспечивающая управление архивными данными, размещаемыми в кластерной вычислительной системе. Однако до настоящего времени нет известных проектов разработки параллельной СУБД для кластерных вычислительных систем на основе свободно распространяемой СУБД.

Архитектура СУБД PostgreSQL

Система управления базами данных PostgreSQL состоит из трех основных компонентов [6] (см. рис. 1).

1. *Libpq* – коммуникационная библиотека, предоставляющая пользователю программный интерфейс для взаимодействия с СУБД.
2. *Postmaster* – процесс, реализующий механизм инициализации транзакции.
3. *Backend* – процесс, реализующий обработку запросов одной транзакции.

Процесс *backend* состоит из трех основных подсистем. Подсистема *compiler* выполняет лексический разбор запроса и формирует его внутреннее представление в виде дерева. Подсистема *rewriter* на основе правил оптимизации запросов и набора эвристик формирует оптимизированный план запроса. Подсистема *executor* (исполнитель запроса) интерпретирует план запроса и формирует результат.

Схему обработки запроса в СУБД PostgreSQL можно описать следующим образом. Пользователь при помощи библиотеки *libpq* запрашивает соединение с СУБД. Процесс *postmaster* устанавливает соединение с клиентом, порождает новый процесс *backend*, передает ему дескриптор соединения и переходит в состояние ожидания. Процесс *backend* передает пользователю сообщение «с*оединение установлено*».

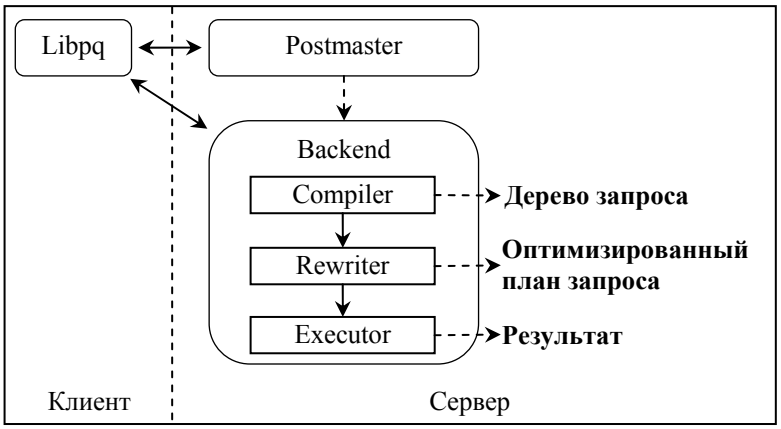

Рисунок 1. Схема обработки запроса
в СУБД PostgreSQL.

Пользователь передает запрос на сервер. Процесс *backend* принимает запрос и выполняет его обработку. По окончании обработки запроса *backend* передает пользователю результат.

Анализ архитектуры СУБД PostgreSQL показал, что для реализации параллельной обработки запросов на каждом узле кластерной вычислительной системы необходимо размещать компоненты *postmaster* и *backend*. При этом задача доставки запроса в параллельную СУБД и задача получения результата должны обеспечиваться коммуникационной библиотекой libpq и медиатором, выполняющим функцию прокси-сервера между клиентом и вычислительными узлами кластера, а также координирующего ход параллельной обработки запроса на вычислительных узлах кластера.

Организация параллельной обработки запросов в СУБД PostgreSQL

Реализация параллельной обработки запросов в кластерных вычислительных системах основывается на подходе, называемом инкапсуляцией параллелизма [7]. В соответствии с данным подходом в код СУБД PostgreSQL встраиваются специальные «гранулы», обеспечивающие параллельную обработку запроса. При этом изменения в коде СУБД сводятся к минимуму. Для проектирования и разработки компонентов СУБД, обеспечивающих параллельную обработку запросов в рамках данной работы на термин «вычислительный кластер» накладываются следующие ограничения.

