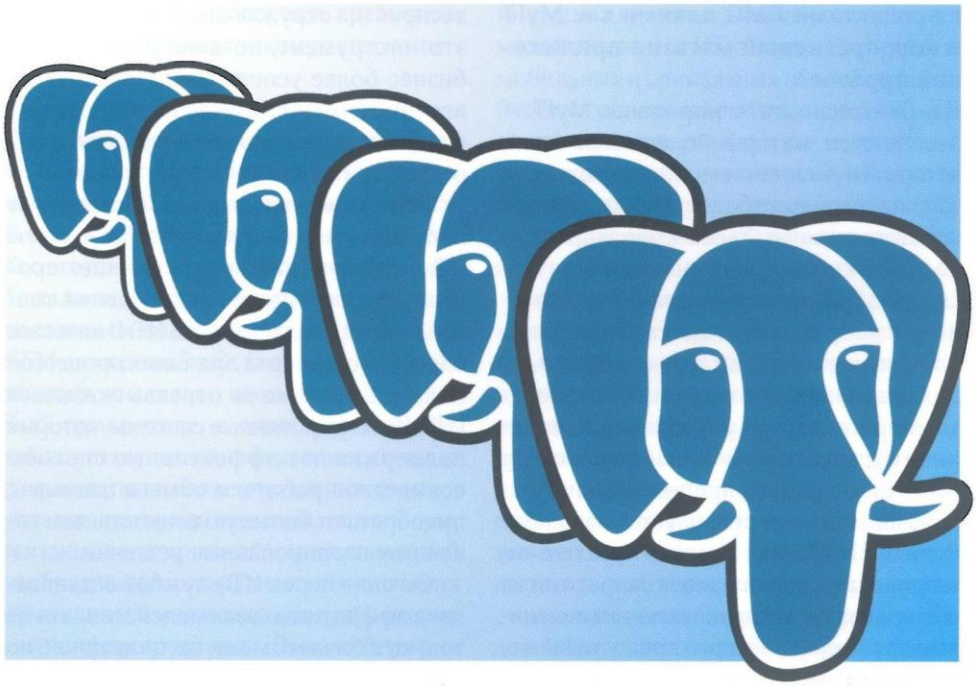
Интеграция параллелизма в СУБД с открытым кодом

Одним из основных факторов, влияющих сегодня на технологии обработки данных, является появление сверхбольших баз данных, требующих параллельных СУБД. Спектр предлагаемых таких СУБД сегодня достаточно широк, однако они либо дороги, либо отрицают реляционную модель данных. Для создания эффективных и относительно недорогих параллельных СУБД можно непосредственно в открытый код последовательной СУБД интегрировать параллельную обработку запросов.

Ключевые слова: параллельные СУБД, фрагментный параллелизм, PostgreSQL, открытый код



Константин Пан,
Леонид Соколинский,
Михаил Цымблер

Спектр параллельных систем баз данных, предлагаемых сегодня пользователям, достаточно широк. С одной стороны, это многочисленный клан систем NoSQL: Hadoop, Cassandra, MongoDB, CouchDB, Redis и др. Эти системы хорошо масштабируются на платформе кластерных вычислительных систем и распространяются свободно на уровне исходных кодов. Однако по своей природе NoSQL-системы отрицают реляционную модель данных, остающуюся сегодня стандартом де-факто для хранения и обработки данных. С другой стороны — параллельные СУБД на основе реляционной модели данных. Здесь преобладают проприетарные решения: Teradata, Greenplum, IBM DB2 Parallel Edition, Oracle Real Application Clusters и др., которые недешевы и обычно ориентированы на специфические аппаратно-программные платформы.

Решением проблемы отсутствия свободной параллельной СУБД для массово используемых многоядерных и многопроцессорных систем могла бы стать разработка параллельной СУБД на основе имеющейся последовательной СУБД. Этот подход реализован, например, в исследовательских проектах ParGRES и GParGRES, а также в свободной СУБД HadoopDB. В данном случае разработчики дополнили последовательную свободную СУБД PostgreSQL, запускаемую на узлах кластера, программным обеспечением связующего слоя, реализующим распараллеливание запросов и сборку результатов в единую таблицу.

Однако есть и другой путь получения эффективных, хорошо масштабируемых и относительно недорогих параллельных СУБД — внедрение механизмов параллельной обработки запросов непосредственно в код последовательной СУБД с открытыми кодами, например в PostgreSQL.

