ЮЖНО-УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

На правах рукописи

ПАН Константин Сергеевич

МЕТОДЫ ВНЕДРЕНИЯ ФРАГМЕНТНОГО ПАРАЛЛЕЛИЗМА В ПОСЛЕДОВАТЕЛЬНУЮ СУБД С ОТКРЫТЫМ ИСХОДНЫМ КОДОМ

Специальность 05.13.11 — математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Диссертация на соискание ученой степени
кандидата физико-математических наук

Научный руководитель:
ЦЫМБЛЕР Михаил Леонидович,
кандидат физ.-мат. наук, доцент

Челябинск — 2013

Оглавление

Введение **4**

1. Фрагментный параллелизм и СУБД с открытым кодом **15**

1.1. Фрагментный параллелизм . . . 15

1.2. Обзор последовательных свободных СУБД с открытым исходным кодом . . . 18

1.3. Архитектура СУБД PostgreSQL . . . 23

1.3.1. Взаимодействие процессов СУБД . . . 23

1.3.2. Этапы обработки запроса . . . 24

1.3.3. Модульная структура . . . 25

1.3.4. Размещение компонентов . . . 26

1.4. Выводы по главе 1 . . . 26

2. Внедрение параллелизма в последовательную СУБД **29**

2.1. Методы внедрения фрагментного параллелизма . . . 29

2.1.1. Тиражирование запросов . . . 30

2.1.2. Организация обменов . . . 30

2.1.3. Построение параллельного плана запроса . . . 32

2.1.4. Обработка запросов на изменение данных . . . 33

2.1.5. Хранение метаданных о фрагментации . . . 35

2.1.6. Портирование приложений . . . 35

2.1.7. Модификация исходных текстов . . . 35

2.2. Архитектурные подходы и алгоритмы . . . 36

2.2.1. Подсистема тиражирования . . . 37

2.2.2. Подсистема портирования . . . 38

2.2.3. Оператор обмена (*exchange*) . . . 39

2.2.4. Параллелизатор плана запроса . . . 45

2.2.5. Запросы на изменение данных . . . 48

2.2.6. Запросы на определение данных . . . 49
2.3. Архитектура параллельной СУБД PargreSQL . . . 50
2.3.1. Взаимодействие процессов СУБД . . . 51
2.3.2. Этапы обработки запроса . . . 52
2.3.3. Модульная структура . . . 53
2.3.4. Размещение компонентов . . . 54
2.4. Выводы по главе 2 . . . 55

3. Применение параллельной СУБД PargreSQL для интеллектуального анализа графов 57

3.1. Определение задачи разбиения графа . . . 57
3.2. Обзор существующих решений задачи разбиения графа . . . 58
3.3. Разбиение графа с помощью СУБД PargreSQL . . . 61
3.3.1. Реляционная схема данных . . . 62
3.3.2. Огрубление графа . . . 64
3.3.3. Уточнение разбиения графа . . . 66
3.4. Выводы по главе 3 . . . 69

4. Вычислительные эксперименты 71

4.1. План и аппаратная платформа экспериментов . . . 71
4.2. Ускорение и расширяемость . . . 72
4.3. Производительность на тестах TPC . . . 75
4.4. Исследование разбиения сверхбольших графов . . . 76
4.5. Выводы по главе 4 . . . 79

Заключение 81

Литература 89

Приложение. Статистические данные о популярности современных свободных СУБД 100

Введение

Актуальность темы

В настоящее время одним из феноменов, оказывающих существенное влияние на область технологий обработки данных, являются *сверхбольшие данные*. В условиях современного информационного общества имеется широкий спектр приложений (социальные сети [24, 64], электронные библиотеки [1, 90], геоинформационные системы [59, 78] и др.), в каждом из которых производятся неструктурированные данные, имеющие сверхбольшие объемы и высокую скорость прироста (от 1 Терабайта в день). Исследования экспертов корпорации EMC показывают, что к 2020 г. мировой объем данных достигнет 40 Зеттабайт[1].

Сверхбольшие данные путем интеллектуального анализа преобразуются в сверхбольшие реляционные базы данных, которые сохраняют в структурированном виде полученные результаты анализа, требующие параллельной обработки.

Следствием интеллектуального анализа сверхбольших данных на основе соответствующих методов, алгоритмов и программного обеспечения является появление *сверхбольших реляционных баз данных*, в которых в структурированном виде сохраняются полученные результаты аналитической обработки.

В настоящее время *параллельные системы баз данных* [32], которые обеспечивают обработку запросов на многопроцессорных и многоядерных вычислительных системах, признаются научным сообществом как единственное эффективное средство для организации хранения и обработки сверхбольших баз данных. Базисной концепцией параллельных систем баз данных является *фрагментный параллелизм*, предполагающий разбиение

[1]Press Release EMC2. URL: `http://www.emc.com/about/news/press/2012/20121211-01.htm` (дата обращения: 10.03.2013).

отношений базы данных на горизонтальные фрагменты, которые могут обрабатываться независимо на разных узлах многопроцессорной системы.

Однако существующие сегодня коммерческие СУБД, использующие фрагментный параллелизм (Teradata [69], Greenplum [88], IBM DB2 Parallel Edition [22] и др.), имеют высокую стоимость и, во многих случаях, ориентированы на специфические аппаратно-программные платформы. Например, аппаратно-программные решения корпорации Teradata предлагаются по ценам от $16 500 за 1 терабайт обслуживаемой базы данных[2]; СУБД Greenplum продается по бессрочной лицензии, которая стоит $16 000 за используемое вычислительной системой процессорное ядро или $70 000 за 1 терабайт обслуживаемой базы данных, а годовая поддержка продукта стоит 22% от суммы покупки[3]. СУБД Oracle Real Application Cluster предлагается по цене $10 000 за процессор при бессрочной лицензии[4].

Идеологически близкими к параллельным СУБД, основанным на концепции фрагментного параллелизма, являются кластерные СУБД с ПО промежуточного уровня [19]. В зависимости от функций, которые выполняет промежуточное ПО, можно различать кластерные СУБД, ориентированные на приложения класса OLTP, и кластерные СУБД, ориентированные на приложения класса OLAP.

В случае *сценария OLTP (Online Transaction Processing, оперативная обработка транзакций)* СУБД обрабатывает большое количество коротких транзакций и промежуточное ПО обеспечивает межтранзакционный параллелизм. Подключающиеся к системе клиенты распределяются для обработки нескольким экземплярам СУБД, что позволяет повысить доступность системы при большом количестве клиентов.

Сценарий OLAP (Online Analytical Proicessing, оперативная аналитическая обработка) подразумевает, что СУБД выполняет сложные запро-

[2]Teradata Workload-Specific Platform Pricing. URL: `http://www.teradata.com/t/WorkArea/DownloadAsset.aspx?id=4682` (дата обращения: 01.10.2013).

[3]Greenplum update – Release 3.3. URL: `http://www.dbms2.com/2009/06/05/greenplum-update-release-3-3/` (дата обращения: 01.10.2013).

[4]Oracle Online Store. URL: `http://shop.oracle.com` (дата обращения: 01.10.2013)

сы над большим количеством данных. В этом случае промежуточное ПО обеспечивает внутризапросный параллелизм, осуществляя прием запросов пользователя, их преобразование и распределение на узлы кластера, слияние частичных результатов и передачу их пользователю.

Промежуточный слой обеспечивает распределение клиентов по узлам в кластерных СУБД, предназначенных в первую очередь для сценария OLTP, когда система должна обрабатывать большое количество мелких запросов. Архитектура взаимодействия клиента и сервера в таких системах представлена на рис. 1. В этом случае требуется наличие межтранзакционного параллелизма, который подобный подход хорошо обеспечивает. Подключающиеся к системе клиенты распределяются для обработки нескольким экземплярам СУБД, что позволяет повысить доступность системы при большом количестве клиентов.

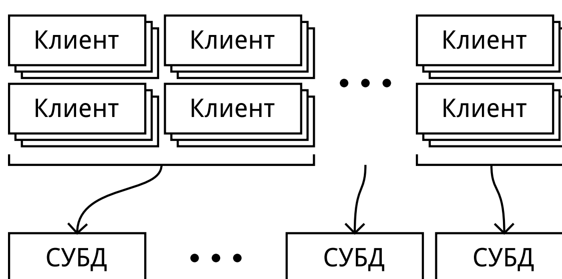


Рис. 1. Кластер баз данных для OLTP

Одним из представителей такого подхода к созданию кластерных СУБД является система *MySQL Cluster* [79]. Данная СУБД представляет собой модифицированную версию MySQL. СУБД MySQL использует слой абстракции для обеспечения низкоуровневого хранилища данных и таким образом поддерживает множество различных движков хранилищ, реализованных в виде подключаемых модулей. MySQL Cluster — это версия MySQL, которая использует в качестве хранилища систему NDB. NDB позволяет хранить данные в памяти множества узлов с учетом фрагментации и репликации. Архитектура системы MySQL Cluster показана на рис. 2.