Кластер состоит из группы вычислительных узлов и хост-машины. Вычислительный узел содержит один процессор, диск и модуль оперативной памяти. Вычислительные узлы однородны в смысле производительности и размера дисковой и оперативной памяти. Вычислительные узлы связаны между собой высокопроизводительной коммуникационной сетью передачи данных и сервисной коммуникационной сетью для передачи служебных команд. Хост-машина представляет собой выделенный вычислительный узел, основная задача которого состоит в обеспечении взаимодействия пользователя с кластером. Хост-машина взаимодействует с вычислительными узлами при помощи сервисной сети. Для уменьшения загрузки хост–машины в вычислительный кластер может быть добавлена одна или несколько хост–машин.

Схема параллельной обработки запросов в кластерной вычислительной системе показана на рис. 2. На хост–машине выполняется *медиатор*, обеспечивающий взаимодействие пользователя с параллельной СУБД. На каждом вычислительном узле выполняется экземпляр *ядра СУБД*, состоящий из СУБД PostgreSQL со встроенными в нее средствами параллельной обработки запросов.

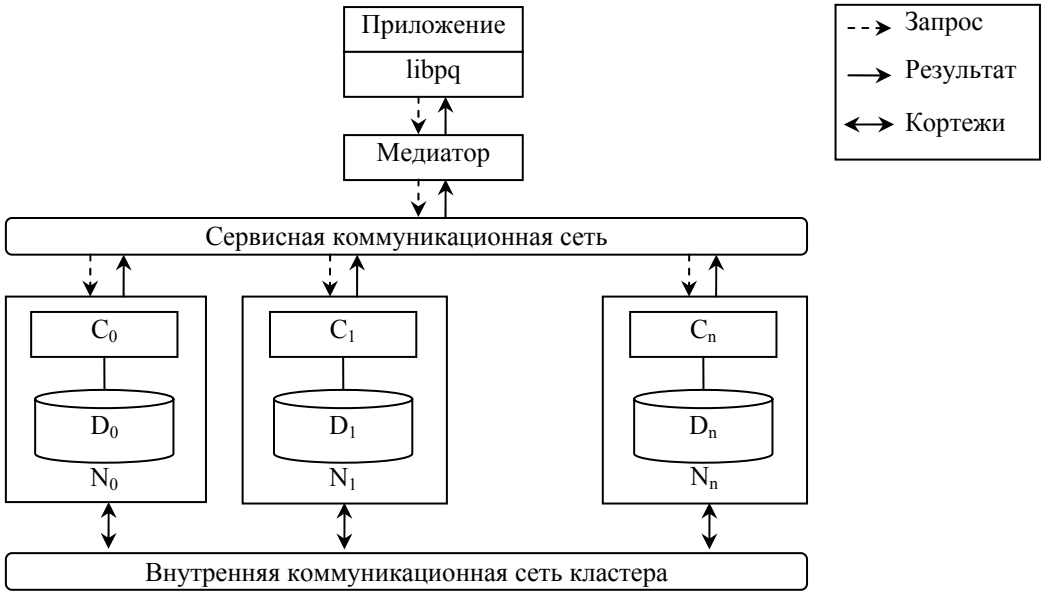

Рисунок 2. Схема параллельной обработки запроса. N_i – вычислительный узел кластера; C_i – ядро параллельной СУБД; D_i – фрагмент базы данных на i-том узле.

Экземпляр процесса *backend*, выполняющего обработку запроса на вычислительном узле, будем называть параллельным агентом [8]. Среди параллельных агентов выделяется агент-сборщик, который собирает результаты обработки запроса от всех параллельных агентов и формирует итоговое результирующее отношение.

Размещение базы данных на узлах кластера задается следующим образом [8]. Каждое отношение разбивается на непересекающиеся фрагменты, которые размещаются на различных вычислительных узлах. При этом на каждом узле располагается один фрагмент данного отношения. Ядро СУБД управляет обработкой фрагментов, расположенных на диске своего вычислительного узла.

