РАЗРАБОТКА ПАРАЛЛЕЛЬНОЙ СУБД НА ОСНОВЕ СВОБОДНОЙ СУБД POSTGRESQL

К.С. Пан

Введение

В настоящее время СУБД PostgreSQL [1] представляет собой надежную свободно распространяемую на уровне исходных кодов альтернативу коммерческим СУБД [2]. Ведутся интенсивные исследования, целью которых является расширение и улучшение СУБД PostgreSQL [3–6]. Нами предлагается модификация PostgreSQL для обеспечения параллельной обработки запросов. Предлагаемая параллельная СУБД, основанная на PostgreSQL, названа PargreSQL. В данной работе рассматриваются архитектура PargreSQL, реализация ее основных подсистем и вычислительные эксперименты.

Статья имеет следующую структуру. В разделе 2 представлена архитектура PargreSQL. Разделы 3, 4 и 5 посвящены реализации трех подсистем СУБД PargreSQL. Раздел 6 содержит описание вычислительных экспериментов.

Архитектура PargreSQL

PargreSQL использует идею фрагментного параллелизма [7]. 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 размещен этот кортеж. Запрос выполняется в виде нескольких параллельных процессов, каждый из которых обрабатывает отдельный фрагмент отношения. Полученные фрагменты сливаются в результирующее отношение.

Архитектура клиент-серверного взаимодействия в PargreSQL предполагает, что клиент взаимодействует с двумя и более серверами одновременно (см. Рис. 1).

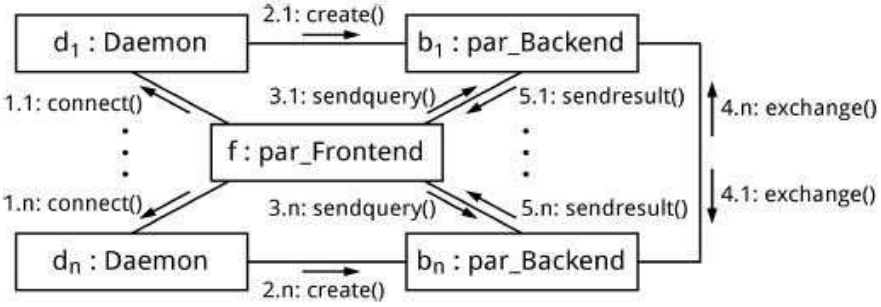

Рис. 1. Клиент-серверная модель PargreSQL

На шаге 1 клиентское приложение (par_Frontend) соединяется со всеми экземплярами СУБД, запущенными на узлах вычислительной системы. Затем, на шаге 2, демоны (Daemon) принимают соединения и создают для них обработчики (par_Backend), по одному на каждом вычислительном узле для каждого входящего соединения. На шаге 3 приложение отправляет запрос всем обработчикам. Шаг 4 заключается в обменах кортежами между экземплярами СУБД, которые необходимы для получения правильного результата. На шаге 5 приложение агрегирует результаты от всех экземпляров СУБД.

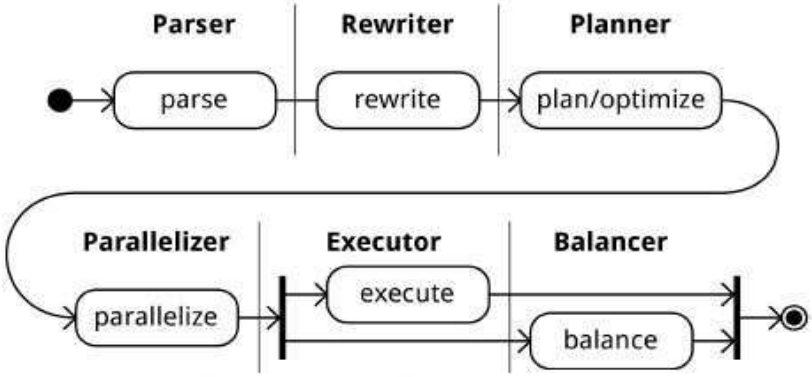

Рис. 2. Этапы обработки запроса

На Рис. 2 показаны этапы обработки запроса обработчиком par_Backend. На этапе **parallelize** предлагаемая нами подсистема *параллелизации запросов* превращает обычный план выполнения запроса в

параллельный путем вставки специального оператора **exchange** в нужные места плана.

