Министерство образования и науки Российской Федерации
Государственное образовательное учреждение высшего профессионального образования
”ЮЖНО-УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”
Факультет Вычислительной математики и информатики
Кафедра системного программирования

Рецензент
доктор. физ.-мат. наук, профессор
_____ Л.Б. Соколинский
“____”_____ 2011 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Зав. кафедрой СП
_____ Л.Б. Соколинский
“____”_____ 2011 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
на соискание академической степени магистра прикладной математики и информатики по направлению 010500.68 ”Прикладная математика и информатика” (магистерская программа ”Системное программирование”)

Интеграция параллелизма в СУБД PostgreSQL

Ученый секретарь
_____ М.Л. Цымблер
“____”_____ 2011 г.

Научный руководитель
кандидат физ.-мат. наук, доцент
_____ М.Л. Цымблер

Автор работы
студент группы ВМИ-248
_____ К.С. Пан

Челябинск-2011

Министерство образования и науки Российской Федерации
Государственное образовательное учреждение высшего профессионального образования
”ЮЖНО-УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”
Факультет Вычислительной математики и информатики
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП

_____ Л.Б. Соколинский

“____”_____ 2011 г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы магистра студенту группы ВМИ-248 Пану Константину Сергеевичу, обучающемуся в магистратуре по направлению 010500.68 ”Прикладная математика и информатика” (магистерская программа ”Системное программирование”)

1. **Тема работы** (утверждена приказом ректора от 22.03.2011 № 514)
   Интеграция параллелизма в СУБД PostgreSQL
2. **Срок сдачи студентом законченной работы:** 05.06.2011.
3. **Исходные данные к работе**
   - PostgreSQL Source Code Documentation.
     URL: `http://doxygen.postgresql.org/` (дата обращения: 05.05.2011)
   - Справочник по функциям СУБД Омега.
     URL: `http://omega.susu.ru/prototype/guider/` (дата обращения: 05.05.2011)
   - *Лепихов А.В., Соколинский Л.Б.* Обработка запросов в СУБД для кластерных систем // Программирование. 2010. № 4. С. 25–39.
4. **Перечень подлежащих разработке вопросов**
   4.1. Изучить архитектуру СУБД PostgreSQL.
   4.2. Разработать архитектуру и принципы реализации параллельной СУБД PargreSQL на основе внедрения параллелизма в СУБД PostgreSQL.
   4.3. Выполнить проектирование и реализацию следующих основных подсистем параллельной СУБД PargreSQL: оператор обмена, подсистему тиражирования запросов, параллелизатор плана запроса.
   4.4. Разработать набор тестов и провести эксперименты для исследования эффективности разработанной параллельной СУБД.
5. **Дата выдачи задания:** 08.02.2011.

Научный руководитель
доцент каф. системного
программирования ЮУрГУ
кандидат физ.-мат. наук, доцент М.Л. Цымблер

Задание принял к исполнению К.С. Пан

Оглавление

Введение

Актуальность проблемы

В настоящее время все большую популярность обретают задачи, связанные с обработкой сверхбольших данных. Примерами таких задач являются электронная коммерция, научные базы данных, социальные сети, индексирование информации в сети Интернет и др.

Для эффективного решения подобных задач необходимо использовать параллельные СУБД, обеспечивающие параллельную обработку запросов на многопроцессорных вычислительных системах. Среди современных параллельных СУБД существует множество коммерческих продуктов: DB2 Parallel Edition, NonStop SQL, Teradata, Greenplum и др. Однако данные системы имеют экстремально высокую стоимость и не подходят для массового использования.

Свободные СУБД, например, PostgreSQL или MySQL, являются надежной альтернативой коммерческим СУБД [1, 2, 3]. В силу открытости исходного кода свободных СУБД, в них можно внедрить параллельную обработку запросов, что позволит получать эффективные и относительно недорогие параллельные системы баз данных, обладающие хорошей масштабируемостью. Подобное пока никем не реализовано, поэтому задача внедрения параллелизма в свободные СУБД *актуальна*.

Цель и задачи

СУБД PostgreSQL [4] представляет собой свободную СУБД, имеющую исчерпывающе документированный исходный код. Цель данной работы состоит во внедрении фрагментного параллелизма [5] в СУБД PostgreSQL на основе методов параллельной обработки запросов, разработанных в рамках проекта Омега [6, 7]. Для этого необходимо решить следующие задачи:

1. Изучить архитектуру СУБД PostgreSQL.
2. Разработать архитектуру и принципы реализации параллельной

СУБД PargreSQL на основе внедрения параллелизма в СУБД PostgreSQL.

3. Выполнить проектирование и реализацию основных подсистем параллельной СУБД PargreSQL.

4. Разработать набор тестов и провести эксперименты для исследования эффективности разработанной параллельной СУБД.

Структура и объем работы

Работа состоит из введения, четырех основных разделов, заключения и библиографии. Объем работы составляет 33 страницы, объем библиографии — 23 наименования.

Обзор литературы

Существует достаточно большое количество практических приложений баз данных на основе PostgreSQL и исследовательских проектов, посвященных расширению и улучшению PostgreSQL.

В работе [8] рассматривается внедрение поддержки XML в PostgreSQL. Добавление новых типов данных для обеспечения поддержки стандарта обмена медицинской информацей HL7 в PostgreSQL описывается в [9]. Авторы работы [10] предлагают расширение PostgreSQL для обработки изображений. В работе [11] представлен подход к интеграции PostgreSQL с Semantic Web.

Исследования, посвященные использованию PostgreSQL для параллельной обработки запросов, представлены следующими работами.

В [12] предлагается расширение PostgreSQL для распределенной обработки запросов. В данном расширении не предусмотрена фрагментация таблиц, а распределенная обработка запросов достигается путем хранения разных таблиц в разных произвольных источниках данных, обращение к которым реализуется с помощью специальных функций-оберток.

СУБД ParGRES [13] представляет собой промежуточное программное обеспечение (middleware) с открытым исходным кодом для высокопро-

изводительной обработки OLAP-запросов с использованием PostgreSQL. ParGRES реализует внутризапросный параллелизм на кластерных вычислительных системах и адаптивное виртуальное распределение базы данных. Аналогичным продуктом является pgpool [14], реализующий прозрачный и для клиентов, и для серверов пул соединений. Pgpool обеспечивает репликацию базы данных, балансировку загрузки путем отправки разных запросов на выборку разным серверам, а также параллельное выполнение запросов.

СУБД GParGRES [15] 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 уровнях: на уровне грид (реализовано в GParGRES) и на уровне узлов (реализовано в ParGRES).

В данной работе использованы методы параллельной обработки запросов [16, 6, 7], разработанные в рамках проекта Омега [17].

В настоящее время отсутствуют параллельные СУБД на основе последовательных СУБД с открытым исходным кодом.

Содержание работы

В первом разделе, «Архитектура PostgreSQL», представлен обзор архитектуры свободной СУБД с открытым исходным кодом PostgreSQL.

Второй раздел, «Архитектура PargreSQL», посвящен описанию архитектуры параллельной СУБД PargreSQL.

Третий раздел, «Реализация PargreSQL», раскрывает детали реализации параллельной СУБД PargreSQL.

Четвертый раздел, «Вычислительные эксперименты», содержит описание вычислительных экспериментов и их результаты.

В заключении суммируются основные результаты работы.

1. Архитектура СУБД PostgreSQL

В данном разделе представлена архитектура СУБД PostgreSQL.

