Архитектура и принципы реализации параллельной СУБД PargreSQL*

К.С. Пан, М.Л. Цымблер

Южно-Уральский государственный университет

В работе описана архитектура и принципы реализации параллельной СУБД PargreSQL для кластерных вычислительных систем, разрабатываемой на основе свободно распространяемой СУБД PostgreSQL.

1. Введение

СУБД PostgreSQL [1] представляет собой свободно распространяемую реляционную СУБД с открытым исходным кодом. Научный проект Омега [2] направлен на разработку прототипа параллельной СУБД для мультипроцессорных вычислительных систем с кластерной архитектурой.

Параллельная СУБД PargreSQL разрабатывается в рамках проекта Омега. Базовой идеей этой разработки является внедрение поддержки фрагментного параллелизма [3] в СУБД PostgreSQL. В данной работе описана архитектура и основные принципы разработки СУБД PargreSQL.

Работа имеет следующую структуру. В разделе 2 приводится краткий обзор работ по тематике исследования. Раздел 3 содержит описание архитектуры СУБД PostgreSQL. В разделах 4 и 5 описана архитектура и принципы реализации СУБД PargreSQL. Раздел 6 содержит заключение.

2. Работы по тематике исследования

В настоящее время СУБД PostgreSQL представляет собой надежную свободно распространяемую на уровне исходных кодов альтернативу коммерческим СУБД. Существует достаточно большое количество практических приложений баз данных на основе PostgreSQL и исследовательских проектов, посвящённых расширению и улучшению PostgreSQL.

В работе [4] рассматривается внедрение поддержки XML в PostgreSQL. Добавление новых типов данных для обеспечения поддержки стандарта обмена медицинской информацей HL7 в PostgreSQL описывается в [5]. Авторы работы [6] предлагают расширение PostgreSQL для обработки изображений. В работе [7] представлен подход к интеграции PostgreSQL с Semantic Web.

Исследования, посвященные использованию PostgreSQL для параллельной обработки запросов могут быть представлены следующими работами. В [8] предлагается расширение PostgreSQL для распределенной обработки запросов. Описаны необходимые изменения в исполнителе запросов PostgreSQL и предложены соответствующие способы увеличения производительности.

СУБД ParGRES [9] представляет собой промежуточное программное обеспечение (middleware) с открытым исходным кодом для высокопроизводительной обработи OLAP-запросов. ParGRES использует внутризапросный параллелизм на кластерных вычислительных системах и адаптивное виртуальное распределение базы данных. СУБД GParGRES [10] является продолжением продукта ParGRES для грид-сред. GParGRES использует репликацию базы данных и меж- и внутризапросный параллелизм для эффективной обработки OLAP-запросов в грид. Предложенный подход подразумевает разбиение данных на двух

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

уровнях: на уровне грид (реализовано в GParGRES) и на уровне узлов (реализовано в ParGRES).

Нами предлагается внедрение фрагментного параллелизма [11] в СУБД PostgreSQL на основе методов параллельной обработки запросов, разработанных в рамках проекта Омега [12, 13].

3. Архитектура PostgreSQL

В основе архитектуры PostgreSQL лежит модель «клиент-сервер». В сеансе работы с PostgreSQL участвуют три вида взаимодействующих процессов (см. Рис. 1): *приложение-клиент (frontend)*, *серверный процесс (backend)* и *демон (daemon)*. Демон осуществляет прием соединений, устанавливаемых клиентами, и создает отдельный серверный процесс для обработки запросов каждого отдельного клиента.

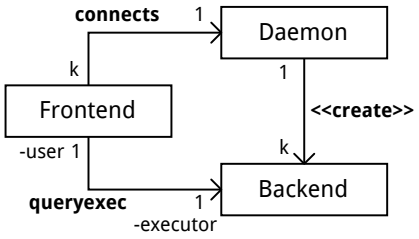

Рис. 1. Процессы СУБД PostgreSQL

Порядок взаимодействия клиента и СУБД представлен на Рис. 2. Сначала клиент устанавливает соединение с демоном. Демон принимает соединение от клиента и затем с помощью системного вызова `fork()` создает серверный процесс. После этого клиент отправляет запрос серверному процессу, который выполняет обработку этого запроса и отправку результатов обратно клиенту.