Проект PostgreSQL был начат в 1995 году как ответвление от проекта POSTGRES Майкла Стоунбрейкера и до сих пор разрабатывается группой энтузиастов, став сегодня одной из наиболее популярных СУБД с открытым кодом. Сначала PostgreSQL отличался от своего предка только наличием SQL-синтаксиса в запросах, но сегодня эта СУБД представляет собой полноценную объектно-реляционную СУБД с открытым кодом для практически всех популярных операционных систем. PostgreSQL поддерживает стандарт SQL:2011, ACID-транзакции и хранимые процедуры на различных языках высокого уровня. Максимальный размер таблицы в PostgreSQL равен 32 Тбайт, а максимальный размер поля таблицы — 1 Гбайт, этого пока достаточно для работы со сверхбольшими по нынешним меркам базами. PostgreSQL имеет хорошо документированный код и применяется в многочисленных коммерческих организациях, госструктурах и университетах.

Проект PargreSQL [1], выполняемый в Южно-Уральском государственном национальном исследовательском университете, посвящен разработке параллельной СУБД путем внедрения фрагментного параллелизма в СУБД PostgreSQL.

Фрагментный параллелизм [2] подразумевает горизонтальную фрагментацию каждой таблицы базы данных по дискам кластерной системы (рис. 1). Способ фрагментации определяется функцией фрагментации, которая получает запись таблицы с указанием поля фрагментации в этой таблице и выдает номер диска, где должна храниться данная запись. На каждом узле запускается параллельный агент, представляющий собой модифицированный экземпляр СУБД PostgreSQL, обрабатывающий свои фрагменты. Затем частичные результаты сливаются в результирующую таблицу. В рамках технологии внедрения фрагментного параллелизма СУБД PostgreSQL рассматривается как одна из подсистем СУБД PargreSQL (рис. 2). Технология предполагает внесение изменений в подсистемы оригинальной СУБД и реализацию ряда новых подсистем.

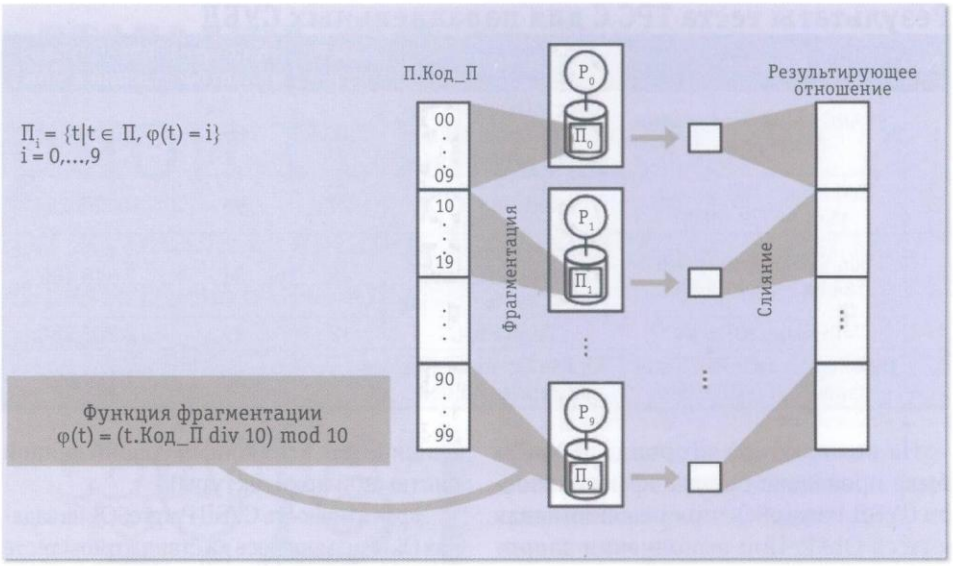


Рис. 1. Фрагментный параллелизм

Важным аспектом технологии является добавление в этапы обработки запроса шага построения параллельного плана запроса, представляющего собой последовательный план запроса, в нужные места которого добавлен специальный оператор обмена EXCHANGE [3]. Данный оператор обеспечивает обмены данными между параллельными агентами во время исполнения запроса и инкапсулирует в себе все механизмы, необходимые для реализации внутриоперационного параллелизма. Исполнитель запросов СУБД PostgreSQL выполняет оператор EXCHANGE таким образом, как если бы это был обычный реляционный оператор плана (соединение таблиц, выборка из таблицы, удаление дубликатов и др.), не замечая его «параллельной природы».

Оператор EXCHANGE имеет два атрибута: порт и функция пересылки. Порт представляет собой уникальный идентификатор оператора и позволяет включать в параллельный план запроса произвольное количество операторов EXCHANGE. Функция пересылки для каждой записи, поступающей на вход оператора EXCHANGE, возвращает номер узла, на котором должна быть обработана эта запись. Сам оператор состоит из четырех операторов: Split, Scatter, Gather и Merge. Оператор Split, получая запись, определяет, является ли она «своей», которую нужно обработать на текущем узле кластера, или «чужой», которую необходимо передать на другой узел. Оператор Scatter отправляет «чужие» записи на соответствующие им узлы, а Gather принимает «свои» записи от других узлов. Оператор Merge попеременно выдает результаты, получаемые операторами Gather и Split. Для реализации пересылок данных между узлами кластера использован интерфейс MPI.