1.1. Клиент-серверное взаимодействие

В основе архитектуры PostgreSQL лежит модель «клиент-сервер». В сеансе работы с СУБД PostgreSQL участвуют три вида взаимодействующих процессов (см. Рис. 1): приложение-клиент (*frontend*), серверный процесс (*backend*) и демон (*daemon*).

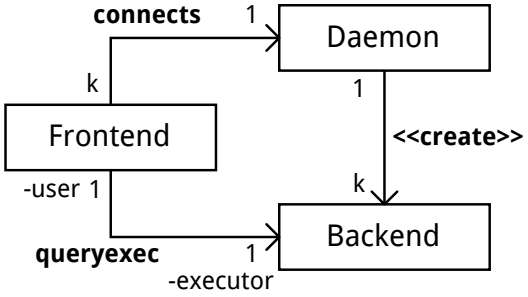

Рис. 1. Процессы СУБД PostgreSQL

Демон осуществляет прием соединений, устанавливаемых клиентами, и создает отдельный серверный процесс для обработки запросов каждого отдельного клиента.

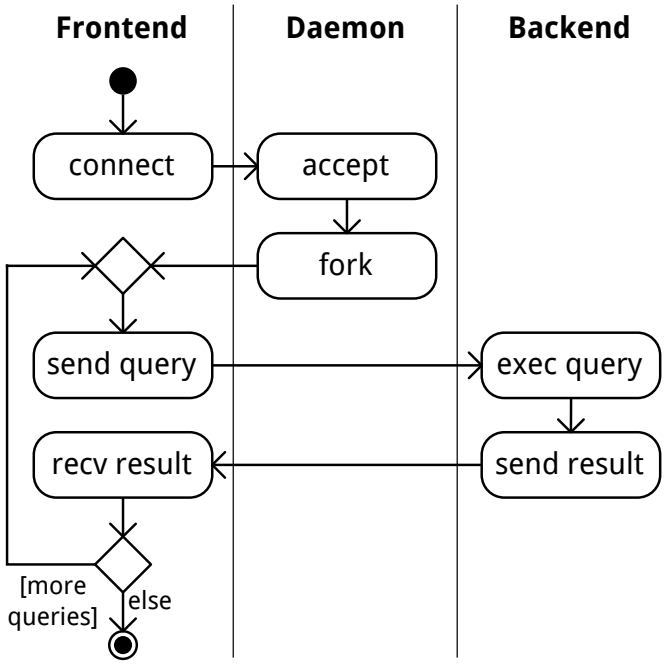

Рис. 2. Взаимодействие клиента и сервера PostgreSQL

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.

1.2. Этапы обработки запроса

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.

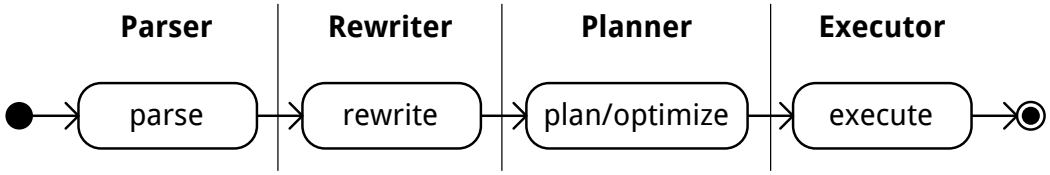

Рис. 3. Обработка запроса в СУБД PostgreSQL

1.3. Подсистемы

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).

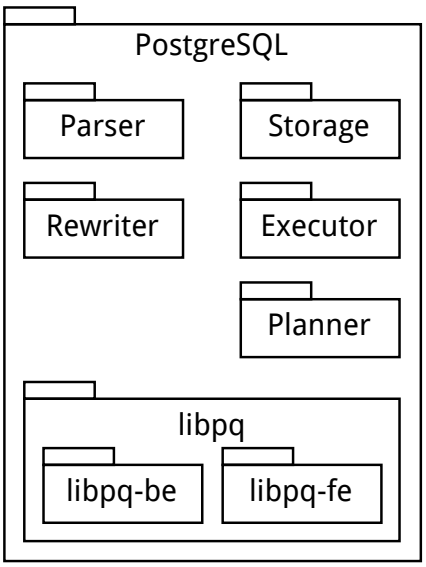

Рис. 4. Подсистемы СУБД PostgreSQL

1.4. Размещение компонентов

Размещение компонентов СУБД PostgreSQL приведено на Рис. 5.

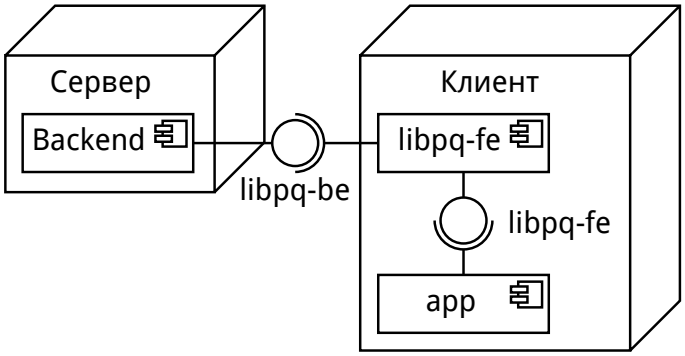

Рис. 5. Размещение компонентов PostgreSQL

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

2. Архитектура СУБД PargreSQL

В данном разделе представлена архитектура разработанной системы PargreSQL, описаны изменения в процедуре обработки запроса, новые подсистемы, их взаимодействие и размещение.

2.1. Фрагментный параллелизм

PargreSQL использует идею фрагментного параллелизма [16]. Общая схема параллельной обработки запроса представлена на Рис. 6.

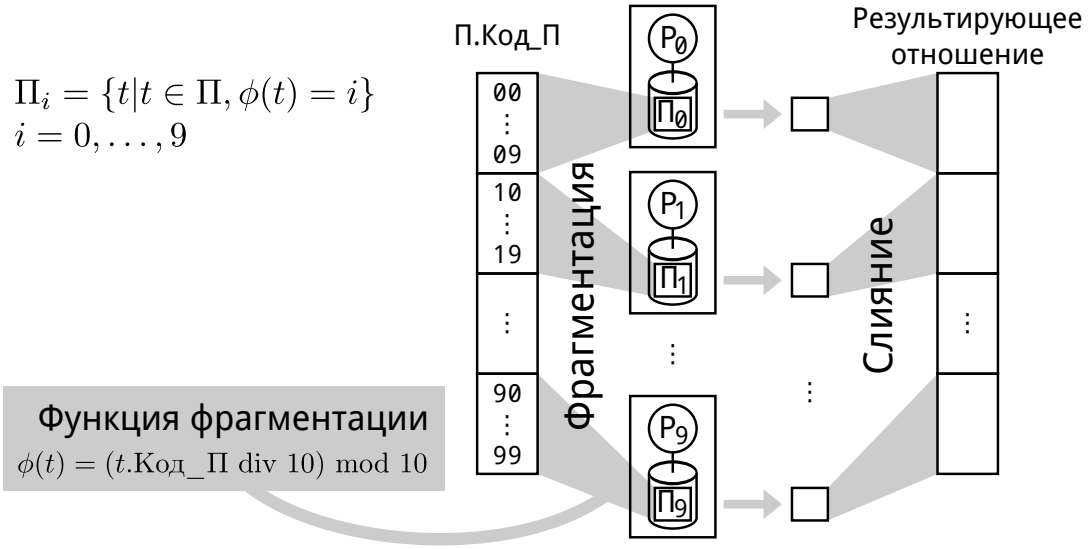

Рис. 6. Параллельная обработка запроса на основе фрагментного параллелизма

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 (*агентов*), каждый из которых обрабатывает отдельный фрагмент отношения. Полученные фрагменты сливаются в результирующее отношение.