Следует отметить, что MySQL Cluster ориентирована на параллельную

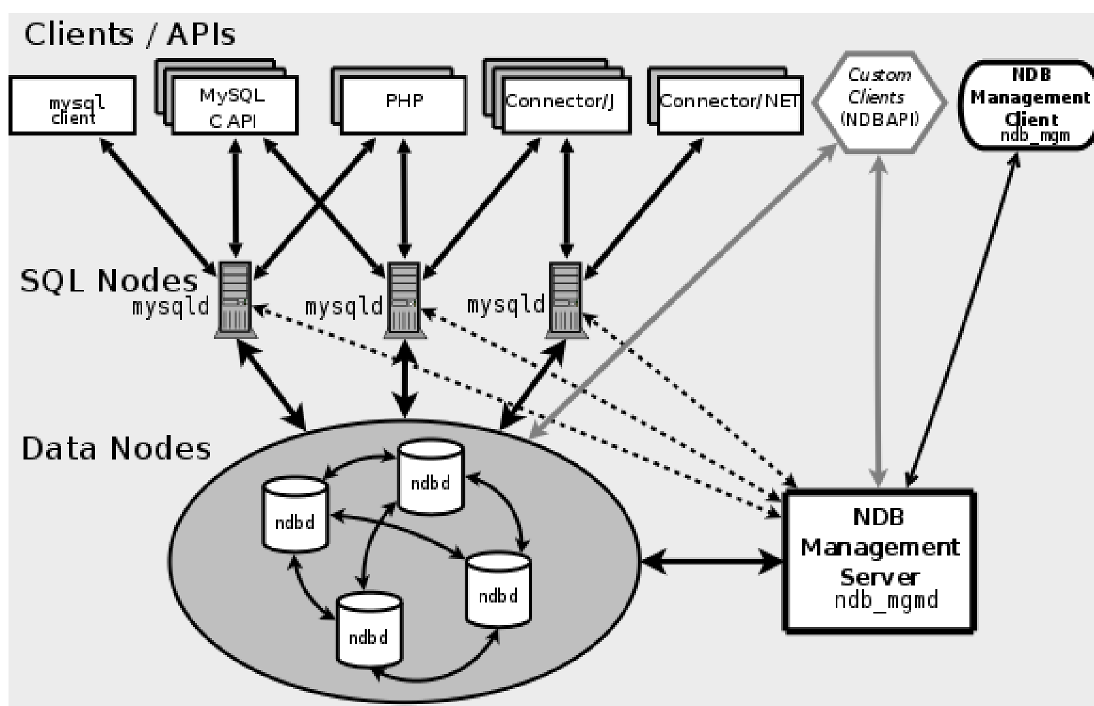


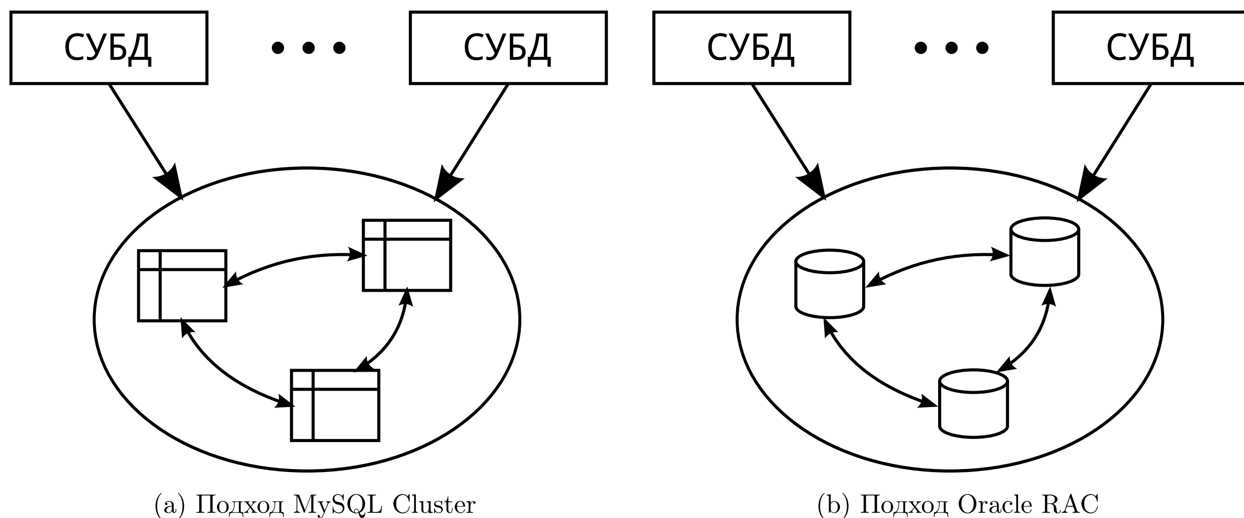
Рис. 2. Архитектура СУБД MySQL Cluster

обработку большого количества OLTP-запросов и не предоставляет внутризапросного параллелизма, а также обладает ограничениями в плане масштабируемости (не более 48 узлов хранения) и размера базы данных (не более 3 терабайт).

Решение *Oracle Real Application Clusters (Oracle RAC)* [77] также представляет собой пример кластерной СУБД. В качестве промежуточного программного обеспечения в данном продукте используется система Oracle Clusterware, предоставляющая следующие основные функции: управление членством узлов в кластере, мониторинг состояния служб кластера, восстановление после сбоя одного или нескольких узлов и др. СУБД Oracle RAC построена на основе архитектуры с разделяемыми дисками. В СУБД Oracle RAC поддерживается хранение до трех реплик базы данных, вследствие чего обеспечивается высокая готовность данных и балансировка нагрузки между узлами кластера. Однако Oracle Clusterware не обеспечивает внутризапросный параллелизм, в силу чего основным назначением Oracle RAC являются приложения класса OLTP. Кроме того, Oracle RAC может

включать в себя не более 100 узлов.

Системы MySQL Cluster и Oracle RAC основаны на использовании общего хранилища. Хранилище может быть реализовано в виде множества вычислительных узлов, обеспечивающих распределенное хранение данных в оперативной памяти (как в MySQL Cluster, см. рис. 3a), или как некоторый общий диск (как в Oracle RAC, см. рис. 3b).



(a) Подход MySQL Cluster

(b) Подход Oracle RAC

Рис. 3. Кластер баз данных общим хранилищем

Если кластерная СУБД ориентирована на OLAP, то есть на обработку сложных аналитических запросов к данным большого объема, то промежуточный слой ПО должен обеспечивать внутризапросный параллелизм, осуществляя прием запросов пользователя, их преобразование и распределение на узлы кластера, слияние частичных результатов и передачу их пользователю. Архитектура взаимодействия клиента и сервера в таких СУБД представлена на рис. 4.

Концепция кластерной СУБД на основе ПО промежуточного слоя реализована в исследовательских проектах *PowerDB-IR* [39], *C-JDBC* [28] и *ParGRES* [68]. Продолжением проекта ParGRES является разработка *GParGRES* [53], которая представляет собой программное обеспечение промежуточного слоя для объединения ParGRES-кластеров в грид. Результаты вычислительных экспериментов показывают, как правило, хорошую

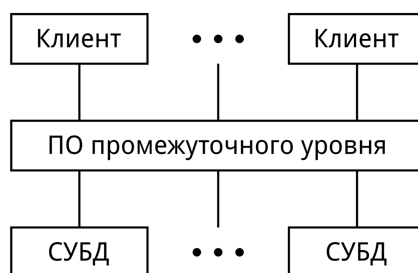


Рис. 4. Кластер баз данных с ПО промежуточного слоя

масштабируемость таких кластеров при выполнении OLAP-запросов. Определенным недостатком данных систем можно считать использование полной репликации всех таблиц базы данных на узлах кластерной системы.

По-видимому, единственной существующей в настоящее время системой с открытым исходным кодом для параллельной обработки сложных запросов к реляционным данным является СУБД *HadoopDB* [17], которая представляет собой архитектурный гибрид вычислительной парадигмы MapReduce и технологий реляционных СУБД.

Парадигма распределенных вычислений *MapReduce* [33] для кластерных систем, разработанная корпорацией Google, может быть описана кратко следующим образом. Один из узлов кластерной системы трактуется как мастер, остальные — рабочие. Обработка данных выполняется в виде последовательности из двух шагов: Map и Reduce. На шаге Map узел-мастер получает входные данные и распределяет их по узлам-рабочим. На шаге Reduce узел-мастер выполняет свертку данных, предварительно обработанных рабочими, и отправку конечного результата пользователю. В качестве подсистемы, реализующей MapReduce-вычисления, в СУБД HadoopDB в используется система *Hadoop* [89]. Hadoop обеспечивает коммуникационную инфраструктуру, объединяющую узлы кластера, на которых выполняются экземпляры СУБД PostgreSQL. Запросы пользователя на языке SQL транслируются в задания для среды MapReduce, которые далее передаются в экземпляры СУБД.

Эксперименты показывают [17, 74], что, хотя система HadoopDB способ-

на демонстрировать устойчивость к отказам и падению производительности узлов, свойственную системам MapReduce, в большинстве случаев параллельные СУБД существенно превосходят ее в производительности. Одной из причин отставания производительности HadoopDB от параллельных СУБД являются предусмотренные архитектурой этой системы накладные расходы на взаимодействие между средой MapReduce и PostgreSQL [3].

Еще одной системой баз данных, которую можно отнести к данному классу, является *vParNDB* [63]. Она представляет собой ПО промежуточного слоя, которое переписывает запросы таким образом, чтобы они выполнялись параллельно с использованием нескольких узлов MySQL Cluster, подключенных к общему хранилищу NDB. Хотя эксперименты на тестах TPC-H показывают хорошее ускорение с использованием такого подхода, любые решения на основе MySQL Cluster наследуют все ограничения данной СУБД.

В настоящее время надежной альтернативой коммерческим СУБД являются свободные *СУБД с открытым кодом* [35, 73], однако проведенный обзор дает основания полагать, что на сегодя отсутствуют свободные СУБД, реализующие фрагментный параллелизм. Данный факт объясняется тем, что параллельная СУБД относится к классу сложного системного программного обеспечения, разработка которого требует существенных финансовых и временны́х затрат.

В связи с этим перспективной является идея модернизации существующего исходного кода свободной последовательной СУБД для построения на ее основе параллельной СУБД путем *внедрения фрагментного параллелизма*. При этом модернизация исходного кода подразумевает отсутствие масштабных изменений в реализации существующих подсистем, которые равнозначны разработке параллельной СУБД «с нуля».

В качестве аппаратной платформы параллельных СУБД, получаемых посредством внедрения фрагментного параллелизма, нами выбраны кластерные системы. *Кластер* представляет собой связанный сетью набор пол-

ноценных компьютеров, используемый в качестве единого вычислительного ресурса. Кластеры сочетают в себе экономичность и высокую вычислительную мощность и в настоящее время претендуют на доминирующее положение в области высокопроизводительных вычислений[5,6].

Коммерческие параллельные СУБД, ориентированные на специализированные аппаратные платформы, покажут ожидаемо бо́льшую эффективность в сравнении с параллельной СУБД для кластеров, полученной посредством модернизации исходного кода последовательной СУБД. Однако параллельная СУБД, полученная посредством внедрения фрагментного параллелизма, потенциально способна показать сравнимую с коммерческими параллельными СУБД масштабируемость, достигаемую путем добавления в кластер новых вычислительных узлов, но оставаясь при этом финансово менее затратным решением.

Таким образом, *актуальной* является задача разработки методов внедрения фрагментного параллелизма в свободные реляционные СУБД с открытым кодом, позволяющие осуществить параллелизацию без масштабных изменений исходного кода.

Цель и задачи исследования

Цель данной работы состояла в разработке методов, архитектурных подходов и алгоритмов, обеспечивающих внедрение фрагментного параллелизма в имеющиеся последовательные СУБД, свободно распространяемые на уровне исходных кодов, а также в проверке разработанных методов путем их применения для решения задач аналитической и оперативной обработки сверхбольших баз данных.