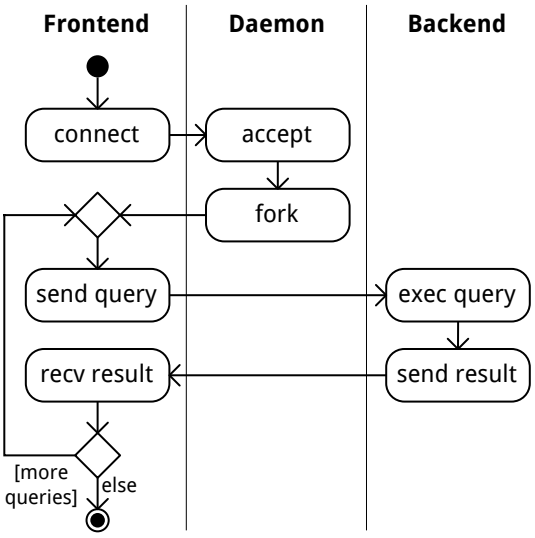

Рис. 2. Взаимодействие клиента и сервера PostgreSQL

Обработка запроса состоит из следующих этапов (см. Рис. 3):

- *parse* — разбор запроса на языке SQL;
- *rewrite* — преобразование запроса;
- *plan/optimize* — составление плана запроса и его оптимизация;

- *execute* — выполнение плана запроса.

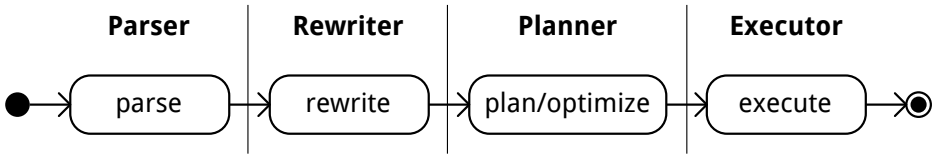

Рис. 3. Обработка запроса в СУБД PostgreSQL

СУБД PostgreSQL содержит следующие подсистемы, представленные на Рис. 4:

- *Parser* — подсистема, которая осуществляет разбор SQL-запросов;
- *Rewriter* — подсистема, выполняющая преобразование запроса в соответствии с правилами подстановки, которые хранятся в базе данных (например, для реализации представлений);
- *Storage* — подсистема хранения данных и метаданных;
- *Planner* — подсистема, которая выполняет составление плана запроса;
- *Executor* — подсистема, исполняющая план запроса;
- *libpq* — библиотека, реализующая протокол взаимодействия клиента (libpq-fe) и сервера (libpq-be).

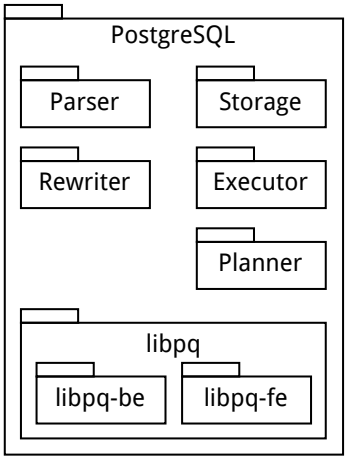

Рис. 4. Подсистемы СУБД PostgreSQL

Размещение компонентов СУБД PostgreSQL приведено на Рис. 5.

На клиенте размещается библиотека `libpq` и приложение пользователя. Все остальные компоненты размещаются на узлах сервера.

4. Архитектура PargreSQL

PargreSQL использует идею фрагментного параллелизма [3]. Общая схема параллельной обработки запроса представлена на Рис. 6. Каждое отношение (таблица) базы данных делится на горизонтальные *фрагменты*, распределяемые по процессорным узлам вычислительной системы. Способ фрагментации определяется *функцией фрагментации*, вычисляющей для каждого кортежа отношения номер процессорного узла, на котором должен быть

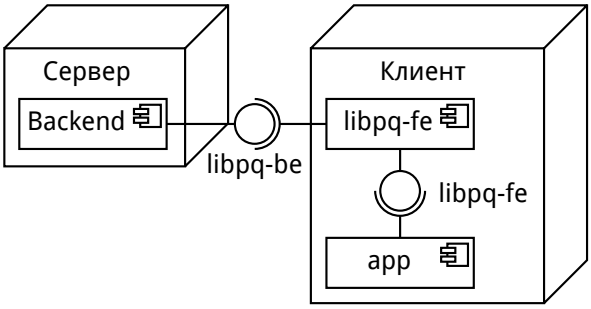

Рис. 5. Размещение компонентов PostgreSQL

размещен этот кортеж. Запрос выполняется в виде нескольких параллельных процессов (*агентов*), каждый из которых обрабатывает отдельный фрагмент отношения. Полученные фрагменты сливаются в результирующее отношение.