2.2. Клиент-серверное взаимодействие

Архитектура клиент-серверного взаимодействия параллельной СУБД PargreSQL, в отличие от последовательной СУБД PostgreSQL, предпола-

гает, что клиент взаимодействует с двумя или более серверами одновременно (см. Рис. 7).

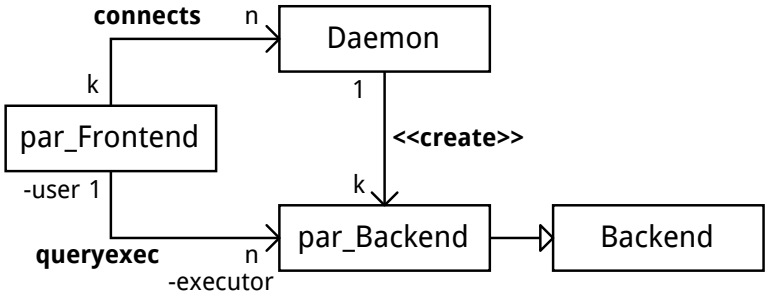

Рис. 7. Процессы СУБД PargreSQL

Порядок взаимодействия клиентского приложения и СУБД PargreSQL представлен на Рис. 8. Клиентское приложение подключается сразу ко всем экземплярам СУБД и отправляет им одинаковый запрос, после чего получает от них и агрегирует результаты.

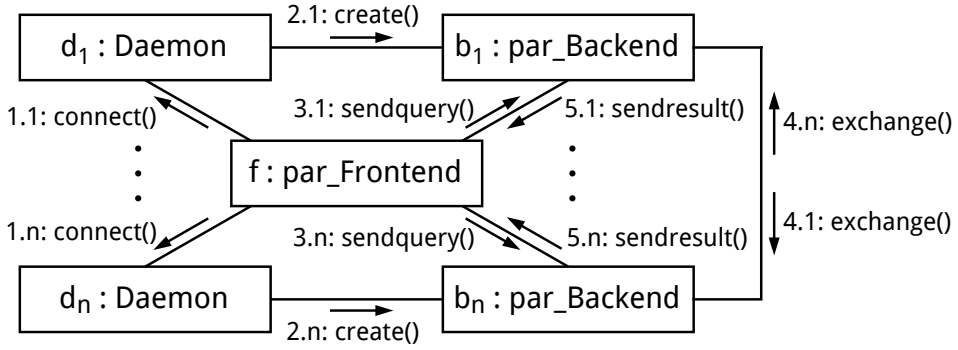

Рис. 8. Взаимодействие клиента и серверов PargreSQL

2.3. Этапы обработки запроса

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

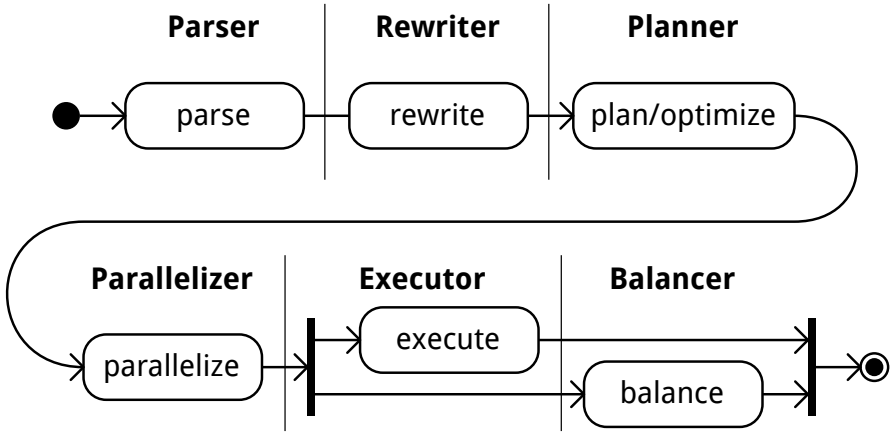

Рис. 9. Обработка запроса в СУБД PargreSQL

2.4. Подсистемы

Архитектура PargreSQL представлена на Рис. 10. PostgreSQL является подсистемой в рамках системы PargreSQL.

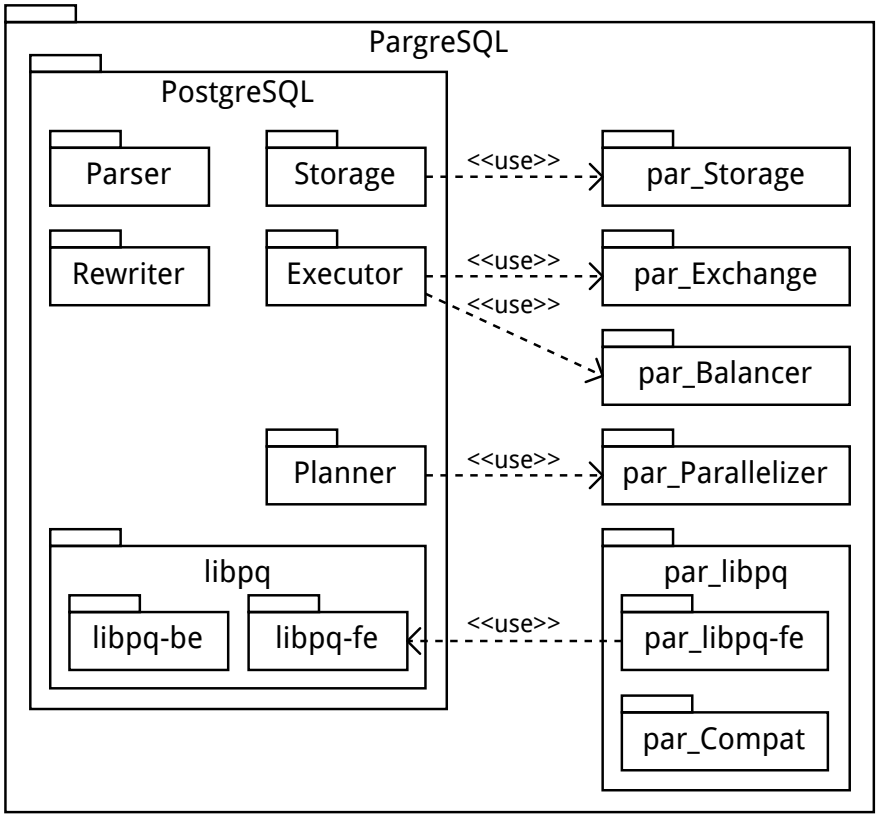

Рис. 10. Архитектура СУБД PargreSQL

Для разработки PargreSQL необходимо внести изменения в исходные тексты следующих подсистем PostgreSQL: Storage, Executor и Planner. Данные изменения обеспечивают внедрение следующих новых подсистем:

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

- *par_Exchange* — подсистема, реализующая оператор *exchange* [16], 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. Подробнее об операторе *exchange* см. раздел 3.3.

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`.

2.5. Размещение компонентов

Размещение компонентов СУБД PargreSQL приведено на Рис. 11.