Структура СУБД PargreSQL изображена на Рис. 3.

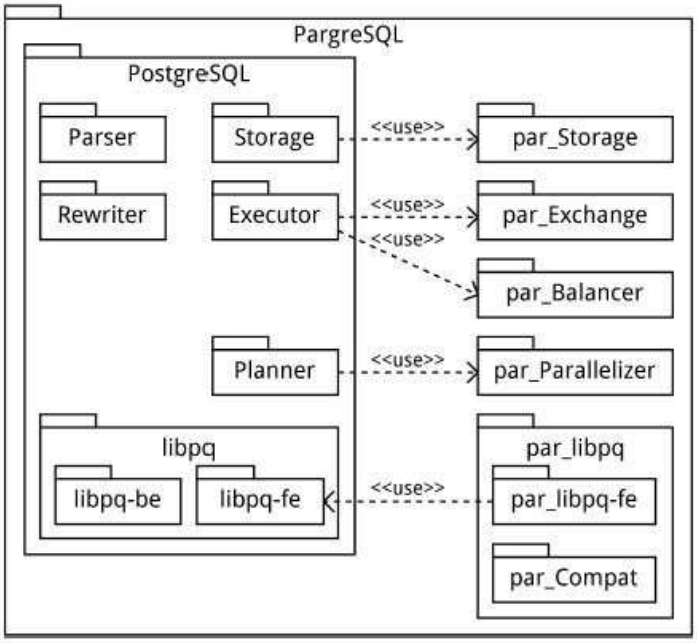

Рис. 3. Структура PargreSQL

Подсистема исполнения запросов Executor дополняется новой операцией — exchange. Подсистема хранения метаданных Storage расширяется для хранения данных о фрагментации отношений. Планировщик Planner дополняется параллелизатором Parallelizer. Пользовательская библиотека libpq дополняется оберткой par_libpq, тиражирующей запросы.

Подсистема *тиражирования запросов* служит для реализации шагов 1, 3 и 5 (см. Рис. 1). Другая подсистема — *оператор exchange* — реализует шаг 4. *Параллелизатор плана запроса* осуществляет модификацию плана, генерируемого планировщиком. Далее рассмотрены особенности реализации новых подсистем.

Тиражирование запросов

Приложения PostgreSQL используют библиотеку libpq-fe для доступа к СУБД. Данная библиотека реализует функции, необходимые для установления соединения с демоном, отправки запросов, получения результатов, проверки статуса выполнения запроса и т.п. В случае PargreSQL приложение подключает библиотеку-обертку par_libpq-fe, которая обладает аналогичным интерфейсом (см. Табл. 1), но реализует его путем многократного вызова соответствующих функций оригинальной библиотеки libpq-fe.

Табл. 1. Изменение интерфейса прикладной библиотеки libpq

libpq-fe	par_libpq-fe

struct **PGconn** {
...
}
 	
Хранит данные о соединении с сервером.
	struct **par_PGconn** {
	int len;
	struct PGconn *conns;
	}
	
	Хранит данные о соединениях с узлами PargreSQL. Является контейнером, содержащим массив структур PGconn.

PGconn ***PQconnectdb**(const char *conninfo) Устанавливает соединение с сервером, указанным в conninfo.	par_PGconn ***par_PQconnectdb**() Устанавливает соединение с узлами, указанными в конфигурационном файле PargreSQL.
ConnStatusType	ConnStatusType **par_PQstatus**(

PQstatus(const PGconn *conn) Возвращает статус соединения с сервером PostgreSQL.	const par_PGconn *conn) Возвращает статус соединения с узлами PargreSQL. Результат формируется путем агрегации результатов, полученных от вызова PQstatus() для всех вычислительных узлов.
PGresult ***PQexec**(PGconn *conn, const char *query) Отправляет запрос.	PGresult ***par_PQexec**(par_PGconn *conn, const char *query) Тиражирует запрос, вызывая PQexec() для каждого вычислительного узла.
void **PQfinish**(PGconn *conn) Завершает соединение с сервером PostgreSQL.	void **par_PQfinish**(par_PGconn *conn) Завершает соединения с узлами PargreSQL.