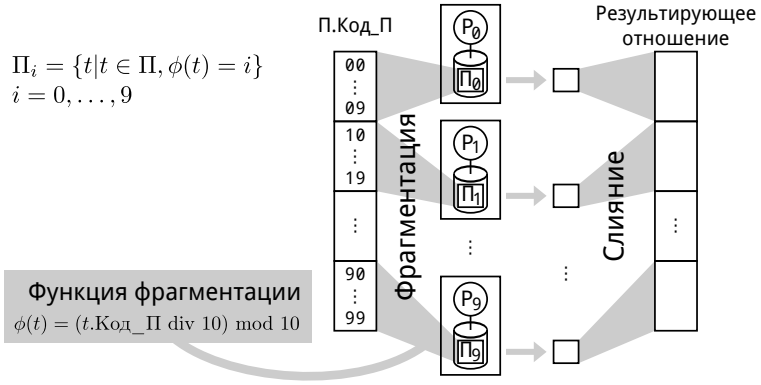

Рис. 6. Параллельная обработка запроса на основе фрагментного параллелизма

Архитектура клиент-серверного взаимодействия параллельной СУБД PargreSQL, в отличие от последовательной СУБД PostgreSQL, предполагает, что клиент взаимодействует с двумя или более серверами одновременно (см. Рис. 7).

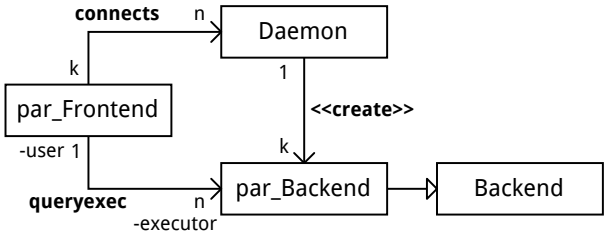

Рис. 7. Процессы СУБД PargreSQL

Порядок взаимодействия клиента и СУБД PargreSQL представлен на Рис. 8. Клиентское приложение подключается сразу ко всем экземплярам СУБД и отправляет им одинаковый запрос.

Параллельная обработка запроса состоит из следующих этапов (см. Рис. 9):

- *parse* — разбор запроса на языке SQL;
- *rewrite* — преобразование запроса;
- *plan/optimize* — составление последовательного плана запроса и его оптимизация;

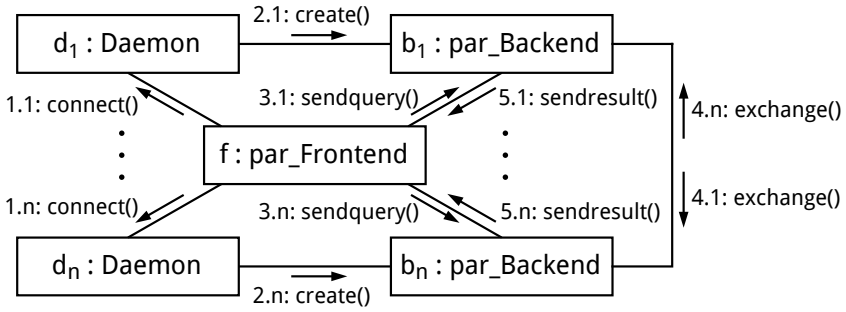

Рис. 8. Взаимодействие клиента и серверов PargreSQL

- *parallelize* — формирование параллельного плана запроса на основе последовательного путем вставки операторов *exchange*;
- *execute* — выполнение плана запроса;
- *balance* — балансировка загрузки серверных процессов.