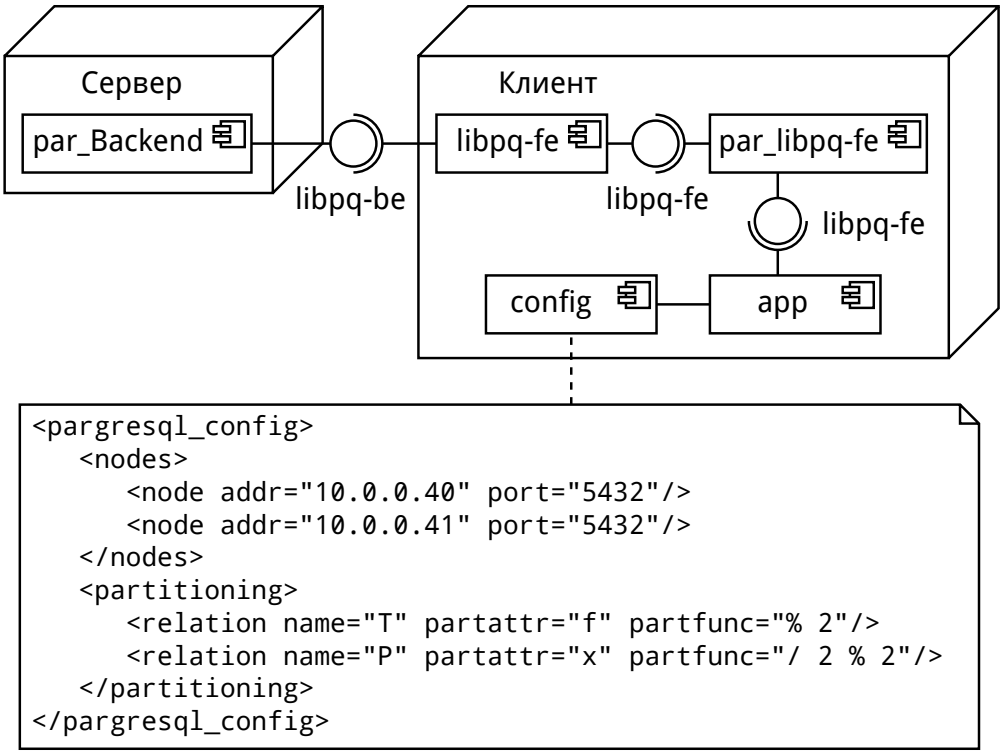

Рис. 11. Размещение компонентов PargreSQL

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 кластера, фрагментация таблиц и др.). Остальные компоненты размещаются на узлах сервера.

3. Реализация СУБД PargreSQL

В данном разделе изложены детали реализации основных подсистем СУБД PargreSQL.

3.1. Подсистема тиражирования

Подсистема `par_libpq` служит для тиражирования запросов на все вычислительные узлы, на которых запущена СУБД PargreSQL.

3.1.1. API

Подсистема `par_libpq` состоит из двух частей (см. Рис. 12): библиотеки `par_libpq-fe` и набора макросов `par_Compat`.

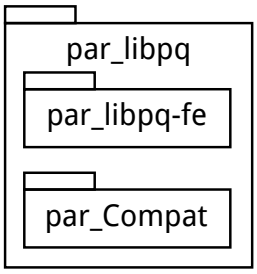

Рис. 12. Структура подсистемы `par_libpq`

Библиотека `par_libpq-fe` используется в приложении для доступа к СУБД. В ней определяются структуры данных и функции, необходимые приложению для работы с параллельной СУБД PargreSQL. Библиотека `par_libpq-fe` имеет интерфейс, аналогичный интерфейсу `libpq-fe`.

В Табл. 1 приведены отличия интерфейсов библиотек `par_libpq-fe` и `libpq-fe`.

Табл. 1. Изменения в API `libpq-fe`

libpq-fe	par_libpq-fe
`struct PGconn` Хранит данные о соединении с сервером PostgreSQL.	`struct par_PGconn` 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 PQstatus(`
`  const PGconn *conn`
`)`
Возвращает статус соединения с сервером PostgreSQL.
	`ConnStatusType`
	`par_PQstatus(`
	`  const par_PGconn *conn`
	`)`
	Возвращает статус соединения с узлами PargreSQL. Результат получается с помощью агрегации результатов, полученных от вызова `PQstatus()` для всех вычислительных узлов.

`PGresult *PQexec(`
`  PGconn *conn,`
`  const char *query`
`)`
Отправляет запрос.
	`PGresult *par_PQexec(`
	`  PGconn *conn,`
	`  const char *query`
	`)`
	Тиражирует запрос, вызывая `PQexec()` для каждого узла.

`void PQfinish(`
`  PGconn *conn`
`)`
Завершает соединение с сервером.
	`void par_PQfinish(`
	`  par_PGconn *conn`
	`)`
	Завершает соединения с узлами PargreSQL.

3.1.2. Реализация

Диаграмма классов, реализующих `par_libpq-fe`, показана на Рис. 13.

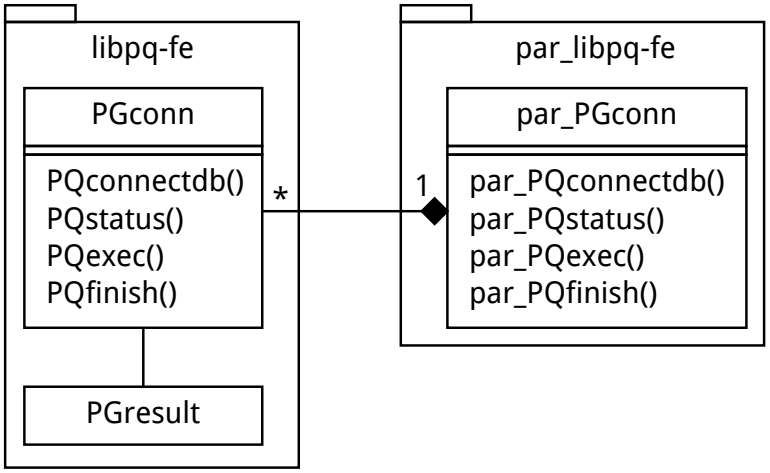

Рис. 13. Классы, реализующие `par_libpq-fe`

3.1.3. Прозрачность (par_Compat)

Набор макросов `par_Compat` служит для того, чтобы упростить подключение библиотеки `par_libpq-fe`, сведя его к изменению одной строки в исходном коде приложения. Таким образом, подсистема `par_Compat` позволяет использовать библиотеку `par_libpq-fe` с интерфейсом оригинальной библиотеки `libpq-fe`, чтобы изменения в коде приложения при переходе на PargreSQL было минимальным.

На Рис. 14 в качестве примера приведена часть реализации подсистемы `par_Compat`.

```
#define PGconn par_PGconn
#define PQconnectdb(X) par_PQconnectdb()
#define PQfinish(X) par_PQfinish(X)
#define PQstatus(X) par_PQstatus(X)
#define PQexec(X,Y) par_PQexec(X,Y)
```

Рис. 14. Пример реализации `par_Compat`

3.2. Подсистема обмена сообщениями

Подсистема обмена сообщениями служит для обмена кортежами между серверными процессами (*backend*), выполняющими параллельно общий запрос.

Одной из наиболее широко применяемых технологий обмена сообщениями для параллельного программирования является MPI [18], но использовать данный интерфейс для реализации PargreSQL невозможно, поскольку серверные процессы порождаются динамически. В реализации СУБД PargreSQL используется сторонняя разработка [19] для организации обмена сообщениями.

Данная подсистема состоит из двух частей: коммуникатора и библиотеки. Коммуникатор, представляя собой MPI-программу, запускается в виде независимого от СУБД демона в одном экземпляре на каждом вычислительном узле. Серверные процессы СУБД PargreSQL, используя библиотеку, подключаются через общую память к локальному коммуникатору и отправляют/принимают через него сообщения.