Для достижения этой цели необходимо было решить следующие основные *задачи*:

[5] TOP500 supercomputer sites. URL: `http://www.top500.org` (дата обращения: 01.10.2013)

[6] TOP50 суперкомпьютеров России. URL: `http://www.supercomputers.ru` (дата обращения: 01.10.2013)

1. Разработать методы внедрения фрагментного параллелизма в последовательную СУБД с открытым исходным кодом.
2. На основе разработанных методов предложить архитектурные подходы и алгоритмы, реализующие фрагментный параллелизм в рамках последовательной СУБД с открытым исходным кодом. Выполнить параллелизацию одной из свободных последовательных СУБД.
3. Исследовать эффективность предложенных подходов применительно к решению задач классов OLAP и OLTP, связанных с обработкой сверхбольших баз данных.

Методы исследования

Проведенные в работе исследования базируются на реляционной модели данных. При разработке программной системы применялись методы объектно-ориентированного проектирования и язык UML. В разработке алгоритмов использован аппарат теории графов.

Научная новизна

Научная новизна работы заключается в следующем:

1. Впервые разработаны эффективные методы внедрения фрагментного параллелизма в последовательную СУБД с открытым исходным кодом.
2. Впервые выполнено внедрение фрагментного параллелизма в последовательную СУБД PostgreSQL.
3. Разработан новый алгоритм разбиения сверхбольших графов, состоящих из миллионов вершин и ребер, ориентированный на реляционные СУБД с фрагментным параллелизмом.

Теоретическая и практическая ценность

Теоретическая ценность работы состоит в том, что в ней предложены методы, архитектурные решения и алгоритмы, позволяющие интегрировать фрагментный параллелизм в последовательные реляционные СУБД, свободно распространяемые на уровне исходных кодов.

Практическая ценность работы заключается в том, что путем применения предложенных методов к свободной СУБД PostgreSQL получена параллельная СУБД для кластерных систем, названная PargreSQL, которая применима для решения реальных задач, связанных с обработкой сверхбольших баз данных.

Структура и объем работы

Диссертация состоит из введения, четырех глав, заключения, библиографии и приложения. Объем диссертации составляет 101 страницу, объем библиографии — 98 наименований.

Содержание работы

Первая глава, «Фрагментный параллелизм и СУБД с открытым кодом», содержит описание концепции фрагментного параллелизма и аналитический обзор современных последовательных СУБД, свободно распространяемых на уровне исходных кодов, на основе которого осуществлен выбор СУБД PostgreSQL для внедрения фрагментного параллелизма. Описана архитектура СУБД PostgreSQL.

Во второй главе, «Внедрение параллелизма в последовательную СУБД», предлагается комплекс методов для внедрения фрагментного параллелизма в последовательную СУБД с открытыми исходными ко-

дами и описывается параллельная СУБД PargreSQL, реализованная путем внедрения фрагментного параллелизма в свободную СУБД PostgreSQL.

В третьей главе, «Применение параллельной СУБД PargreSQL для интеллектуального анализа графов», представлен новый подход к решению задачи разбиения сверхбольших графов, которые состоят из миллионов вершин и ребер, основанный на использовании параллельной СУБД PargreSQL. Данная задача решается с целью проверки применимости СУБД PargreSQL к задачам аналитической обработки сверхбольших баз данных. Описаны алгоритмы, которые реализуют стадии разбиения графа в виде запросов SQL.

В четвертой главе, «Вычислительные эксперименты», приводятся результаты вычислительных экспериментов, выполненных на суперкомпьютере «Торнадо ЮУрГУ» с использованием разработанной параллельной СУБД PargreSQL. В рамках экспериментов исследованы ускорение и расширяемость СУБД PargreSQL, ее производительность на стандартных тестах консорциума TPC (Transaction Processing Council), а также эффективность и качество разбиения реальных сверхбольших графов.

В заключении суммируются основные результаты диссертационной работы, выносимые на защиту, приводятся данные о публикациях и апробациях автора по теме диссертации и рассматриваются направления дальнейших исследований в данной области.

В приложение вынесены статистические данные о популярности современных свободных СУБД с открытым исходным кодом, которые были использованы при принятии решения о выборе СУБД для внедрения фрагментного параллелизма.

Глава 1. Фрагментный параллелизм и СУБД с открытым кодом

Данная глава содержит описание концепции фрагментного параллелизма и аналитический обзор современных последовательных СУБД, свободно распространяемых на уровне исходных кодов, на основе которого осуществлен выбор СУБД для внедрения фрагментного параллелизма. Глава организована следующим образом.

В разделе 1.1 представлено описание концепции фрагментного параллелизма. В разделе 1.2 сделан краткий обзор современных свободных СУБД, на основе которого осуществлен выбор СУБД PostgreSQL для внедрения фрагментного параллелизма. Раздел 1.3 содержит краткое описание архитектуры СУБД PostgreSQL. В разделе 1.4 суммируются основные результаты, полученные в данной главе.

1.1. Фрагментный параллелизм

Базисной концепцией параллельной обработки запросов в реляционных системах баз данных является фрагментный параллелизм [94]. Принципиальная схема обработки запроса с использованием фрагментного параллелизма выглядит следующим образом (см. рис. 5).

Реляционные отношения, хранящиеся в базе данных, подвергаются горизонтальной фрагментации по дискам многопроцессорной системы. Способ фрагментации определяется функцией фрагментации, которая для каждого кортежа отношения вычисляет номер процессорного узла, на котором должен быть размещен этот кортеж. Запрос параллельно выполняется на всех процессорных узлах в виде набора параллельных агентов [2], каждый из которых обрабатывает отдельный фрагмент отношения на выделенном

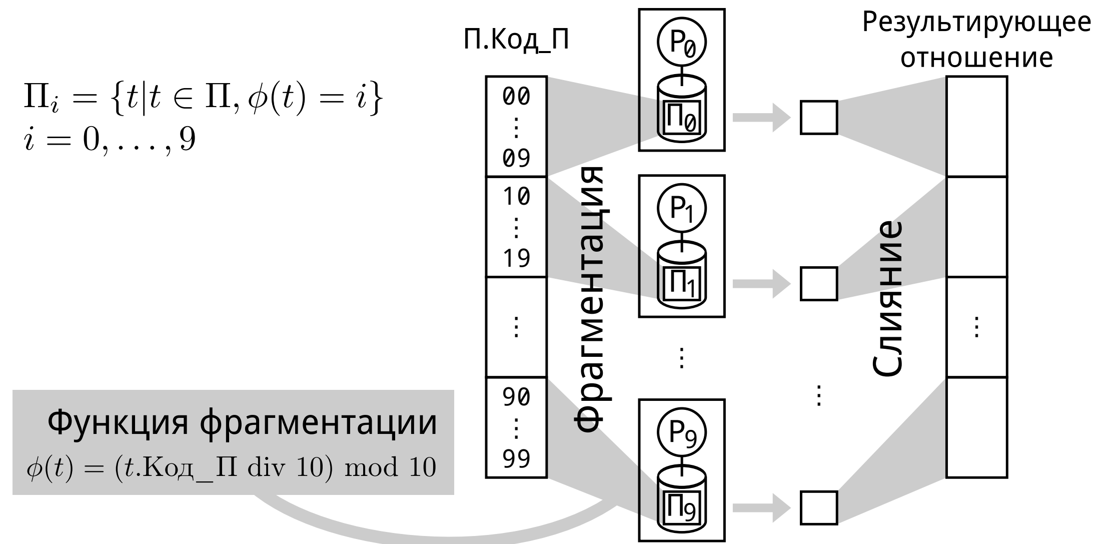


Рис. 5. Параллельная обработка запроса на основе фрагментного параллелизма

ему процессорном узле. Полученные агентами результаты сливаются в результирующее отношение.

Функция фрагментации отношения R

$$\varphi : R \to \{0, 1, ..., k-1\}$$

вычисляет номер процессорного узла, на котором должен храниться кортеж. Величина k (количество процессорных узлов) называется *степенью фрагментации*. Для фрагментации целесообразно использовать *атрибутную фрагментацию*, которая предполагает, что

$$\forall r \in R(\varphi_R(r) = f_A(r.A)),$$

где $f_A : D_A \to \{0, ..., k-1\}$ является некоторой функцией, определенной на домене атрибута A. То есть атрибутная фрагментация предполагает, что функция фрагментации зависит от определенного атрибута фрагментируемого отношения. Преимущество атрибутной фрагментации в том, что она допустима основными реляционными операциями: *группировка*, *удаление дубликатов*, *естественное соединение*.

Несмотря на то, что каждый параллельный агент в процессе выполнения запроса независимо обрабатывает свой фрагмент отношения, для полу-

чения корректного результата необходимо выполнять пересылки кортежей между процессорными узлами. Для организации таких пересылок в соответствующие места дерева плана запроса вставляется оператор *exchange* [12]. Оператор *exchange* однозначно задается номером порта обмена и функцией пересылки. Функция пересылки для каждого входного кортежа вычисляет логический номер процессорного узла, на котором данный кортеж должен быть обработан. Порт обмена позволяет включать в дерево запроса произвольное количество операторов *exchange* (для каждого оператора указывается свой уникальный порт обмена).

Общая схема организации обработки запросов в параллельной СУБД выглядит следующим образом [2]. Мы будем полагать, что вычислительная система представляет собой кластер из N вычислительных узлов (см. рис. 6), и каждое отношение базы данных, задействованное в обработке запроса, фрагментировано по всем этим узлам. В соответствии с данной схемой обработка запроса состоит из трех этапов.

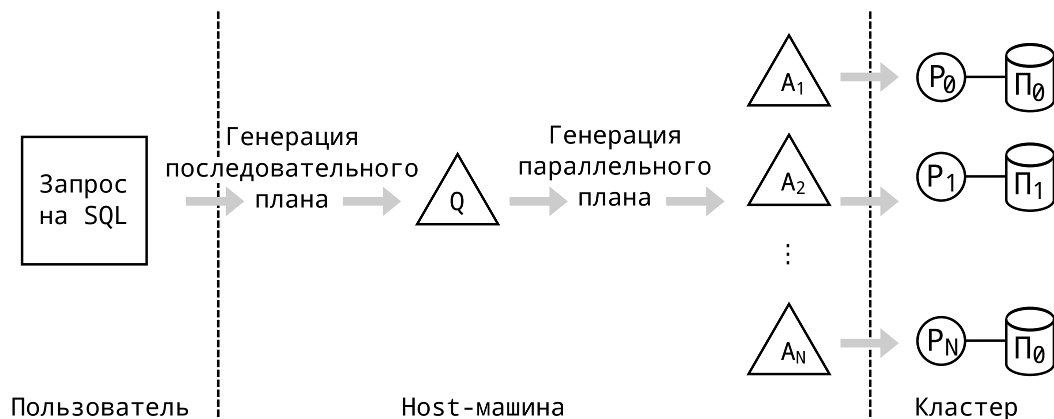


Рис. 6. Схема обработки запроса в параллельной СУБД. Q — последовательный физический план, A_i — параллельный агент, P_i — вычислительный узел, D_i — диск

На первом этапе SQL-запрос передается пользователем на выделенную host-машину, где транслируется в некоторый последовательный физический план [30].