Процесс параллельной обработки запроса можно описать следующим образом.

1. Клиент устанавливает соединение и передает запрос медиатору.
2. Медиатор устанавливает соединение и отправляет запрос каждому ядру СУБД через сервисную коммуникационную сеть кластера.
3. Ядро СУБД принимает запрос и запускает параллельного агента
4. Параллельный агент выполняет обработку запроса и передает полученный результат на агент-сборщик.
5. Агент-сборщик передает полученный результат медиатору.
6. Медиатор принимает результат выполнения запроса от агента-сборщика и передает его клиенту.

Структура параллельного агента

В соответствии с концепцией инкапсуляции параллелизма, параллельный агент разрабатывается на базе исполнителя запросов СУБД PostgreSQL. В код штатного исполнителя запросов встраивается специальная подсистема «Параллелизатор», которая обеспечивает взаимодействия и обмен данными между параллельными агентами на каждом этапе обработки запроса. Компоненты параллелизатора и его место в структуре параллельного агента показаны на рис. 3.

Генератор параллельного плана запроса выполняет анализ последовательного плана запроса и вставляет в определенные места плана специальный оператор обменов *exchange* [9] представляющий собой оператор физической алгебры.

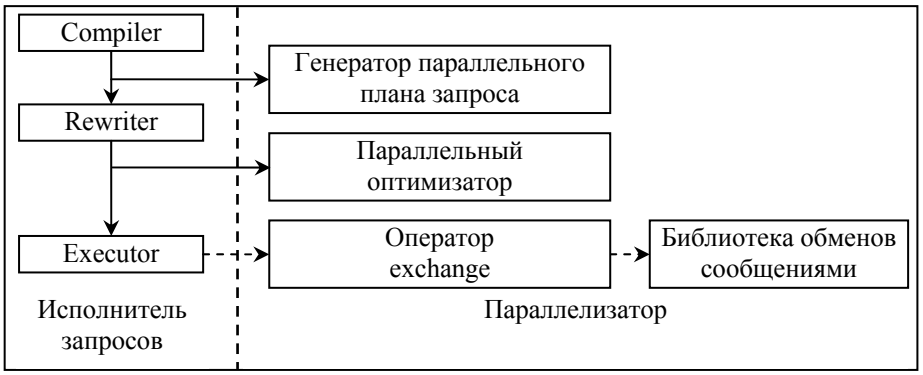

Рисунок 3. Структура параллельного агента.

Основной задачей оператора exchange является организация обменов данными в процессе обработки запроса. Параллельный оптимизатор выполняет настройку параллельного плана запроса с целью минимизации обменов данными между параллельными агентами в ходе обработки запроса.

В процессе интерпретации плана запроса исполнитель обращается к оператору exchange, который организует обмены кортежами между параллельными агентами. Библиотека обменов сообщениями предоставляет параллельному агенту следующий минимальный набор асинхронных функций обмена данными по внутренней коммуникационной сети кластера:

1. *Create_connection()* – устанавливает соединение между параллельными агентами, задействованными в обработке запроса.
2. *Close_Connection()* – закрывает установленное ранее соединение.
3. *Isend()* – инициализировать отправку запроса. Возвращает идентификатор операции *opt*. Получить состояние отправки сообщения можно при помощи функции *Test()*.
4. *Irecv()* – инициализировать операцию приема сообщения. Возвращает идентификатор операции *opt*. Получить сведения о завершении операции приема можно при помощи функции *Test()*.
5. Test() – проверить состояние операции приема сообщения. Возвращает значение состояния операции, заданной идентификатором *opt* (1 – завершено/ 0 – не завершено).

Медиатор

Основным требованием к медиатору является минимизация времени отклика на действия клиента, что предопределяет его простую структуру и небольшую функциональную нагрузку.

Медиатор выполняет следующие основные операции.