Интерфейс библиотеки обмена сообщениями состоит из следующих основных функций

- `INIS_Init()` открывает доступ к разделяемой памяти.
- `INIS_Finalize()` завершает работу с разделяемой памятью.
- `INIS_GetNode()` возвращает номер текущего узла.
- `INIS_GetNodesCount()` возвращает общее количество узлов.
- `INIS_Isend(dst, port, size, buf, request)` начинает асинхронную отправку сообщения размера `size` из буфера `buf` на узел `dst` и порт `port`. Параметр `port` позволяет отличать, какому оператору exchange в каком из серверных процессов отправляется сообщение. В общем случае на вычислительном узле запущено много серверных процессов, а в серверном процессе обрабатывается много операторов exchange. В параметр `request` сохраняется иден-

тификатор операции отправки, используемый позже для того, чтобы отследить окончание отправки.

- `INIS_Irecv(src, port, size, buf, request)` начинает асинхронное получение сообщения максимального размера `size` в буфер `buf` с узла `dst` на порт `port`.

- `INIS_Test(request, flag)` выполняет проверку состояния операции, ассоциированной с идентификтором `request`. Если соответствующая операция выполнена, в параметре `flag` возвращается 1, иначе — 0.

3.3. Оператор обмена (exchange)

В данном разделе представлена архитектура и реализация оператора *exchange* в параллельной СУБД PargreSQL.

Оператор *exchange* [16] служит для обмена кортежами между экземплярами параллельной СУБД PargreSQL. Оператор *exchange* вставляется в план запроса подсистемой Parallelizer с целью обеспечения корректности результатов запроса.

Каждый оператор обмена обладает двумя характеристиками: *функция обмена* и *порт*. Порт является уникальным числом и обозначает место оператора обмена в плане среди других операторов обмена. Функция обмена ставит каждому кортежу в соответствие номер узла, на который необходимо передать данный кортеж.

3.3.1. Архитектура оператора exchange

Архитектура оператора *exchange* представлена на Рис. 15.

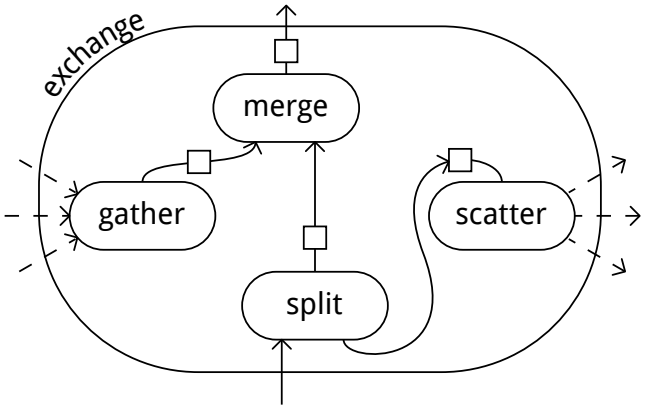

Рис. 15. Архитектура оператора exchange

Оператор Split, применяя функцию обмена, разделяет кортежи на две категории: «свои» и «чужие» — и передает «чужие» кортежи оператору Scatter для рассылки, а «свои» — оператору Merge для продвижения выше по плану.

Оператор Scatter принимает кортежи от оператора Split и рассылает их на соответствующие им вычислительные узлы.

Оператор Gather выполняет обратное — получает кортежи от других вычислительных узлов, отправленные на тот же *порт*, и передает их оператору Merge для продвижения выше по плану.

Оператор Merge выполняет слияние кортежей, полученных от операторов Split и Gather.

Далее описаны новые типы данных и функции, добавленные в оригинальную СУБД PostgreSQL для реализации оператора *exchange*.

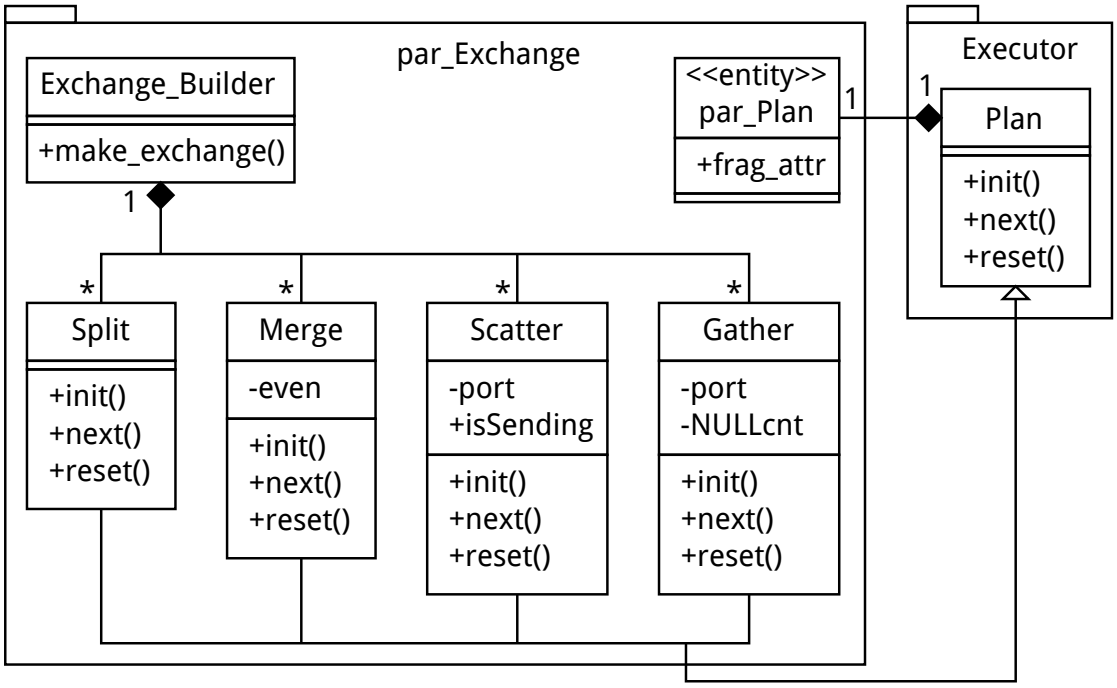

Рис. 16. Диаграмма классов оператора exchange

На Рис. 16 показана диаграмма классов UML, иллюстрирующая новые классы (собраны в пакете par_Exchange). На данной диаграмме показаны собственно классы Merge, Split, Scatter и Gather, реализующие одноименные узлы оператора exchange, а также строитель Exchange_Builder, единственный метод которого служит для создания перечисленных четырех узлов и формирования из них цельного оператора exchange.

В класс Plan — общий предок всех узлов плана запроса — добавляется атрибут `frag_attr`, для того, чтобы хранить атрибут фрагментации отношений.

Далее подробно рассмотрены структуры данных и функции, реализующие данные классы.

3.3.2. Алгоритмы

В данном разделе с помощью диаграмм деятельности UML на Рис. 17, 18, 19 и 20 изложены алгоритмы, реализующие метод `next` узлов опера-

тора *exchange*.