На втором этапе последовательный физический план преобразуется в

параллельный план, представляющий собой совокупность параллельных агентов. Это достигается путем вставки оператора обмена *exchange* в соответствующие места плана запроса.

На третьем этапе параллельные агенты пересылаются с host-машины на соответствующие вычислительные узлы, где интерпретируются исполнителем запросов. Результаты выполнения агентов объединяются корневым оператором *exchange* на нулевом узле, откуда передаются на host-машину. Роль host-машины может играть любой узел вычислительного кластера.

1.2. Обзор последовательных свободных СУБД с открытым исходным кодом

В данном разделе представлен обзор основных существующих в настоящее время свободных СУБД с открытым исходным кодом. Целью обзора является выбор последовательной СУБД для внедрения в нее фрагментного параллелизма.

Для выбора свободной СУБД, в которую целесообразно внедрить фрагментный параллелизм, нами предлагается следующий набор из шести критериев: поддержка реляционной модели данных, автономность, либеральность лицензии, документирование, популярность и сопровождение. Первые три критерия являются *необходимыми*, остальные — *достаточными*. Далее кратко описана семантика критериев выбора, расположенных в порядке убывания их важности.

1. **Поддержка реляционной модели** необходима, поскольку в настоящее время эта модель является де-факто стандартом в области технологий баз данных и наиболее приспособлена для реализации концепции фрагментного параллелизма.

2. **Автономность** означает, что система не является встраиваемой и обес-

печивает существенно необходимую возможность запуска пользовательских приложений отдельно от СУБД.

3. **Либеральность лицензии** соответствует требованию отсутствия ограничений на использование СУБД и распространения ее модификаций.

4. **Документирование** означает, что исходные тексты системы должны быть качественно специфицированы и иметь качественную документацию для облегчения модификации исходных текстов.

5. **Популярность** предполагает как можно более широкий круг приложений и групп пользователей данного продукта.

6. **Сопровождение** означает, что в настоящее время не прерваны техническая поддержка и доработка системы в соответствии с изменяющимися нуждами пользователей.

В обзор включены следующие системы: SQLite, PostgreSQL, MySQL, MariaDB, MaxDB, Ingres, HSQLDB, BerkeleyDB и Firebird, – поскольку в настоящее время они являются наиболее популярными свободными СУБД с открытым исходным кодом [35, 73].

SQLite [42, 96] представляет собой встраиваемую реляционную СУБД, которая реализована в виде библиотеки. Обладает самой большой популярностью (используется в широком спектре продуктов, начиная со смартфонов и заканчивая самолетами), однако, в отличие от серверных решений, СУБД SQLite не представляет собой отдельный процесс, она интегрируется в приложение и работает напрямую с файлом базы данных, предоставляя приложению интерфейс на языке SQL. Это исключает использование параллельных вычислений и кластерных систем для обработки запроса. Исходный код SQLite находится в общественном достоянии и имеет документацию для программистов и краткие спецификации в комментариях.

Система **PostgreSQL** [85] является реляционной СУБД, разрабатываемой группой энтузиастов и распространяемой на условиях свободной

лицензии BSD (Berkley Software Distribution). В силу характера проекта обладает подробнейшими внутренними спецификациями и документацией для программистов, а также активным сообществом разработчиков. СУБД PostgreSQL поддерживает стандарт SQL:2008, реализует ACID-транзакции и поддерживает большое количество процедурных языков. Для PostgreSQL создается большое количество расширений и дополнений сторонними разработчиками: поддержка данных в формате XML [80] и Semantic Web [55], обработка изображений [41] и медицинских данных [44], обработка распределенных запросов [54] и др.

СУБД **MySQL** [66, 16] — одна из самых популярных на сегодня реляционных СУБД с открытым исходным кодом. Большое распространение система получила как элемент связки LAMP (Linux-Apache-MySQL-Perl/PHP/Python) — наиболее часто используемой комбинации ПО для развертывания веб-серверов. Разрабатывается корпорацией Oracle и распространяется по двум лицензиям: свободной и коммерческой. СУБД MySQL поддерживает стандарт SQL:1999 и ACID-транзакции. Интересной особенностью является использование подключаемых модулей в качестве низкоуровневого хранилища данных. Это позволяет подстраивать СУБД под свои нужды, используя подходящий случаю втроенный модуль или от сторонних разработчиков. СУБД MySQL имеет достаточно подробную документацию для программиста, что делает возможным внедрение своих изменений в код.

Проект **MariaDB** [95] был создан разработчиками СУБД MySQL после покупки его компании корпорацией Oracle. СУБД MariaDB является ответвлением СУБД MySQL, которое было создано в целях сохранения свободного статуса данной СУБД. СУБД MariaDB заявляется разработчиками как СУБД, полностью совместимая с СУБД MySQL, которую можно использовать в приложениях как прямую замену СУБД MySQL. В отличие от СУБД MySQL, данный продукт разрабатывается сообществом и распространяется только по свободной лицензии. Однако в настоящее вре-

мя система MariaDB обладает существенно меньшей популярностью, чем СУБД MySQL.

СУБД **MaxDB** [23] является реляционной СУБД, реализующей стандарт SQL-92. СУБД MaxDB разрабатывается компанией SAP AG и распространялась по свободной лицензии GPL вплоть до 2007 г., однако в дальнейших версиях исходные коды недоступны.

Реляционная СУБД **Ingres** [45] была создана в 1974 г. и поддерживается в настоящее время, распространяясь по свободной лицензии GPL (General Public Licence). Данная СУБД поддерживает ACID-транзакции и реализует с помощью транзакций все, включая запросы на изменение структуры данных. Единственная доступная документация для разработчиков – краткие комментарии в коде.

Проект **HSQLDB** [62] реализует реляционную СУБД на языке Java и поддерживает стандарты SQL-92 и SQL:2008. HSQLDB может запускаться как отдельный демон или встраиваться в виде библиотеки и распространяется по лицензии BSD, поэтому широко используется в качестве хранилища данных в сторонних проектах (OpenOffice, LibreOffice, Mathematica и др.).

Система **BerkeleyDB** [65] представляет собой библиотеку, реализующую встраиваемую СУБД для хранения пар «ключ-значение». Хранит данные произвольного типа в виде массивов байтов и не является реляционной СУБД. Разрабатывается корпорацией Oracle и распространяется по лицензии AGPL, которая обязывает предоставить пользователю исходный код данной СУБД или ее модифицированной версии, даже если сама система не распространяется, но ею пользуются, например, в составе интернет-сервиса.

Система **Firebird** [25] является свободной реляционной СУБД. Проект ответвился от СУБД Borland InterBase [27] в 2000 г. Реализует стандарт SQL-92 и ACID-транзакции. Поддерживает работу в режиме демона и во встраиваемом режиме. Разработчиками планируется использование СУБД Firebird в качестве замены СУБД HSQLDB в проекте LibreOffice. Из доку-

ментации в наличии комментарии в коде и описание общей архитектуры.

Характеристики рассмотренных СУБД резюмированы в табл. 1. Необходимые критерии оценивались баллами как 1 («выполняется») либо 0

Табл. 1. Сравнительные характеристики свободных СУБД

Система	Поддержка РМД	Автономность	Лицензия	Документирование	Популярность	Сопровождение	Сумма
PostgreSQL	1	1	2	3	2	1	10
MySQL	1	1	1	2	3	1	9
HSQLDB	1	1	2	2	1	1	8
SQLite	1	0	2	1	3	1	8
MariaDB	1	1	1	2	1	1	7
MaxDB	1	1	1	2	1	1	7
BerkeleyDB	0	0	1	1	3	1	6
Firebird	1	1	1	1	1	1	6
Ingres	1	1	1	1	1	1	6

(«не выполняется»). Документирование СУБД оценивалось от 3 до 1 балла в зависимости от степени обеспечения следующих возможностей: наличие спецификаций всех подпрограмм, комментариев с кратким описанием семантики структур данных и алгоритмов, справочника по исходным текстам, руководства для разработчика и использования системы контроля версий исходных текстов. Степень популярности оценивалась от 3 до 1 балла и представляла собой интегральную характеристику следующих нормализованных статистических показателей: количество найденных поисковой системой упоминаний данной СУБД web-страниц в интернет-репозиториях свободного ПО SourceForge.net, FreeCode.com, GitHub.com и

code.google.com, портала объявлений о вакансиях Job.com и онлайн-магазина книг Amazon.com. Соответствующие статистические данные вынесены в приложение (см. с. 100).

Таким образом, наиболее перспективной для целей исследования была выбрана СУБД PostgreSQL. Далее рассмотрена архитектура данной СУБД и описаны методы внедрения фрагментного параллелизма на ее примере.

1.3. Архитектура СУБД PostgreSQL

1.3.1. Взаимодействие процессов СУБД

В основе архитектуры PostgreSQL лежит модель «клиент-сервер». В сеансе работы с PostgreSQL участвуют три вида взаимодействующих процессов (см. рис. 7): *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.