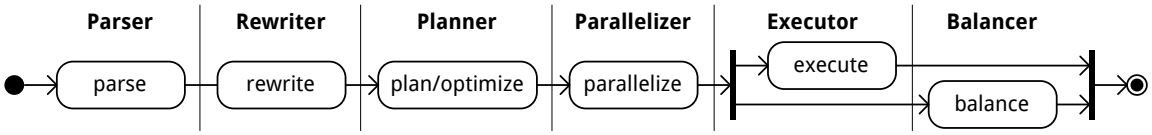

Рис. 9. Обработка запроса в СУБД PargreSQL

Архитектура PargreSQL представлена на Рис. 10. PostgreSQL является подсистемой в рамках системы PargreSQL. Для разработки PargreSQL необходимо внести изменения в исходные тексты следующих подсистем PostgreSQL: Storage, Executor и Planner.

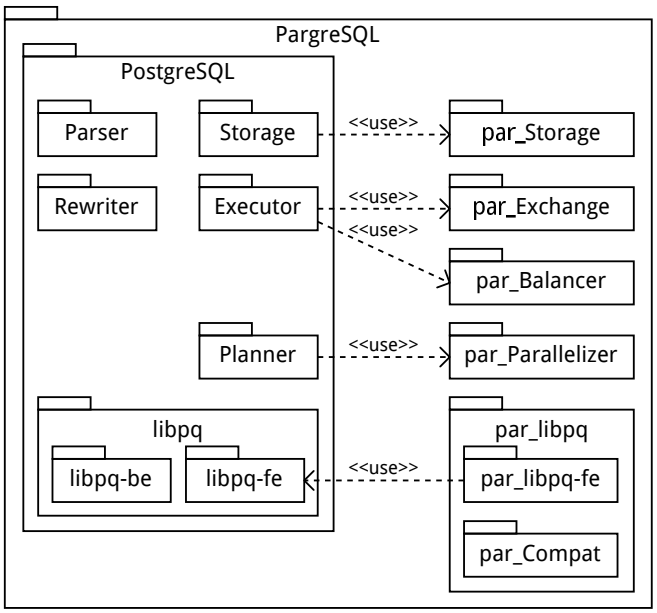

Рис. 10. Архитектура СУБД PargreSQL

Данные изменения обеспечивают внедрение следующих новых подсистем:

- *par_Storage* — подсистема хранения данных о фрагментации отношений;

- *par_Exchange* — подсистема, реализующая оператор *exchange* [3], который выполняет обмен кортежами между экземплярами СУБД;
- *par_Parallelizer* — подсистема, выполняющая добавление в нужные места последовательного плана запроса операторов *exhange*;
- *par_Balancer* — подсистема, выполняющая динамическую балансировку загрузки серверных процессов.

Задача оператора *exchange* — передать все кортежи, которые должны быть обработаны ядрами PargreSQL на других вычислительных узлах, и получить все кортежи, предназначенные для обработки ядром PargreSQL на данном узле. Реализация оператора *exchange* инкапсулирует все аспекты, связанные с распараллеливанием запроса, так как он имеет стандартный итераторный интерфейс и может быть помещен в любое место дерева запроса.

В PargreSQL также входят следующие новые подсистемы, которые не требуют изменения оригинальных подсистем PostgreSQL:

- *par_libpq-fe* — надстройка над `libpq-fe`, реализующая тиражирование запроса;
- *par_Compat* — подсистема, реализующая прозрачное для приложения подключение `par_libpq-fe`.

Размещение компонентов СУБД PargreSQL приведено на Рис. 11.

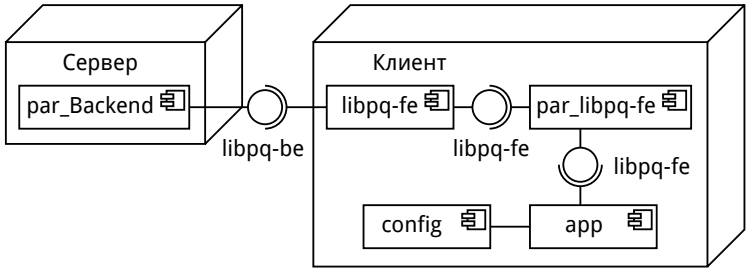

Рис. 11. Размещение компонентов PargreSQL

На клиенте размещаются библиотеки `par_libpq` и `libpq-fe` и приложение пользователя вместе с конфигурационным файлом в формате XML, в котором определяются параметры работы приложения (адреса узлов, фрагментация таблиц и др.). Остальные компоненты размещаются на узлах сервера.