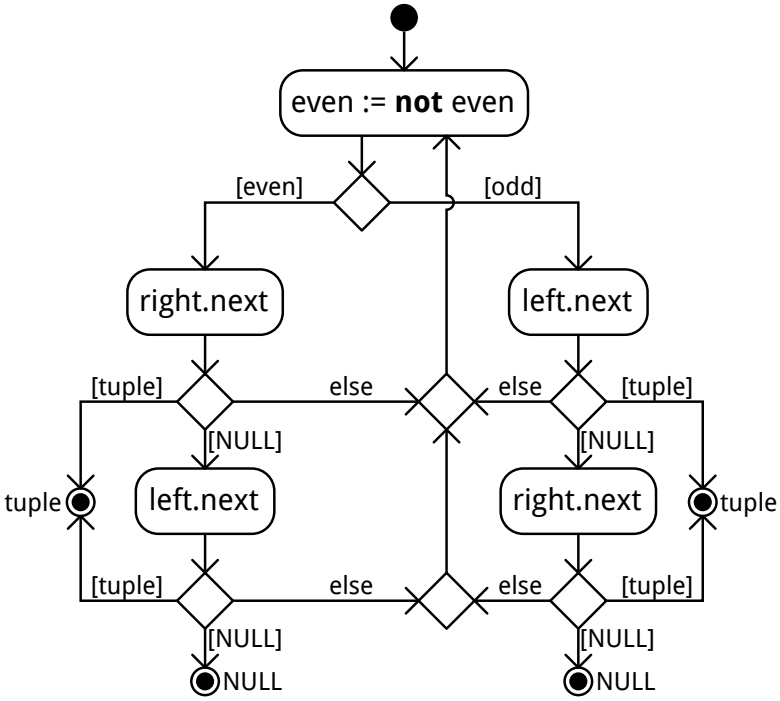

Рис. 17. Метод `next` узла Merge

Оператор Merge выполняет слияние кортежей, попеременно вызывая метод next() у своих дочерних операторов — Split и Gather.

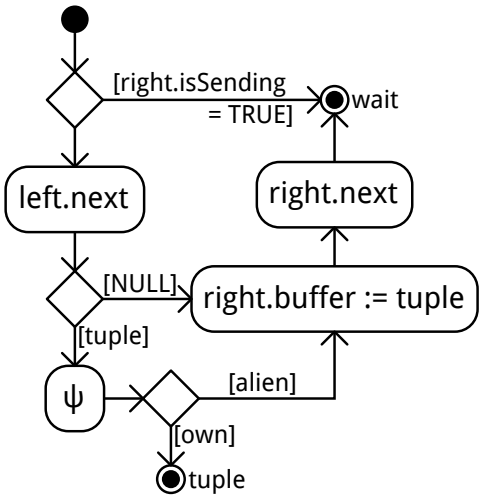

Рис. 18. Метод `next` узла Split

Оператор Split вызывает метод next() дочернего узла и к полученному кортежу применяет функцию обмена, которая возвращает число. Если данное число совпадает с номером текущего узла, то кортеж возвращается в качестве результата вызова метода. Иначе кортеж передается правому дочернему оператору (Scatter), после чего вызывается метод next() оператора Scatter и возвращается значение WAIT.

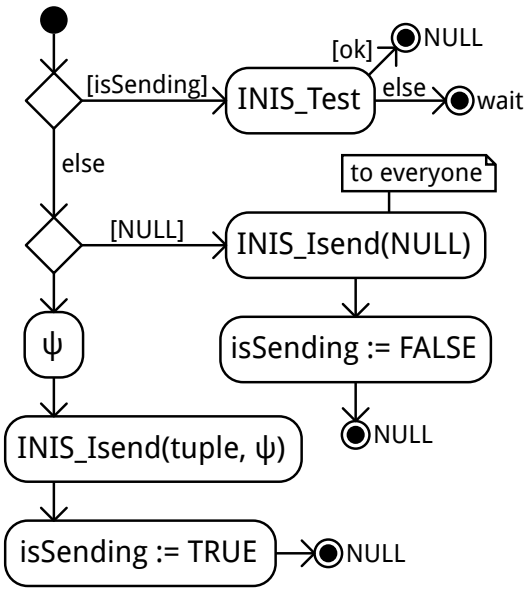

Рис. 19. Метод `next` узла Scatter

Оператор Scatter не имеет дочерних операторов, вызов его метода next() инициирует отправку кортежа, переданного ему от родительского оператора (Split), на тот вычислительный узел, номер которого совпадает с результатом применения функции обмена к кортежу. Если во время вызова метода next() до сих пор отправляется кортеж, то возвращается значение WAIT.

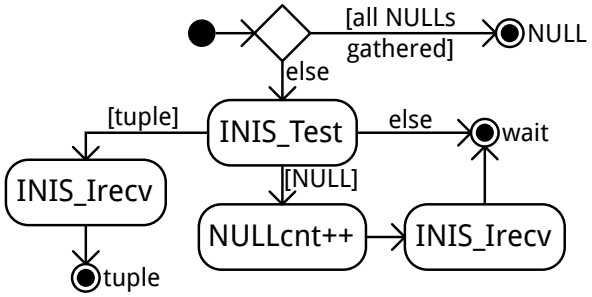

Рис. 20. Метод `next` узла Gather

Оператор Gather инициирует получение кортежей от всех вычислительных узлов. При вызове метода next() данного оператора проверяются статус операций получения, и если получен кортеж от некоторого узла, то инициируется новая операция получения от данного узла, а полученный кортеж возвращается в качестве результата. Если от всех кортежей получено значение NULL вместо кортежа, значит отношение закончилось и метод возвращает NULL как признак конца отношения.

3.3.3. Реализация

Новые типы данных

В файле `src/include/nodes/nodes.h` определен тип `enum NodeTag`. Первым элементом структуры, представляющей узел плана, идет значение данного типа. Оно служит для того, чтобы отличать узлы друг от друга. Для реализации оператора exchange нужно расширить тип новыми значениями — `T_Merge`, `T_Split`, `T_Scatter` и `T_Gather` — как показано на Рис. 21.

```
typedef enum NodeTag
{
        ... // старые значения

        T_Merge, // новые значения
        T_Split,
        T_Scatter,
        T_Gather
} NodeTag;
```

Рис. 21. Добавление новых значений в enum NodeTag

Файл `src/include/nodes/plannodes.h` содержит определения структур, представляющих узлы плана, генерируемого планировщиком.

Первым элементом структуры указывается либо структура Plan, либо структура, представляющая другой узел, который в свою очередь прямо или косвенно основан на Plan. Plan же содержит первым элементом значение типа NodeTag. Таким образом, все узлы плана первым элементом содержат значение типа NodeTag, по которому узлы можно различать.

Для реализации оператора exchange в этом файле необходимо определить собственные структуры `Merge`, `Split`, `Scatter` и `Gather`, как показано на Рис. 22.

```
typedef struct Merge {
        Plan        plan;
} Merge;
```

Рис. 22. Пример структуры, описывающей узел Merge

Перед выполнением узла Plan исполнитель создает другую структуру, состоящую из узлов типа PlanState или его потомков. Данная структура полностью повторяет оригинальный план, сгенерированный планировщиком, за исключением того, что в узлах типа PlanState хранится конкретное состояние исполнителя запроса. Такая мера нужна для того, чтобы план можно было повторно использовать.

Тип PlanState и его потомки определены в файле `src/include/nodes/execnodes.h`. Соответственно, в данный файл добавляются определения структур `MergeState`, `SplitState`, `ScatterState` и `GatherState`.

Новые функции

Файл `src/backend/optimizer/plan/createplan.c` содержит определения функций, создающих узлы плана. Это конструкторы для узлов.

В данном файле необходимо определить соответствующие функции для создания узлов `Merge`, `Split`, `Scatter` и `Gather`, как показано на Рис. 23.

```
Merge *create_merge_plan(...) {
        // создать структуру Merge
        // присвоить ее элементам значения
}
```

Рис. 23. Пример структуры, описывающей узел Merge

Файл `src/backend/executor/execProcnode.c` содержит определения абстрактных методов узла плана: `ExecInitNode`, `ExecProcNode`, `ExecEndNode`.