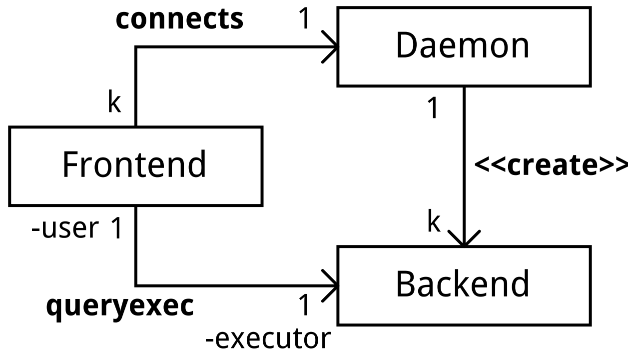


Рис. 7. Процессы СУБД PostgreSQL

Порядок взаимодействия клиента и СУБД представлен на рис. 8. 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 про-

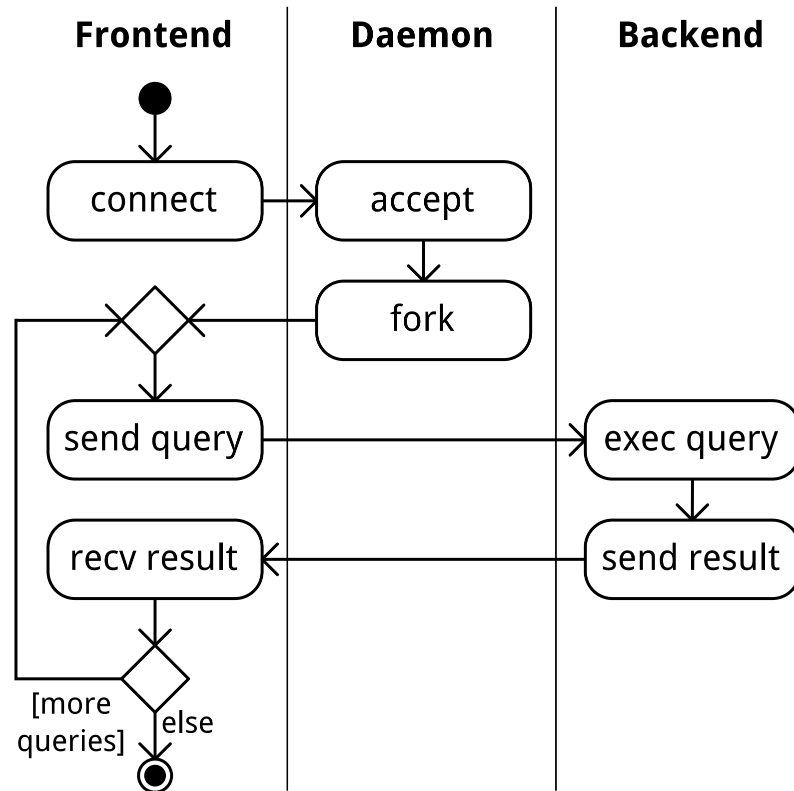


Рис. 8. Взаимодействие клиента и сервера

цессу, который выполняет обработку этого запроса и отправку результатов обратно клиенту.

1.3.2. Этапы обработки запроса

Схема обработки запроса в СУБД PostgreSQL представлена на рис. 9.

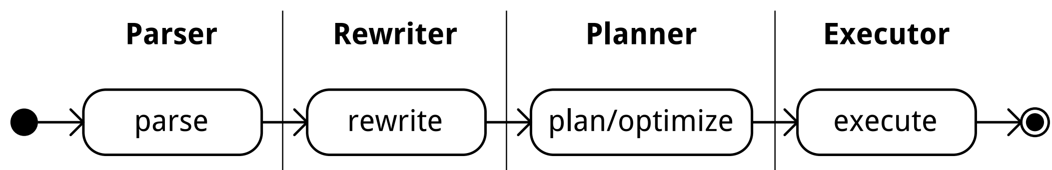


Рис. 9. Обработка запроса в СУБД PostgreSQL

Обработка запроса состоит из следующих шагов.

На шаге *parse (разбор)* осуществляется разбор запроса на языке SQL и формирование дерева разбора.

На шаге *rewrite* выполняется преобразование дерева разбора в соответствии с правилами, которые заданы администратором базы данных, например, в случае реализации пользовательских представлений базы данных.

Шаг *plan/optimize* заключается в формировании последовательного оптимального плана запроса. Результатом данного шага является физический план исполнения запроса, имеющий оптимальную оценку сложности [61].

Шаг *execute* предполагает выполнение плана запроса. Поскольку операции в плане запроса имеют итераторный интерфейс, исполнение сводится к перебору всех кортежей, поступающих из корня плана, и применению к ним соответствующего запросу действия (вставка, удаление, обновление, вывод).

1.3.3. Модульная структура

СУБД PostgreSQL содержит следующие подсистемы, представленные на рис. 10:

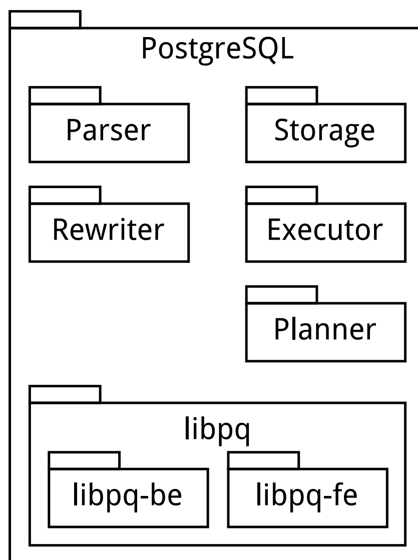


Рис. 10. Подсистемы СУБД PostgreSQL

— *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).

1.3.4. Размещение компонентов

Размещение компонентов СУБД PostgreSQL приведено на рис. 11.

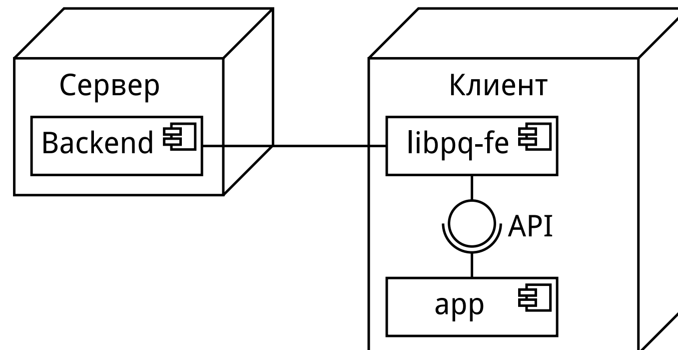


Рис. 11. Размещение компонентов СУБД PostgreSQL

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

1.4. Выводы по главе 1

В данной главе рассмотрены основные подходы к построению параллельных систем баз данных, существующие в настоящее время: фрагментный параллелизм и кластерные СУБД с промежуточным ПО.

Фрагментный параллелизм предполагает горизонтальную фрагментацию каждого отношения базы данных и распределение фрагментов по дискам многопроцессорной вычислительной системы. Способ фрагментации определяется функцией фрагментации, вычисляющей для каждого кортежа отношения номер процессорного узла, на котором должен быть размещен этот кортеж. Запрос параллельно выполняется на всех процессорных узлах в виде набора параллельных агентов, каждый из которых обрабатывает отдельный фрагмент отношения на выделенном ему процессорном узле. Полученные агентами результаты сливаются в результирующее отношение. Имеющиеся в настоящее время параллельные СУБД на базе фрагментного параллелизма представляют собой специализированные аппаратно-программные комплексы и имеют высокую стоимость.

Параллельная система баз данных, построенная на основе промежуточного ПО и кластера, на каждом узле которой запущен экземпляр последовательной СУБД. В зависимости от функций, которые выполняет промежуточное ПО, различают кластерные СУБД для приложений класса OLTP и кластерные СУБД для приложений класса OLAP.

Кластерные СУБД для OLTP приложений, среди которых имеются как коммерческие (СУБД Oracle Real Application Cluster), так и свободные решения (СУБД MySQL Cluster), имеют ограниченную масштабируемость. Имеющиеся в настоящее время свободные кластерные СУБД для OLAP приложений также имеют свои недостатки: например, СУБД ParGRES использует полную репликацию базы данных, а СУБД HadoopDB показывает более низкую производительность, чем параллельные СУБД на основе фрагментного параллелизма в силу обусловленных архитектурой построения этой системы накладных расходов на взаимодействие между средой MapReduce и свободной СУБД PostgreSQL.

В соответствии с этим в настоящее время является актуальной задача разработки методов внедрения фрагментного параллелизма в СУБД с открытым исходным кодом, решение которой позволит получать масшта-

бируемые и относительно недорогие параллельные системы баз данных.

Проведен сравнительный анализ существующих свободных СУБД с целью выбора СУБД, в которую будет осуществляться внедрение фрагментного параллелизма. В результате данного анализа кандидатом на внедрение параллелизма была выбрана свободная СУБД PostgreSQL.

Глава 2. Внедрение параллелизма в последовательную СУБД

В данной главе описаны разработанные методы внедрения фрагментного параллелизма в свободную СУБД с открытым исходным кодом. Предложены архитектурные подходы и алгоритмы, реализующие фрагментный параллелизм в рамках последовательной СУБД с открытым исходным кодом. Описана параллелизация последовательной СУБД PostgreSQL, в ходе которой получена параллельная СУБД, названная PargreSQL Раздел 2.2 содержит описание методов внедрения фрагментного параллелизма в последовательную СУБД с открытым исходным кодом. В разделе 2.3 представлены архитектурные подходы и алгоритмы, реализующие фрагментный параллелизм в последовательной СУБД на примере PostgreSQL. В разделе 2.4 суммируются основные результаты, полученные в данной главе.

2.1. Методы внедрения фрагментного параллелизма

Данный раздел содержит описание методов внедрения фрагментного параллелизма и алгоритмов реализации подсистем, составляющих параллельную СУБД PargreSQL, и организован следующим образом. Раздел 2.1.1 содержит описание метода тиражирования запросов. Раздел 2.1.2 посвящен методу организации обменов кортежами между экземплярами параллельной СУБД. Раздел 2.1.6 посвящен методу портирования приложений последовательной СУБД в параллельную СУБД.

2.1.1. Тиражирование запросов

Тиражирование запроса предполагает отправку данного запроса множеству экземпляров последовательной СУБД, каждый из которых обрабатывает свой собственный фрагмент базы данных. Описанные в разде-

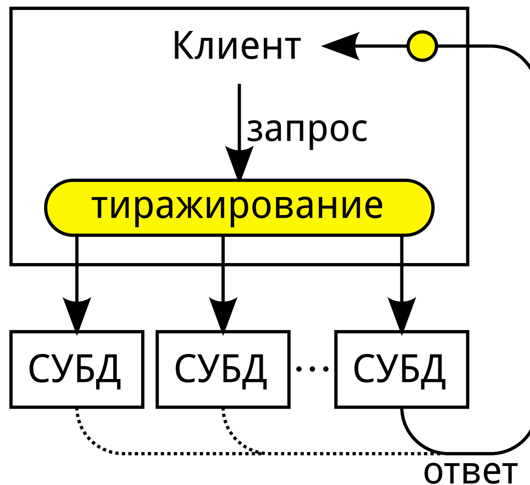


Рис. 12. Тиражирование запросов

лах 2.1.2 и 2.1.3 методы обеспечивают сборку частичных результатов запроса на одном экземпляре, поэтому все экземпляры, кроме одного, дают пустой ответ. Кроме собственно отправки запроса множеству экземпляров СУБД требуется получить и проверить результаты от всех экземпляров, несмотря на то, что они заведомо пусты. В случае возникновения ошибки при обработке запроса на одном из узлов необходимо считать итоговый результат некорректным.

2.1.2. Организация обменов

Для получения корректных результатов в ходе обработки запроса в параллельной СУБД на основе фрагментного параллелизма требуется обеспечить обмены кортежами между вычислительными узлами. Данная задача решается с помощью *оператора обмена* (*exchange*) [12], который вставляется в план запроса на этапе построения параллельного плана запроса (см.