Другой важный аспект технологии внедрения параллелизма — реализация фрагментации таблиц базы данных. СУБД PostgreSQL позволяет расширить синтаксис стандартной команды создания таблиц create table, что позволяет при формировании таблицы указать имя целочисленного поля, используемого для фрагментации. В качестве функции фрагментации берется остаток от деления поля фрагментации на количество вычислительных узлов в кластере.

Следующий аспект технологии — прозрачное портирование пользовательских приложений для PostgreSQL в PargreSQL. Для этого был разработан набор макросов подмены вызовов оригинальных функций API PostgreSQL на их копии PargreSQL. Таким образом, для адаптации PostgreSQL-приложения в исходном тексте приложения требуется изменить лишь одну строку кода: #include <par_libpq-fe.h> вместо #include <libpq-fe.h>.

Следует особо отметить, что перечисленные аспекты внедрения параллелизма в исходный код не являются исключительной прерогативой СУБД PostgreSQL — подобные изменения (оператор EXCHANGE, построение параллельного плана запроса и др.) могут быть проведены над любой СУБД с открытым кодом — например, MySQL. СУБД PostgreSQL была выбрана благодаря качественному и детально документированному исходному коду.

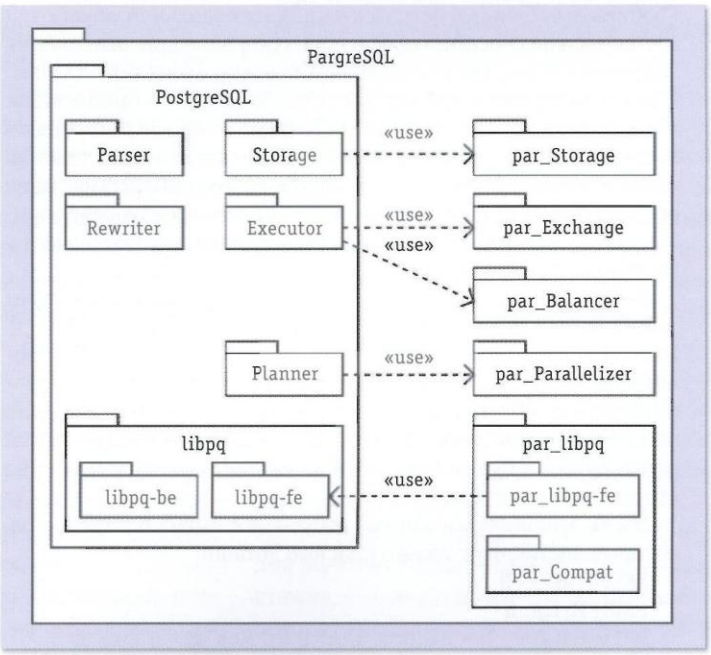


Рис. 2. Модульная структура СУБД PargreSQL

Результаты теста TPC C для параллельных СУБД

№	Кластер	СУБД	Кол-во узлов	Кол-во клиентов	tpm-C (транзакций в мин.)
1	SPARC SuperCluster with T3-4 Servers	Oracle Database 11g R2 Enterprise Edition w/RAC w/Partitioning	108	81	30 249 688
2	IBM Power 780 Server Model 9179-MHB	IBM DB2 9.7	24	96	10 366 254
3	Sun SPARC Enterprise T5440 Server Cluster	Oracle Database 11g Enterprise Edition w/RAC w/Partitioning	48	24	7 646 486
	«Торнадо ЮУрГУ»	PargreSQL	12	29	2 202 531
4	HP Integrity rx5670 Cluster Itanium2/1.5 GHz-64p	Oracle Database 10g Enterprise Edition	64	80	1 184 893

На компьютере «Торнадо ЮУрГУ» была проведена оценка эффективности СУБД PargreSQL при решении задач класса OLTP. При выполнении запроса, предполагавшего соединение двух таблиц размерами 3×10^8 и $7{,}5 \times 10^5$ записей, PargreSQL показала ускорение (отношение времени, требуемого разными конфигурациями при выполнении одного теста), близкое к линейному. Расширяемость, получаемая при переходе к все более сложной конфигурации (отношение времени, требуемого разными конфигурациями при выполнении теста возрастающей сложности), была исследована на таком же запросе с использованием от 1 до 128 узлов при одновременном увеличении количества узлов и объема данных (от таблиц $1{,}2 \times 10^6$ и 3×10^5 записей до $1{,}5 \times 10^8$ и $3{,}8 \times 10^6$ записей). Эксперименты также показали расширяемость, близкую к линейной (расширяемость остается равной единице для всех конфигураций данной системной архитектуры).