В данном файле необходимо дополнить указанные абстрактные методы, добавив в них новые `case` для новых значений `NodeType` — `T_Merge`, `T_Split`, `T_Scatter` и `T_Gather`.

Файлы `src/backend/executor/execMerge.{h,c}` содержат реализацию конкретных методов узлов. Для реализации узла (например, *Merge*), нужно определить там функции, как показано на Рис. 24.

```
PlanState *ExecInitMerge(Plan *node, EState *estate,
        int eflags);
TupleTableSlot *ExecMerge(PlanState *node);
void ExecEndMerge(PlanState *node);
```

Рис. 24. Конкретные методы узла Merge

3.4. Параллелизатор плана запроса

Параллелизатор (Parallelizer) служит для распараллеливания плана запроса путем вставки операторов *exchange* в нужные места плана. Далее представлен интерфейс подсистемы Parallelizer и принципы ее реализации.

3.4.1. API

Интерфейс подсистемы Parallelizer состоит из одного метода — `Plan *parallelize(Plan *plan)` — который распараллеливает план `plan`. Данный метод инкапсулирует в себе всю логику параллелизатора.

3.4.2. Реализация

Метод `parallelize` реализован в виде рекурсивного обхода дерева плана для поиска узлов соединения (Join) и вставки оператора exchange.

Оператор exchange вставляется между узлом Join и его сыном в случае несовпадения атрибута фрагментации сына с атрибутом, участвующим в условии соединения [16]. Однако в случае PostgreSQL необходимо учитывать специфику реализации соединения.

В СУБД PostgreSQL реализовано три типа соединения: соединение вложенными циклами (NestLoop), слиянием (MergeJoin) и хешированием (HashJoin). Вставка оператора exchange для каждого типа соединения выполняется особым образом (см. Рис. 25), поскольку данные типы операции соединения опираются на некоторые предположения о своих аргументах:

- Операция HashJoin предполагает, что для отношений, участвующих в соединении, сформированы хеш-таблицы с помощью операции Hash. То есть операция HashJoin предполагает, что оба дочерних узла

являются узлами типа Hash. Вставка оператора обмена между узлами HashJoin и Hash сделает вычисленную хеш-таблицу ошибочной и приведет к неверным результатам. Поэтому вставка оператора обмена производится на один уровень глубже — между узлом Hash и его поддеревом.

- Операция MergeJoin предполагает, что отношения, участвующие в соединении, упорядочены с помощью операции Sort, то есть оба дочерних узла являются узлами типа Sort. Выполнение обменов после сортировки нарушит порядок кортежей, поэтому вставка оператора обмена производится аналогично — на один уровень глубже.

- Операция NestedLoop предполагает, что правое отношение полностью считано и располагается в памяти для возможности многократного прохода всего отношения внутренним циклом. Вставка узла Exchange между узлом NestedLoop и Material, в общем случае, приведет к взаимной блокировке серверных процессов из-за разного количества итераций во внешнем цикле. Поэтому вставка оператора обмена в правом поддереве производится на один уровень глубже.

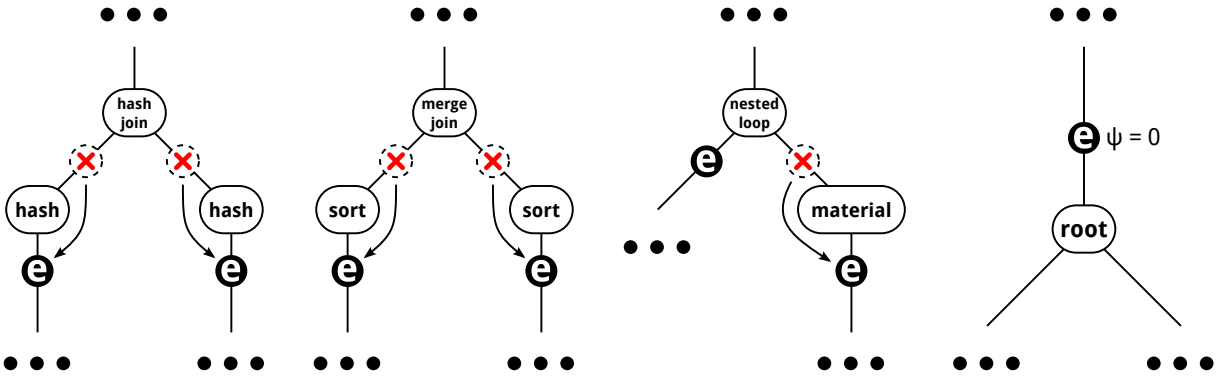

Рис. 25. Вставка оператора exchange

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.

4. Вычислительные эксперименты

В данном разделе описаны вычислительные эксперименты, исследующие эффективность разработанной параллельной СУБД.

Аппаратной платформой для экспериментов послужил вычислительный кластер СКИФ Урал [20] в Южно-Уральском государственном университете, имеющий по два процессора Intel Xeon E5472 на узле и системную сеть InfiniBand (20 Гбит/с).

Входными данными являлись таблицы `t1` и `t2`, структура которых приведена в виде SQL-запросов на Рис. 26.

```
CREATE TABLE t1 (
        a INTEGER,
        b INTEGER
);
CREATE TABLE t2 (
        a INTEGER,
        b INTEGER
);
```

Рис. 26. Структура таблиц `t1` и `t2`

Таблицы `t1` и `t2` содержали соответственно 10^8 и 10^7 кортежей, сгенерированных случайно по закону равномерного распределения. Такой объем данных эквивалентен 2 ГБ. Атрибутами фрагментации были `t1.a` и `t2.a`.

К данным таблицам в ходе экспериментов применялся запрос, код которого показан на Рис. 27.

```
SELECT *
FROM t1 JOIN t2
        ON t1.b = t2.a
WHERE t1.a % 10007 = 0;
```

Рис. 27. Запрос к таблицам `t1` и `t2`

Из кода запроса видно, что атрибут фрагментации отношения `t1` (`a`), не совпадает с атрибутом в условии соединения (`b`), из-за чего перед соединением возникают пересылки кортежей.

Результаты экспериментов представлены на Рис. 28 в виде графика ускорения параллельной СУБД PargreSQL относительно последовательной СУБД PostgreSQL.

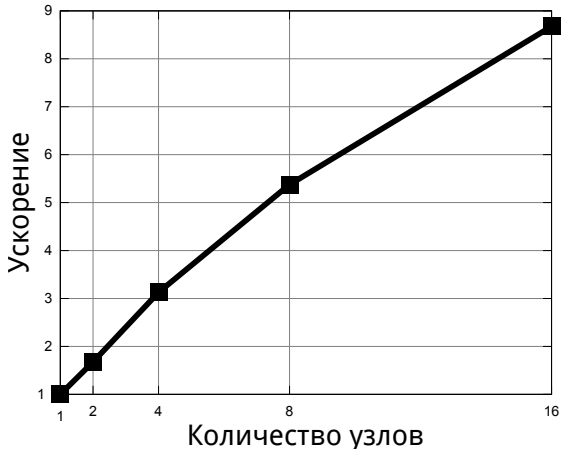

Рис. 28. График ускорения PargreSQL относительно PostgreSQL

Таким образом, СУБД PargreSQL демонстрирует ускорение, близкое к линейному.

Заключение

Данная работа посвящена разработке параллельной СУБД PargreSQL для многопроцессорных вычислительных систем с кластерной архитектурой на основе свободной СУБД PostgreSQL и принципа фрагментного параллелизма.