раздел 2.1.3 и инкапсулирует в себе передачу кортежей между экземплярами СУБД на различных узлах вычислительной системы.

Оператор обмена реализуется аналогично другим операторам физической алгебры в модифицируемой СУБД, которые обычно имеют итераторный интерфейс. Оператор обмена имеет два свойства: порт и функцию пересылки, — и является составным (см. рис. 13).

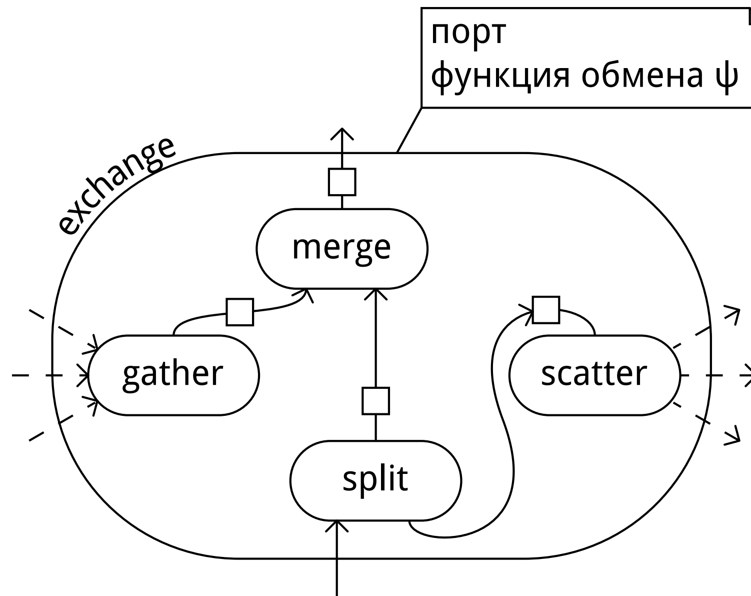


Рис. 13. Архитектура оператора обмена

Свойство *порт* необходимо, чтобы отличать операторы обмена в одном плане запроса друг от друга — кортежи из одной точки плана запроса должны попадать в ту же точку плана на другом вычислительном узле.

Функция пересылки $\psi(t)$ необходима, чтобы вычислять номер узла, на котором требуется данный кортеж t. Если кортеж t требуется на локальном узле, то он передается выше по плану. Иначе — передается на узел под номером $\psi(t)$.

Предполагается, что составляющие операторы *split*, *scatter*, *gather* и *merge* также реализованы на основе итераторной модели.

Оператор *split* представляет собой бинарный оператор, который разделяет поступающие из входного потока кортежи на две категории: «свои» и «чужие». «Свои» кортежи должны быть обработаны на текущем процес-

сорном узле и направляются в выходной буфер оператора *split*. «Чужие» кортежи должны быть обработаны на процессорных узлах, отличных от текущего, и помещаются оператором *split* в выходной буфер оператора *scatter*.

Оператор *scatter* определяется как нульарный оператор, который извлекает кортежи из своего выходного буфера, вычисляет для них значение функции пересылки и пересылает их на соответствующие процессорные узлы, используя заданный номер порта обмена. Кортежи между данным оператором и его родительским идут не вверх, а вниз, то есть *split* и *scatter* трактуют итераторную модель нестандартно.

Оператор *gather* представляет собой нульарный оператор, который выполняет чтение кортежей из указанного порта обмена со всех процессорных узлов, отличных от данного, в свой выходной буфер.

Оператор *merge* определяется как бинарный оператор, который поочередно забирает кортежи из выходных буферов своих сыновей и помещает их в собственный выходной буфер.

2.1.3. Построение параллельного плана запроса

Построение параллельного плана запроса предполагает вставку операторов обмена в последовательный план запроса. Оператор обмена вставляется в те места плана, где требуется обеспечить распределение кортежей между вычислительными узлами по некоторому правилу, которое задается функцией пересылки.

На рис. 14 показаны случаи, которые требуют обменов кортежами для получения корректного результата. При выполнении соединения требуется, чтобы оба соединяемых отношения были фрагментированы по атрибуту соединения, поскольку для того, чтобы условие соединения выполнилось, оба сравниваемых кортежа должны располагаться на одном вычислительном узле. Оператор обмена также вставляется в корень плана, чтобы обеспечить сборку конечного результата на одном узле. Поскольку порядок кор-

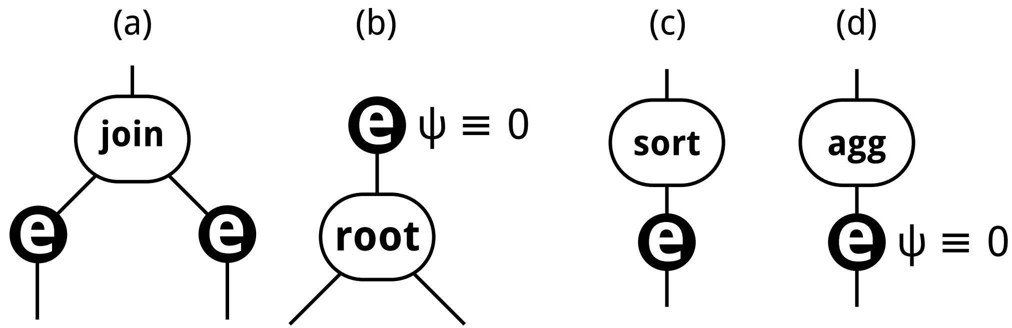


Рис. 14. Вставка оператора обмена

тежей, поступающих от оператора обмена, в общем случае непредсказуем, вставка оператора обмена производится под оператором сортировки. Для него подбирается функция пересылки, зависящая от атрибута, по которому производится сортировка, кроме случая, когда оператор сортировки является корневым. Аналогично производится вставка оператора обмена под оператор агрегации.

2.1.4. Обработка запросов на изменение данных

При обработке запросов на вставку данных необходимо добавлять в корень плана запроса операцию выборки с условием $\psi(t) == i$, где i — номер текущего узла (см. рис. 15). Такое условие отсечет все кортежи, которые

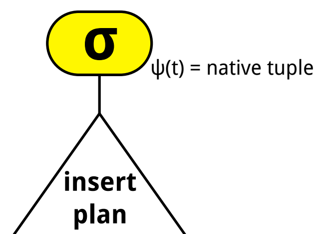


Рис. 15. Параллельный план запроса INSERT

должны быть вставлены на другие вычислительные узлы (см. рис. 16). Таким образом, каждый вставляемый в базу данных кортеж попадет только в один фрагмент базы данных.

При обработке запросов на обновление требуется обеспечить перемещение кортежей, у которых изменилось значение атрибута фрагметнации

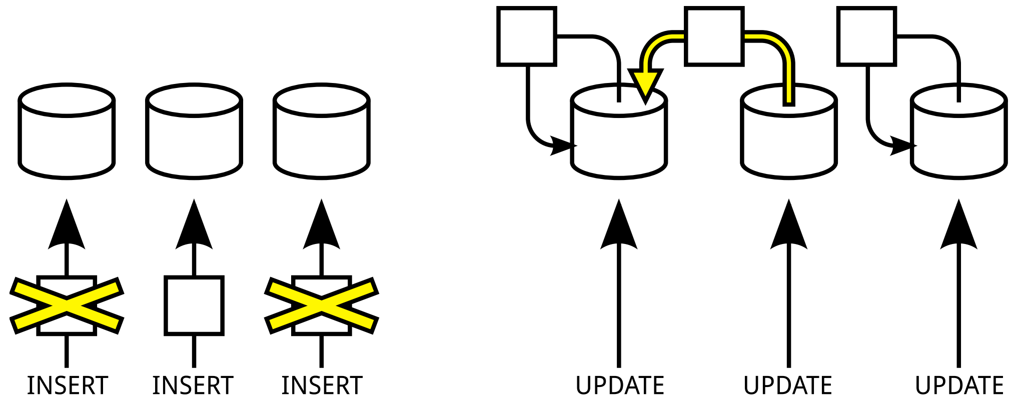


Рис. 16. Вставка и обновление кортежа

(см. рис. 16). Для перемещения таких кортежей в алгоритм работы оператора обмена вносятся модификации. Модифицированный оператор обмена должен обнаруживать кортежи, у которых изменилось значение атрибута фрагментации, и создавать копию таких кортежей. Один экземпляр передается далее по плану с пометкой «удалить», второй экземпляр передается на соответствующий вычислительный узел с пометкой «вставить». Таким образом, модифицированный алгоритм обмена позволяет перемещать кортежи, которые в результате обновления стали «чужими» (если $\varphi(t') \neq \varphi(t)$, то на узле $\varphi(t)$ кортеж t удаляется, а на узле $\varphi(t')$ вставляется t').

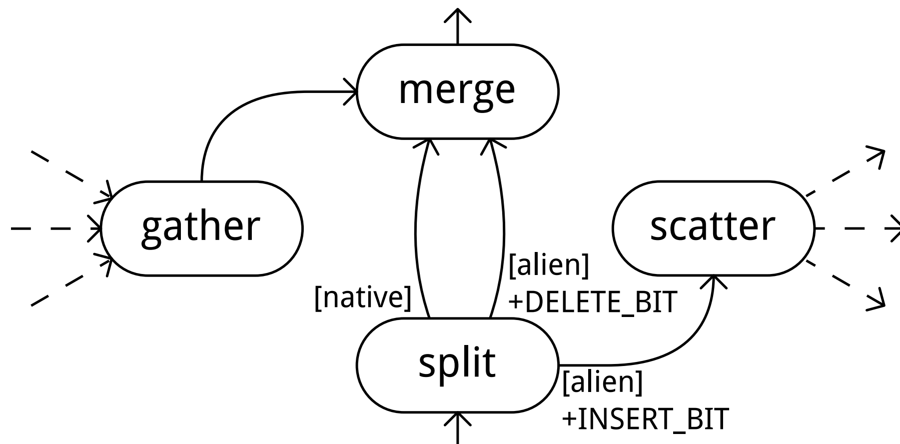


Рис. 17. Поток кортежей в операторе обмена при запросе UPDATE

2.1.5. Хранение метаданных о фрагментации

Чтобы предоставить СУБД информацию о фрагментации таблиц предлагается расширить язык баз данных синтаксическими средствами, который позволят специфицировать функцию фрагментации при создании таблицы. Метод подразумевает добавление нового поля в словарь данных, которое будет определять функцию фрагметации, и использование этого поля при обработке запросов. Наличие метаданных о фрагментации таблицы необходимо при вставке и изменении кортежей, а также может использоваться для построения более оптимального параллельного плана запроса.