Таким образом, приложение устанавливает соединение сразу с несколькими узлами и отправляет каждому один и тот же запрос, после чего агрегирует результаты. Переход к новой библиотеке осуществляется прозрачно для приложения с помощью набора макросов, заменяющих вызовы функций оригинального интерфейса на вызовы соответствующих функций нового интерфейса.

Оператор exchange

Оператор exchange [8] служит для обмена кортежами между экземплярами параллельной СУБД PargreSQL. Он вставляется в план запроса подсистемой Parallelizer. Для каждого оператора exchange задан его уникальный номер в плане запроса, а также *функция обмена* ψ, вычисляющая для каждого кортежа номер процессорного узла, на который нужно передать данный кортеж. Таким образом, оператор exchange связан со всеми операторами exchange, находящимися в том же самом месте плана запроса на других процессорных узлах.

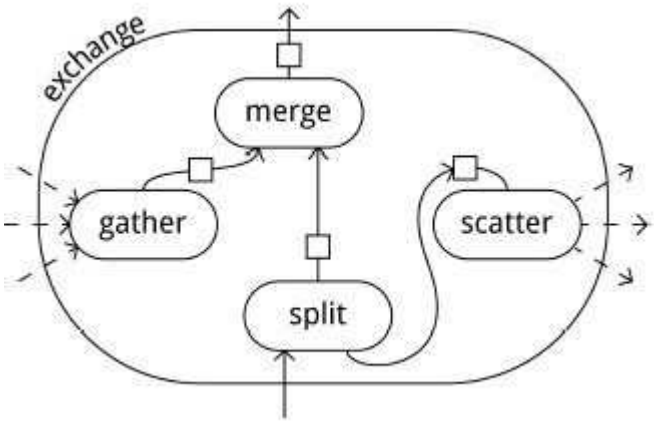

Рис. 4. Архитектура оператора exchange

Архитектура оператора exchange представлена на Рис. 4. Он состоит из четырех узлов: Merge, Split, Scatter и Gather. Данные узлы реализованы в соответствии с итераторной моделью, используемой в PostgreSQL. Она предполагает наличие у каждого узла в дереве плана запроса метода **next()**, который возвращает кортеж, вызывая в свою очередь метод **next()** у своих сыновей. Реализация метода **next()** для каждого из данных четырех узлов приведена на Рис. 5 и 6.

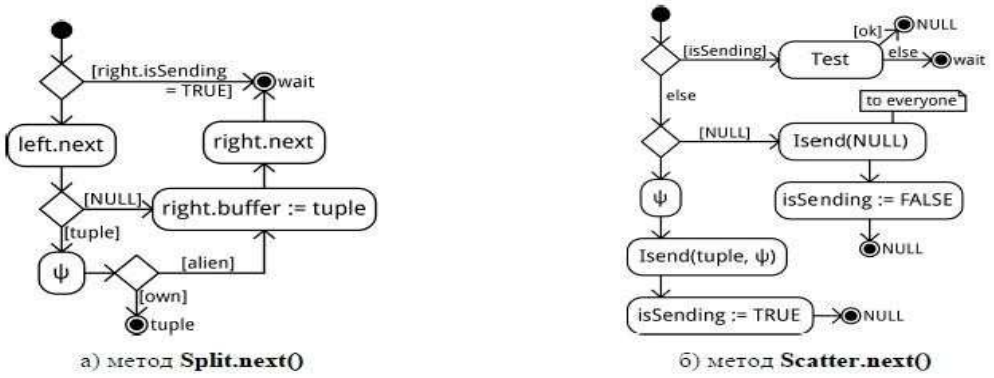

Рис 5. Реализация узлов Split и Scatter

Узел Split вычисляет функцию обмена для каждого поступающего кортежа и передает «свои» кортежи выше по плану (узлу Merge), а «чужие» — узлу Scatter, для дальнейшей отправки на нужный процессорный узел.

Узел Scatter вычисляет функцию обмена для каждого поступающего кортежа и отправляет их на процессорные узлы с соответствующим номером.