Публикации по теме работы

Результаты работы опубликованы автором в следующих изданиях:

- *Pan C.* Development of a Parallel DBMS on the Basis of PostgreSQL // Proceedings of the Seventh Spring Researchers' Colloquium on Databases and Information Systems (SYRCoDIS'2011). Moscow: Moscow State University, 2011. P. 57–61. [21]

- *Пан К.С., Цымблер М.Л.* Архитектура и принципы реализации параллельной СУБД PargreSQL // Параллельные вычислительные технологии (ПаВТ'2011): труды международной научной конференции (Москва, 28 марта – 1 апреля 2011 г.). Челябинск: Издательский центр ЮУрГУ, 2011. С. 577–584. [22]

- *Пан К.С., Цымблер М.Л.* Проект PargreSQL: разработка параллельной СУБД на основе свободной СУБД PostgreSQL // Научный сервис в сети Интернет: суперкомпьютерные центры и задачи: Труды международной научной конференции (Новороссийск, 20–25 сентября 2010 г.). М.: Изд-во МГУ, 2010. С. 308–313. [23]

Апробация работы

Результаты работы докладывались автором на следующих международных конференциях:

- The Seventh Spring Researchers Colloquium on Databases and Information Systems, SYRCoDIS'2011 (June 2–3, 2011, Moscow, Russia).

- Параллельные вычислительные технологии (ПаВТ’2011). Международная научная конференция (Москва, 28 марта – 1 апреля 2011 г.).

- ”Научный сервис в сети Интернет: суперкомпьютерные центры и задачи”: Международная научная конференция (Новороссийск, 20–25 сентября 2010 г.).

Основные результаты работы

1. Разработана архитектура параллельной системы управления базами данных на основе свободной СУБД PostgreSQL и принципа фрагментного параллелизма.

2. Разработаны интерфейсы и выполнена реализация следующих подсистем PargreSQL:

    - Подсистема тиражирования запросов
    - Оператор обмена
    - Подсистема параллелизации запросов

3. Проведены вычислительные эксперименты, подтверждающие эффективность разработанной системы.

Литература

1. *Evdoridis T., Tzouramanis T.* A Generalized Comparison of Open Source and Commercial Database Management Systems // Database Technologies: Concepts, Methodologies, Tools, and Applications / Ed. by J. Erickson. IGI Global, 2009. P. 13–27.

2. *Brown P.G.* Overview of SciDB: large scale array storage, processing and analysis // SIGMOD Conference / Ed. by A.K. Elmagarmid, D. Agrawal. ACM, 2010. P. 963–968.

3. *Stonebraker M., Becla J., DeWitt D.J. et al.* Requirements for Science Data Bases and SciDB // CIDR. www.crdrdb.org, 2009.

4. *Stonebraker M., Kemnitz G.* The Postgres Next Generation Database Management System // *Commun. ACM*. 1991. Vol. 34, No. 10. P. 78–92.

5. *DeWitt D.J., Gray J.* Parallel Database Systems: The Future of High Performance Database Systems // *Commun. ACM*. 1992. Vol. 35, No. 6. P. 85–98.

6. *Sokolinsky L.B.* Organization of Parallel Query Processing in Multiprocessor Database Machines with Hierarchical Architecture // *Programming and Computer Software*. 2001. Vol. 27, No. 6. P. 297–308.

7. *Lepikhov A.V., Sokolinsky L.B.* Query processing in a DBMS for cluster systems // *Programming and Computer Software*. 2010. Vol. 36, No. 4. P. 205–215.

8. *Samokhvalov N.* XML Support in PostgreSQL // SYRCoDIS / Ed. by S.D. Kuznetsov, A. Fomichev, B. Novikov, D. Shaporenkov. Vol. 256 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

9. *Havinga Y., Dijkstra W., de Keijzer A.* Adding HL7 version 3 data types to PostgreSQL // *CoRR*. 2010. Vol. abs/1003.3370.

10. *Guliato D., de Melo E.V., Rangayyan R.M., Soares R.C.* POSTGRESQL-IE: An Image-handling Extension for PostgreSQL // *J. Digital Imaging*. 2009. Vol. 22, No. 2. P. 149–165.

11. *Levshin D.V., Markov A.S.* Algorithms for integrating PostgreSQL with the semantic web // *Programming and Computer Software*. 2009. Vol. 35, No. 3. P. 136–144.

12. *Lee R., Zhou M.* Extending PostgreSQL to Support Distributed/Heterogeneous Query Processing // DASFAA / Ed. by K. Ramamohanarao, P.R. Krishna, M.K. Mohania, E. Nantajeewarawat. Vol. 4443 of *Lecture Notes in Computer Science*. Springer, 2007. P. 1086–1097.

13. *Paes M., Lima A.A.B., Valduriez P., Mattoso M.* High-Performance Query Processing of a Real-World OLAP Database with ParGRES // VECPAR / Ed. by J.M.L.M. Palma, P. Amestoy, M.J. Daydé et al. Vol. 5336 of *Lecture Notes in Computer Science*. Springer, 2008. P. 188–200.

14. Pgpool Programmer's Manual. URL: `http://pgpool.projects.postgresql.org/pgpool-II/doc/pgpool-en.html` (дата обращения: 20.06.2011).

15. *Kotowski N., Lima A.A.B., Pacitti E. et al.* Parallel query processing for OLAP in grids // *Concurrency and Computation: Practice and Experience*. 2008. Vol. 20, No. 17. P. 2039–2048.

16. *Соколинский Л.Б.* Организация параллельного выполнения запросов в многопроцессорной машине баз данных с иерархической архитектурой // *Программирование*. 2001. № 6. С. 13–29.

17. *Sokolinsky L., Axenov O., Gutova S. et al.* Omega: The highly parallel database system project // ADBIS. 1997. P. 88–90.

18. MPI: A Message-Passing Interface Standard Version 2.2. URL: `http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf` (дата обращения: 21.02.2011).

19. *Колтаков А.В.* Разработка системы обменов сообщениями в параллельной СУБД для кластерных вычислительных систем: Вып. квалиф. работа … бакалавра информационных технологий: 010400.62 / Южно-Уральский государственный университет. Челябинск, 2010. 35 л. URL: `http://omega.sp.susu.ac.ru/publications/bachelorthesis/10-Koltakov.pdf` (дата обращения: 20.02.2011).

20. Вычислительный кластер СКИФ Урал. Сайт суперкомпьютерного центра ЮУрГУ. URL: `http://supercomputer.susu.ru/computers/skif_ural/` (дата обращения: 20.06.2011).

21. *Pan C.* Development of a Parallel DBMS on the Basis of PostgreSQL // Proceedings of the Seventh Spring Researchers' Colloquium on Databases and Information Systems (SYRCODIS'2011). Moscow: Moscow State University, 2011. P. 57–61.

22. *Пан К.С., Цымблер М.Л.* Архитектура и принципы реализации параллельной СУБД PargreSQL // Параллельные вычислительные технологии (ПаВТ'2011): Труды международной научной конференции (Москва, 28 марта – 1 апреля 2011 г.). Челябинск: Издательский центр ЮУрГУ, 2011. P. 577–584.

23. *Пан К.С., Цымблер М.Л.* Проект PargreSQL: Разработка параллельной СУБД на основе свободной СУБД PostgreSQL // Научный сервис в сети Интернет: суперкомпьютерные центры и задачи: Труды международной научной конференции (Новороссийск, 20–25 сентября 2010 г.). М: Изд-во МГУ, 2010. P. 308–313.