2.1.6. Портирование приложений

Метод портирования приложений оригинальной СУБД в параллельную СУБД на ее основе заключается в разработке прикладной библиотеки с интерфейсом, аналогичным оригинальной библиотеке. Новая библиотека реализует тиражирование запроса путем многократного вызова функций из оригинальной библиотеки и, имея идентичный оригинальной библиотеке интерфейс, обеспечивает прозрачную работу приложения с параллельной СУБД. Таким образом, в код приложения не требуется вносить изменений при переходе от последовательной СУБД к параллельной.

2.1.7. Модификация исходных текстов

СУБД представляет собой сложное системное программное обеспечение, исходные тексты которого исчисляются десятками тысяч строк. Отсутствие технологической дисциплины при модификации исходных текстов подобных систем приводит, как правило, к несопровождаемым исходным текстам и, в конечном итоге, — к краху проекта в целом. Предлагаемый метод модификации исходных текстов позволяет минимизировать вносимые в код изменения, инкапсулировав новый код в отдельных подсистемах.

В соответствии с этим при внедрении параллелизма в СУБД внесение изменений в исходные тексты осуществляется следующим образом. Дополнения в структуры данных и алгоритмы инкапсулируются в *новых* файлах исходных текстов, подключаемых к исходным текстам СУБД PostgreSQL.

На рис. 18a показан пример применения данного метода для добавления новых полей в оригинальную структуру данных. В новом файле описывается тип `newstruct`, содержащий новые поля, а в оригинальную структуру добавляется новое поле, имеющее тип данных `newstruct`.

```
// origfile.c
#include "newfile.c"
typedef struct origstruct {
        ...
        newstruct ns;
} origstruct;

// newfile.c
typedef struct newstruct {
        ...
} newstruct;
```

(a) добавление полей в структуру

```
// origfile.c
#include "newfile.c"
int origfunc() {
        ...
        newfunc();
        ...
}

// newfile.c
int newfunc() { ... }
```

(b) добавление вызова в функцию

Рис. 18. Внесение изменений в код

На рис. 18b показан пример применения данного метода для изменения оригинальных алгоритмов. В тело оригинальной функции добавляется вызов новой функции `newfunc()`, а сама функция `newfunc()` определяется в файле исходных текстов новой подсистемы.

2.2. Архитектурные подходы и алгоритмы

Данный раздел содержит описание архитектурных подходов и алгоритмов, реализующих фрагментный параллелизм в рамках свободной СУБД,

на примере СУБД PostgreSQL. Параллельная СУБД, полученная с использованием предложенных методов, названа PargreSQL. Описание организовано следующим образом. Раздел 2.2.1 содержит описание подсистемы, реализованной на основе метода тиражирования запросов. Раздел 2.2.2 посвящен описанию подхода к портированию приложений последовательной СУБД в параллельную СУБД. Алгоритмы реализации оператора обмена приведены в разделе 2.2.3. Реализация параллелизатора запросов обсуждается в разделе 2.2.4. В разделе 2.2.5 описаны алгоритмы реализации запросов на модификацию данных: INSERT, UPDATE и DELETE. Раздел 2.2.6 описывает реализацию операций определения метаданных о фрагметации в СУБД PargreSQL.

2.2.1. Подсистема тиражирования

Подсистема тиражирования *par_libpq-fe* представляет собой надстройку над оригинальной библиотекой `libpq-fe` СУБД PostgreSQL и выполняет отправку запроса множеству серверов, на которых запущены экземпляры СУБД PargreSQL.

Табл. 2. Изменения в API `libpq-fe`

libpq-fe	**par_libpq-fe**
`struct PGconn` Хранит данные о соединении с сервером СУБД PostgreSQL.	`struct par_PGconn` Хранит данные о соединениях с экземплярами СУБД PargreSQL. Представляет собой массив структур `PGconn`.

`PGconn *PQconnectdb(`
`const char *conninfo`
`)`
Устанавливает соединение с сервером СУБД PostgreSQL, указанным в строке подключения `conninfo`.
	`par_PGconn *par_PQconnectdb(`
	`void`
	`)`
	Устанавливает соединение с экземплярами СУБД PargreSQL, указанными в конфигурационном файле.

Табл. 2. Изменения в API `libpq-fe`

libpq-fe	par_libpq-fe

`ConnStatusType PQstatus(`
`  const PGconn *conn`
`)`
Возвращает статус соединения с сервером СУБД PostgreSQL.
	`ConnStatusType par_PQstatus(`
	`  const par_PGconn *conn`
	`)`
	Возвращает статус соединения с экземплярами СУБД PargreSQL. Результат получается с помощью агрегации результатов, полученных в результате вызова `PQstatus()` для всех экземпляров.

`PGresult *PQexec(`
`  PGconn *conn,`
`  const char *query`
`)`
Отправляет запрос серверу СУБД PostgreSQL.
	`PGresult *par_PQexec(`
	`  PGconn *conn,`
	`  const char *query`
	`)`
	Отправляет запрос каждому экземпляру СУБД PargreSQL, циклически вызывая `PQexec()`.

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

Подсистема `par_libpq-fe` имеет интерфейс, аналогичный интерфейсу `libpq-fe`. В табл. 2 приведены отличия интерфейсов библиотек `par_libpq-fe` и `libpq-fe`.

Диаграмма классов, реализующих подсистему `par_libpq-fe`, показана на рис. 19.

2.2.2. Подсистема портирования

Подсистема портирования `par_Compat` позволяет использовать библиотеку `par_libpq-fe` с интерфейсом оригинальной библиотеки `libpq-fe`.

Подсистема портирования реализуется в виде набора макроподстановок компилятора, которые позволяют минимизировать изменения в исходных