Эффективность СУБД PargreSQL на задачах OLTP измерялась на стандартном тесте TPC C, моделирующем процессы складского учета. С результатом 2,2 млн запросов в минуту PargreSQL вошла в пятерку лидеров рейтинга TPC C (см. таблицу) среди параллельных СУБД, реализованных на кластерных вычислительных системах (по состоянию на сентябрь 2013 года).

***

Параллельная СУБД PargreSQL, построенная на базе внедрения фрагментного параллелизма в СУБД PostgreSQL с открытым исходным кодом, показала хорошую масштабируемость на задачах обработки сверхбольших данных. Предложенная технология внедрения параллелизма может быть применена к другим свободным СУБД с открытым кодом, что открывает широким массам пользователей путь к решению задач Больших Данных. ■

ЛИТЕРАТУРА

1. Пан К.С., Цымблер М.Л. Разработка параллельной СУБД на основе последовательной СУБД PostgreSQL с открытым исходным кодом // Вестник ЮУрГУ. Сер. Математическое моделирование и программирование. 2012. № 18(277). Вып. 12. С. 112-120.

2. 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.

3. Лепихов А.В., Соколинский Л.Б. Обработка запросов в СУБД для кластерных систем // Программирование. 2010. № 4. С. 25-39.

Константин Пан (kvapen@gmail.com), Леонид Соколинский (sokolinsky@acm.org), Михаил Цымблер (zymbler@gmail.com) — сотрудники, Южно-Уральский государственный университет (Челябинск).

Статья подготовлена на основе материалов доклада, представленного авторами на конференции «Большие Данные в национальной экономике» (грант РФФИ 13-07-06055г). Работа выполнена при поддержке РФФИ (12-07-00443-а, 12-07-31217 мол_а) и стипендии Президента РФ (СП-5427.2013.5).

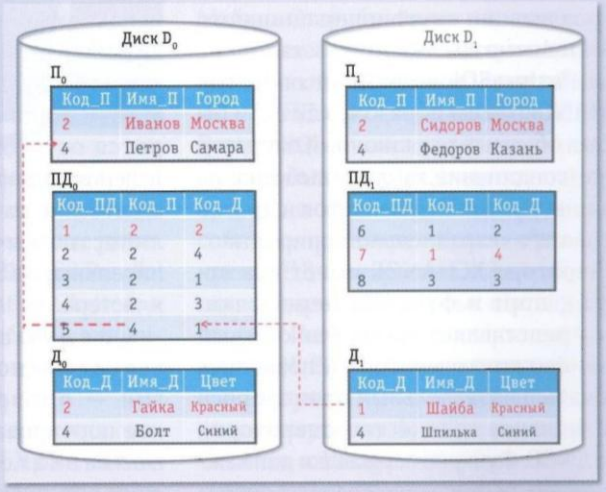
Обмены данными

Разбиение таблиц на фрагменты для их отдельной обработки потенциально способно обеспечить ускорение при выполнении запросов в n раз, где n — количество узлов в кластере. Однако в реальности этому препятствует необходимость организации обменов между параллельными агентами во время выполнения запроса. Дело в том, что при выполнении операции соединения двух таблиц по общему полю в случае, если поле фрагментации хотя бы одной из них не совпадает с полем, по которому производится соединение, необходимы обмены для обеспечения корректности результата.

Пусть имеется база данных о поставках, состоящая из трех таблиц: *поставщики П (Код_П, Имя_П, Город), детали Д (Код_Д, Имя_Д, Цвет) и поставки ПД (Код_ПД, Код_П, Код_Д).* Эти таблицы фрагментированы по двум дискам кластера: таблица П — по полю Код_П, таблица Д — по полю Код_Д и таблица ПД — по полю Код_П. При этом записи с нечетными значениями полей фрагментации будут храниться на первом, а с четными — на втором узле кластера.

Пусть выполняется запрос, выдающий имена поставщиков, которые поставляют только красные детали:

```
SELECT Имя_П
FROM П, ПД, Д
WHERE П.Код_П = ПД.Код_П AND ПД.Код_Д = Д.Код_Д
AND Д.Цвет = 'Красный';
```



Нетрудно увидеть, что при выполнении запроса над каждым фрагментом базы данных в отдельности отсутствие обменов приведет к неверному результату, поскольку в результирующую таблицу не будет включена запись Петров — она хранится на диске D_0 кластера, но при выполнении запроса должна быть обработана на диске D_1.