5. Принципы реализации PargreSQL

PargreSQL разрабатывается в соответствии со следующими основными принципами: масштабируемость, минимальность и прозрачность.

Масштабируемость заключается в том, что параллельная СУБД PargreSQL, запущенная на одном вычислительном узле, работает так же, как последовательная СУБД PostgreSQL.

Масштабируемость реализуется путем использования оператора *exchange* [3]. Оператор *exchange* вычисляет значение функции пересылки для каждого поступающего кортежа и передает кортеж на вычислительный узел, номер которого совпадает со значением функции пересылки.

Таким образом, оператор *exchange* не изменяет кортеж (передает его в вышележащий узел плана), если значение функции пересылки совпадает с номером текущего узла. Когда

PargreSQL запускается на одном узле, функция пересылки тождественно равна номеру единственного узла — нулю.

В исходные тексты PostgreSQL вносятся *минимальные* изменения. Изменения в структурах данных и алгоритмах инкапсулируются в новых файлах исходных текстов, подключаемых к исходным текстам PostgreSQL.

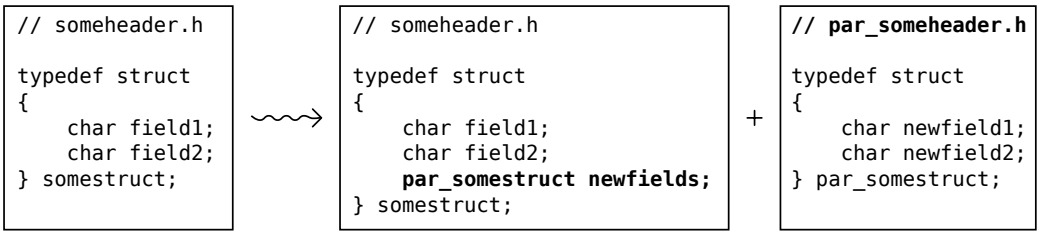

Рис. 12. Техника внесения изменений в определения структур данных

На Рис. 12 показан пример применения данного подхода для добавления новых полей в оригинальную структуру данных. В новом файле описывается тип `par_somestruct`, содержащий новые поля, а в оригинильную структуру добавляется новое поле типа `par_somestruct`.

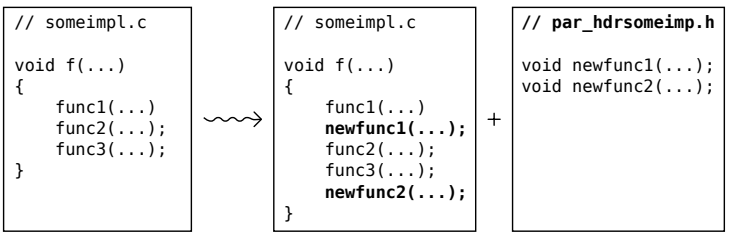

Рис. 13. Техника внесения изменений в исходные тексты функций

На Рис. 13 показан пример применения данного подхода для изменения оригинальных алгоритмов. В тело оригинальной функции добавляется вызов новой функции `newfunc()`, а сама функция `newfunc()` определяется в файле исходных текстов новой подсистемы.

Использование PargreSQL является *прозрачным* для пользовательских приложений. Подключение PargreSQL к прикладным программам, которые до этого использовали PostgreSQL, производится с минимальными изменениями в исходных кодах приложения.