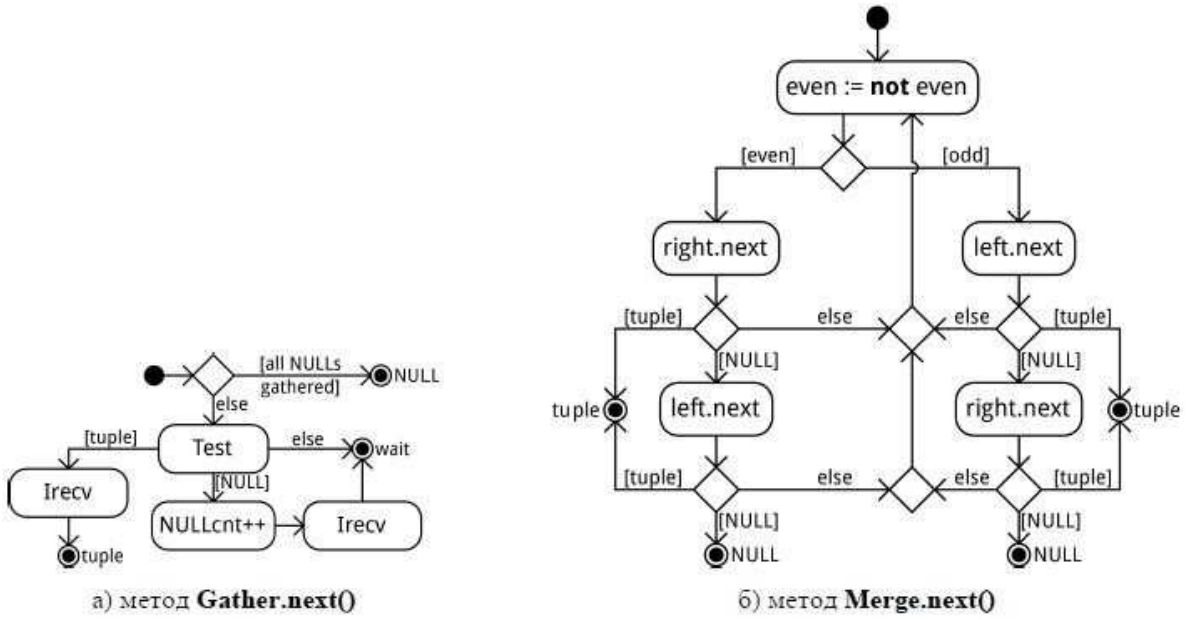

Рис 6. Реализация узлов Gather и Merge

Узел Gather выполняет получение кортежей от всех остальных процессорных узлов.

Узел Merge осуществляет слияние кортежей, поступающих от узлов Gather и Split. Кортежи, возвращаемые узлом Merge являются результатом оператора exchange. Таким образом, оператор exchange, вставленный в план, перераспределяет кортежи в соответствии с функцией обмена ψ.

Распараллеливание плана запроса

Параллелизатор плана запроса выполняет вставку операторов обмена в последовательный план запроса, тем самым превращая план в параллельный. Операторы обмена требуется вставлять между узлом Join и его поддеревьями, чтобы отношения, участвующие в соединении, стали фрагментированными по атрибуту соединения.

В PostgreSQL существует три вида операции соединения: HashJoin, MergeJoin и NestedLoop. Вставка оператора exchange для каждого вида операции выполняется особым образом (см. Рис. 7), поскольку данные операции опираются на некоторые предположения о своих аргументах:

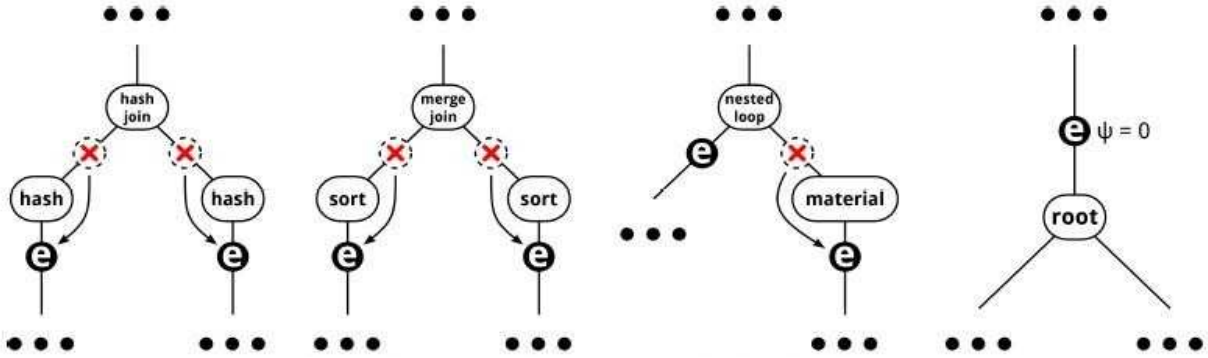

Рис. 7. Особые случаи при вставке оператора exchange

- Операция HashJoin предполагает, что оба аргумента являются узлами типа Hash, поэтому вставка оператора обмена производится под узлы Hash, а не над ними.
- Операция MergeJoin предполагает, что оба аргумента являются узлами типа Sort, поэтому вставка оператора обмена производится аналогично — на один уровень ниже.
- Операция NestedLoop предполагает, что правый аргумент является узлом типа Material, поэтому вставка оператора обмена в правом поддереве производится на один уровень ниже.

Перечисленные особые случаи реализуются следующим правилом. Если при вставке оператора обмена обнаруживается, что он будет вставлен над узлом Hash, Sort или Materialize, то вместо этого он вставляется на уровень глубже.

Особым случаем также является корень плана. В корень вставляется оператор exchange с функцией обмена, тождественно равной нулю. Такая функция позволяет собрать результирующее отношение на одном вычислительном узле.

Эксперименты

Над разработанной параллельной СУБД были поставлены эксперименты с использованием следующего запроса.

```
select * from t1 join t2 on t1.b = t2.a where t1.a % 10007 = 0;
```

Таблица **`t1`** состоит из 10^8 кортежей, таблица **`t2`** — из 10^7 кортежей. Общий объем данных в таком случае приблизительно равен 2 ГБ. Обе таблицы фрагментированы по атрибуту **`a`**. Таким образом, возникают обмены кортежами таблицы **`t1`**, поскольку в условии соединения фигурирует атрибут **`t1.b`**, а таблица фрагментирована по атрибуту **`t1.a`**.

Эксперименты проводились на узлах суперкомпьютера СКИФ Урал в Южно-Уральском государственном университете.

Результаты изображены на Рис. 8 в виде графика ускорения относительно оригинальной версии СУБД PostgreSQL.

Заключение

В данной работе представлена архитектура параллельной СУБД PargreSQL, реализация ее основных подсистем и результаты экспериментов. Разработанная система демонстрирует ускорение, близкое к линейному.

Дальнейшие исследования могут быть направлены на решение задачи балансировки загрузки, реализацию транзакций, вставки и обновления данных, а также обеспечение надежности системы на больших масштабах.

ЛИТЕРАТУРА:

1. M. Stonebraker, G. Kemnitz. The Postgres Next Generation Database Management System // Commun. ACM. 1991. Vol. 34, no. 10. Pp. 78–92.
2. L.D. Paulson. Open Source Databases Move into the Marketplace // IEEE Computer. Vol. 37. Issue. 7. Pp. 13–15. 2004.
3. N. Samokhvalov. XML Support in PostgreSQL // SYRCoDIS, volume 256 of CEUR Workshop Proceedings. 2007.
4. Y. Havinga, W. Dijkstra, A. de Keijzer. Adding HL7 version 3 data types to PostgreSQL // CoRR, abs/1003.3370, 2010.
5. D. Guliato, E.V. de Melo, R.M. Rangayyan, R.C. Soares. POSTGRESQL-IE: An Image-handling Extension for PostgreSQL // Journal of Digital Imaging, 22(2):149–165. 2009.
6. D.V. Levshin, A.S. Markov. Algorithms for integrating PostgreSQL with the semantic web // Programming and Computer Software, 35(3):136–144. 2009.
7. D.J. DeWitt, J. Gray. Parallel Database Systems: The Future of High Performance Database

Systems // Commun. ACM. 1992. Vol. 35, no. 6. Pp. 85–98.
8. Л.Б. Соколинский. Организация параллельного выполнения запросов в многопроцессорной машине баз данных с иерархической архитектурой // Программирование. 2001. No. 6. С. 13–29.