1. Доставка запроса ядрам СУБД. Медиатор принимает запрос от клиента и формирует *дескриптор запроса*. Дескриптор содержит уникальный идентификатор запроса, адрес клиента и список ядер СУБД, задействованных в обработке запроса. Запрос и уникальный идентификатор запроса доставляются каждому ядру СУБД.
2. Доставка результатов обработки запроса клиенту. Медиатор ожидает результатов обработки запроса от ядер СУБД и по мере их поступления передает клиенту. После завершения обработки запроса ядро СУБД передает медиатору сообщение «запрос обработан». Получив данное сообщение от всех ядер СУБД медиатор удаляет дескриптор запроса.
3. Прерывание обработки запроса. В случае отмены клиентом операции выполнения запроса медиатор отправляет ядру СУБД сообщение «отменить

обработку запроса».

4. Обработка сбоев. В случае выхода из строя одного из вычислительных узлов медиатор выполняет отмену обработки запроса на ядрах СУБД и выполняет повторный запуск запроса.

Заключение

В работе описана архитектура параллельной СУБД для кластерных вычислительных систем, разрабатываемая на базе СУБД PostgreSQL. Представлен метод параллельной обработки запросов основанный на подходе, называемом инкапсуляцией параллелизма. Представлено описание подсистемы медиатор и библиотеки обменов сообщениями.

В настоящее время ведется разработка подсистемы медиатор и библиотеки обменов сообщениями. Выполняется проектирование подсистемы «Параллелизатор», обеспечивающей вставку операторов exchange в план запроса.

В качестве направлений дальнейших исследований можно отметить следующие. Во-первых, это – исследование масштабируемости разрабатываемой параллельной СУБД. Во-вторых, планируется исследование влияния перекосов на время обработки запросов и реализация механизма балансировки загрузки. В-третьих, планируется разработка метода миграции данных, который позволит осуществлять профилактику отдельных узлов кластера в прозрачном для пользователей режиме.

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (проект 09-07-00241-а) и Совета по грантам Президента Российской Федерации (грант МК-3535.2009.9).

Литература

1. *Игнатович Н.* Семейство реляционных систем баз данных IBM DB2 // СУБД. 1997. №2. С. 5-17.
2 *Chambers L., Cracknell D.* Parallel Features of NonStop SQL // Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems (PDIS 1993), Issues, Architectures, and Algorithms, San Diego, CA, USA, January 20–23, 1993. – IEEE Computer Society, 1993. –P. 69–70.
3. *Page J.* Study of a Parallel Database Machine and Its Performance: the NCR/Teradata DBC/1012 // Advanced Database Systems, 10th British National Conference on Databases (BNCOD 10), Aberdeen, Scotland, July 6-8, 1992, Proceedings. Springer, 1992 (Lecture Notes on Computer Science, Vol. 618). P. 115-137.
4. *Kotowski N., Lima A., Pacitti E., Valduriez P., Mattoso M.* Parallel Query Processing for OLAP in Grids // Concurrency and Computation: Practice and Experience. Wiley InterScience. 2008.
5 *Orenstein J.* Horizontal Scalability with PostgreSQL // Proceedings of t*he PostgreSQL Conference (PGCon'2008), Ottawa, Canada, May 20–21, 2008. URL: http://www.pgcon.org/2008/schedule/events/57.en.html (дата* обращения*: 5.10.2009)*
6. *Momjian B.* Flowchart of the PostgreSQL backend. URL: http://www.postgresql.org/developer/ext.backend.html (дата обращения: 04.10.2009)
7. *Graefe G.* Query Evaluation Techniques for Large Databases // ACM Computing Surveys. 1993. Vol. 25, No. 2. P. 73-169.
8. *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.
9. *Соколинский Л.Б.* Организация параллельного выполнения запросов в многопроцессорной машине баз данных с иерархической архитектурой // Программирование. 2001. No. 6. С. 13-29.