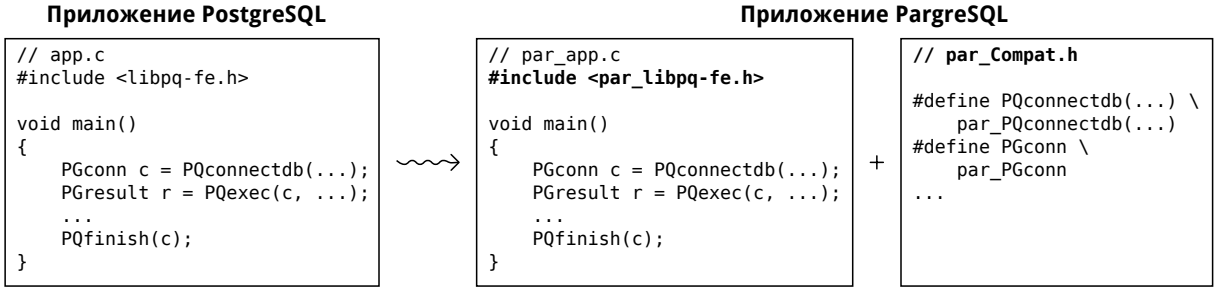

Рис. 14. Прозрачность использования `par_libpq`

Прозрачность реализуется следующим образом. Пользовательское приложение вместе с заголовочным файлом `par_libpq-fe.h` подключает заголовочный файл `par_Compat.h`. Этот файл содержит объявление макросов, заменяющих вызовы функций подсистемы `libpq` на вызовы функций подсистемы `par_libpq`. Таким образом, для адаптации PostgreSQL-приложения в исходном тексте приложения требуется изменение одной строки кода.

На Рис. 14 показан прозрачный способ подключения `par_libpq`.

6. Заключение

В данной работе описана архитектура и принципы реализации параллельной СУБД PargreSQL для многопроцессорных вычислительных систем с кластерной архитектурой. PargreSQL основана на свободной СУБД PostgreSQL и использует фрагментный параллелизм.

Направлением дальнейших исследований является завершение разработки PargreSQL и проведение экспериментов по исследованию ее ускорения и масштабируемости.

Литература

1. *Stonebraker M., Kemnitz G.* The POSTGRES next-generation database management system // Communications of the ACM. Oct. 1991. Vol. 34, No. 10. P. 78–92.
2. *Sokolinsky L., Axenov O., Gutova S.* Omega: The Highly Parallel Database System Project // Proceedings of the First East-European Symposium on Advances in Database and Information Systems (ADBIS'97), St.-Petersburg, September 2–5, 1997. St.-Petersburg: Nevsky Dialect. 1997. Vol. 2. P. 88–90.
3. *Соколинский Л.Б.* Организация параллельного выполнения запросов в многопроцессорной машине баз данных с иерархической архитектурой // Программирование. 2001. № 6. С. 13-29.
4. *Samokhvalov N.* XML Support in PostgreSQL // SYRCoDIS, volume 256 of *CEUR Workshop Proceedings*. 2007.
5. *Havinga Y., Dijkstra W., de Keijzer A.* Adding HL7 version 3 data types to PostgreSQL // CoRR, abs/1003.3370, 2010.
6. *Guliato D., de Melo E.V., Rangayyan R.M., Soares R.C.* POSTGRESQL-IE: An Image-handling Extension for PostgreSQL // Journal of Digital Imaging, 22(2):149–165. 2009.
7. *Levshin D.V., Markov A.S.* Algorithms for integrating PostgreSQL with the semantic web // Programming and Computer Software, 35(3):136–144. 2009.
8. *Lee R., Zhou M.* Extending PostgreSQL to Support Distributed/Heterogeneous Query Processing // Database Systems for Advanced Applications, volume 4443 of *Lecture Notes in Computer Science*, P. 1086–1097. Springer. 2007.
9. *Paes M., Lima A.A.B., Valduriez P., Mattoso M.* High-Performance Query Processing of a Real-World OLAP Database with ParGRES // VECPAR, volume 5336 of *Lecture Notes in Computer Science*, P. 188–200. Springer. 2008.
10. *Kotowski N., Lima A.A.B, Pacitti E., Valduriez P., Mattoso M.* Parallel query processing for OLAP in grids // Concurrency and Computation: Practice and Experience, 20(17):2039–2048. 2008.
11. *DeWitt D.J., Gray J.* Parallel Database Systems: The Future of High Performance Database Systems // Communications of the ACM, 35(6):85–98. 1992.
12. *Sokolinsky L.B.* Organization of Parallel Query Processing in Multiprocessor Database Machines with Hierarchical Architecture // Programming and Computer Software, 27(6):297–308. 2001.
13. *Lepikhov A.V., Sokolinsky L.B.* Query processing in a DBMS for cluster systems // Programming and Computer Software, 36(4):205–215. 2010.