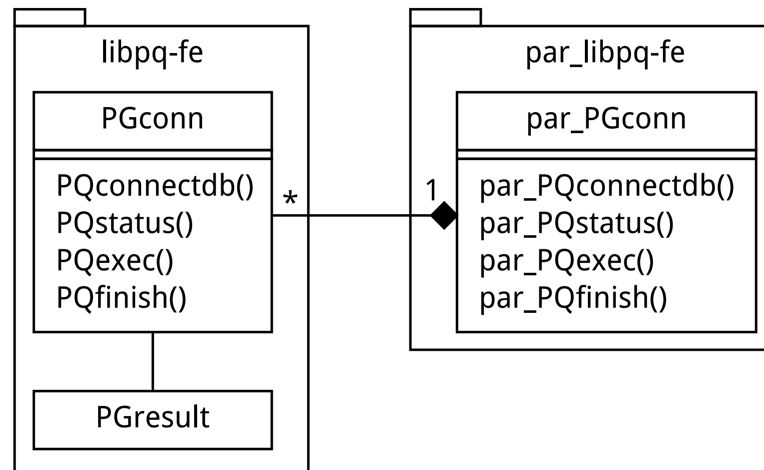


Рис. 19. Классы, реализующие подсистему `par_libpq-fe`

текстах пользовательского приложения для СУБД PostgreSQL при переходе на параллельную СУБД PargreSQL. Данный набор макросов заменяет функции оригинальной библиотеки `libpq-fe` на вызовы соответствующих функций подсистемы тиражирования `par_libp-fe`. Пример использования и часть реализации подсистемы тиражирования приведены на рис. 20.

```
#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. 20. Пример использования подсистемы `par_Compat`

2.2.3. Оператор обмена (*exchange*)

В данном разделе представлена структура и методы реализации оператора *exchange* в параллельной СУБД PargreSQL.

Оригинальный исполнитель запросов PostgreSQL выполняет этот оператор как и любой другой, не подразумевая никакого параллелизма. Параллелизм достигается за счет работы параллелизатора запросов, который размещает операторы *exchange* в плане запросов так, чтобы существующая логика исполнителя запросов привела к корректному результату.

Структура оператора *exchange*

Реализация оператора *exchange* предполагает внедрение в оригинальную СУБД PostgreSQL новых функций и типов данных. На рис. 21 представлен пакет *par_Exchange*, содержащий новые классы, добавляемые в СУБД PostgreSQL.

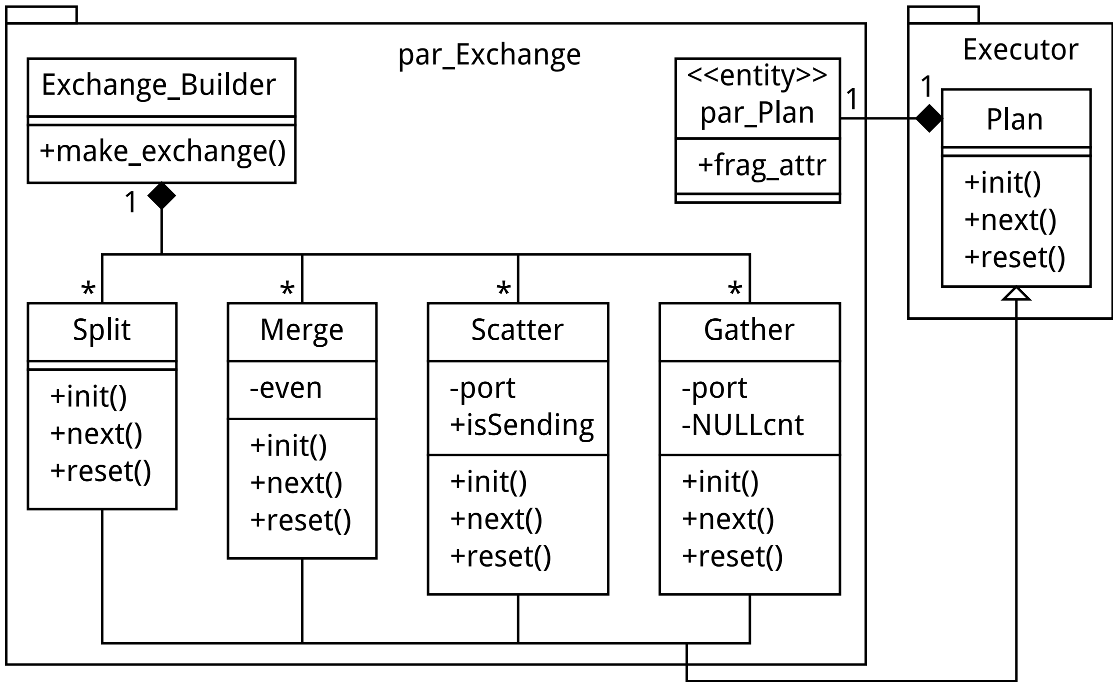


Рис. 21. Диаграмма классов оператора *exchange*

Классы данного пакета *Merge*, *Split*, *Scatter* и *Gather* реализуют одноименные узлы оператора *exchange*. Класс *Exchange_Builder* выполняет функции строителя, предоставляя метод для создания вышеперечисленных узлов плана и формирования из них цельного оператора *exchange*.

Для хранения атрибута фрагментации отношения необходимо выполнить модификацию класса *Plan* в СУБД PostgreSQL, который представляет собой узел плана запроса: в данный класс необходимо добавить целочисленный атрибут `frag_attr`.

Алгоритм реализации метода `next` узла *Split* (см. рис. 22) состоит в следующем. Узел *Split* вызывает метод `next` левого сына и применяет к полученному от него результирующему кортежу функцию пересылки. Ес-

ли функция пересылки показывает, что данный кортеж «свой» (значение функции совпадает с номером текущего вычислительного узла), то узел *Split* завершает работу, возвращая значение этого кортежа в качестве результата. В противном случае данный кортеж помещается в буфер правого сына (узел *Scatter*), осуществляется вызов метода `next` узла *Scatter*, и оператор *exchange* переводится в состояние ожидания.

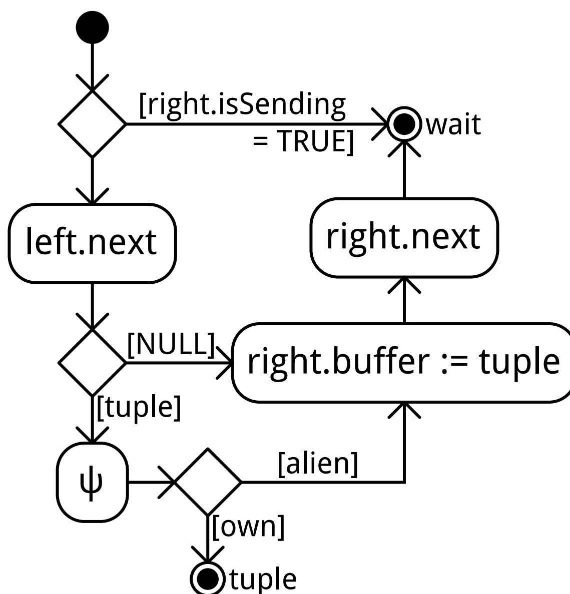


Рис. 22. Метод `next` узла *Split*

На рис. 23 представлен алгоритм метода `next` узла *Merge*. Узел *Merge* попеременно вызывает методы `next` своего левого и правого сыновей – узлов *Gather* и *Split*. Вызовы осуществляются, пока оператор *exchange* находится в состоянии ожидания. Если оба сына вернули пустое значение `NULL`, это означает, что входной поток кортежей исчерпан, и узел *Merge* завершает работу, возвращая значение `NULL`. Если хотя бы один из сыновей возвратил в качестве результата кортеж, то происходит завершение работы узла *Merge*, и этот кортеж возвращается как результат работы.

Алгоритм реализации метода `next` узла *Scatter* показан на рис. 24.

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 ре-

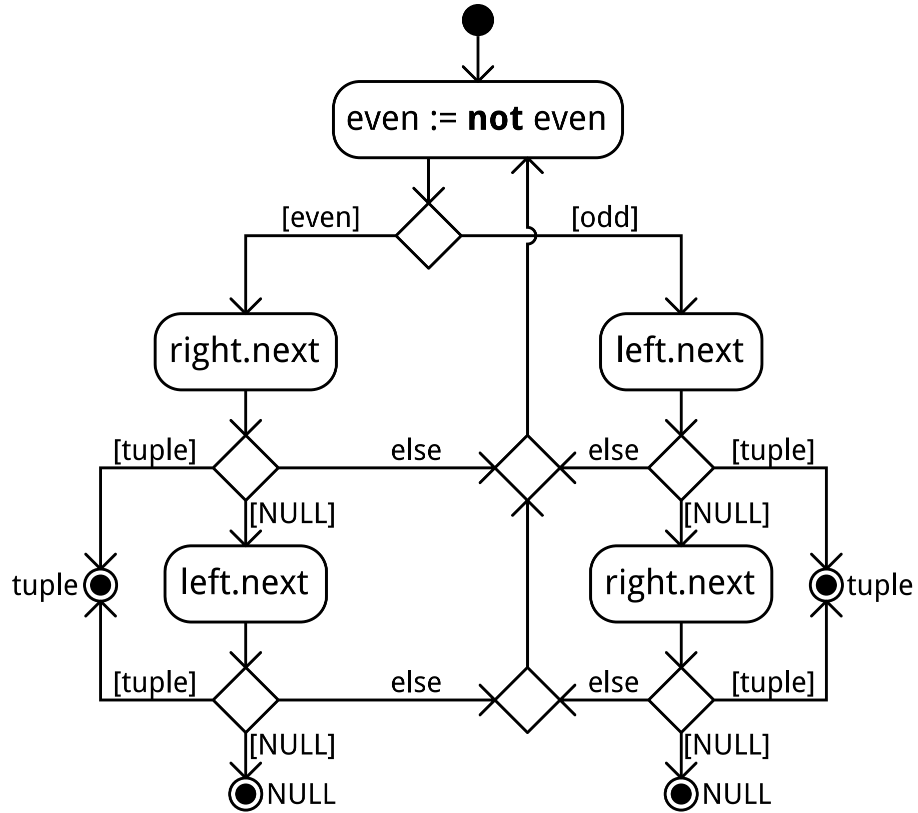


Рис. 23. Метод `next` узла *Merge*

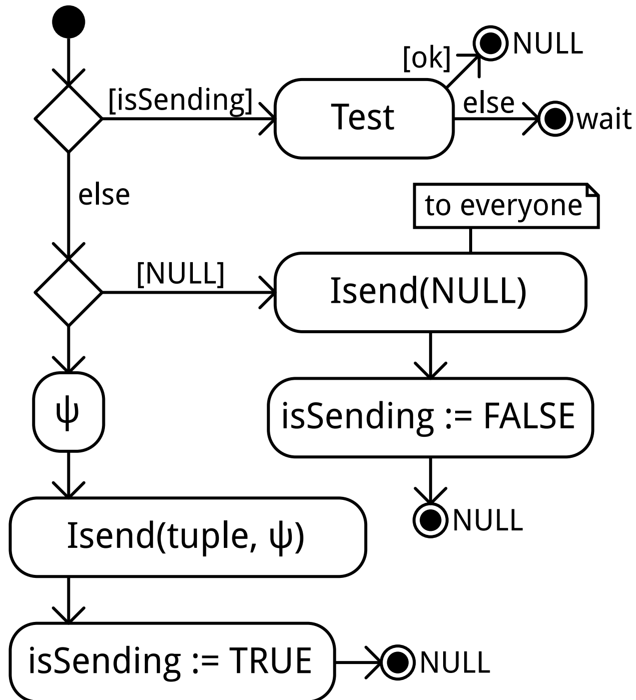


Рис. 24. Метод `next` узла *Scatter*

зультатом применения функции обмена к кортежу. Если во время вызова метода `next` до сих пор отправляется кортеж, то возвращается значение `WAIT`.

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

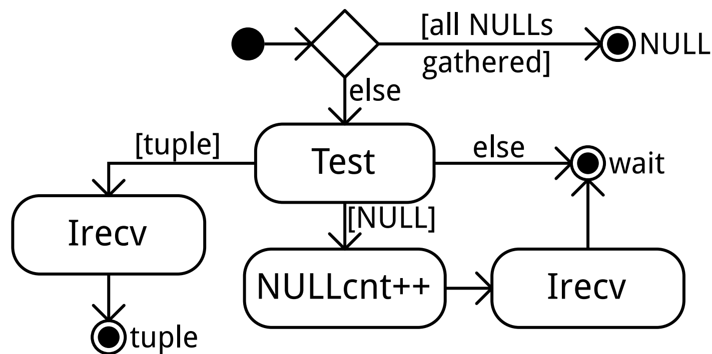


Рис. 25. Метод `next` узла *Gather*

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.

Менеджер сообщений

В настоящее время стандартным подходом в реализации обменов сообщениями в параллельном программировании для систем с распределенной памятью является использование технологии MPI (Message Passing Interface, интерфейс обмена сообщениями) [40]. Однако прямое использование MPI для реализации СУБД PargreSQL существенно затруднено, поскольку в данной системе серверные процессы должны порождаться динамически.

В СУБД PargreSQL для организации обменов сообщениями, реализующих описанные выше операторы *scatter* и *gather*, на основе MPI разработан отдельный *менеджер собщений*. Менеджер сообщений состоит из двух

частей: коммуникатор и библиотека. *Коммуникатор* представляет собой MPI-программу и запускается в виде независимого от СУБД PargreSQL демона в одном экземпляре на каждом вычислительном узле. *Библиотека* предоставляет серверным процессам СУБД PargreSQL интерфейс для подключения к коммуникатору через общую память и организует обмен сообщениями.

Интерфейс библиотеки менеджера сообщений похож на MPI и представлен на рис. 26.

```
// Запустить операцию отправки сообщения.
// Параметры:
// dst -- номер узла-получателя
// port -- порт операции
// size -- длина сообщения
// buf -- указатель на буфер с отправляемым сообщением
// request -- дескриптор операции.
// Возвращаемое значение: 0 в случае успеха или отрицательный код ошибки.
int Isend(dst, port, size, *buf, *request);

// Запустить операцию приема сообщения.
// Параметры:
// port -- порт операции
// size -- длина сообщения
// buf -- указатель на буфер с принятым сообщением
// request -- дескриптор операции.
// Возвращаемое значение: 0 в случае успеха или отрицательный код ошибки.
int Irecv(src, port, size, *buf, *request);

// Проверить завершение заданной операции приема или отправки сообщения.
// request -- дескриптор операции
// flag -- флаг завершения операции (TRUE или FALSE).
// Возвращаемое значение: 0 в случае успеха или отрицательный код ошибки.
int Test(request, *flag);
```

Рис. 26. Интерфейс библиотеки обмена сообщениями

2.2.4. Параллелизатор плана запроса

Подсистема *параллелизатор (Parallelizer)* обеспечивает создание параллельного плана запроса путем вставки операторов *exchange* в нужные места последовательного плана запроса. Необходимость вставки оператора обмена определяется следующим образом.

Параллелизатор осуществляет концевой обход дерева последовательного плана и вставку оператора *exchange* ниже узла соединения в том случае, если атрибут фрагментации сына не совпадает с атрибутом, по которому производится соединение [12]. При этом атрибут фрагментации распространяется по дереву от дочерних узлов к родительским. Таким образом, в каждой точке плана известно, по какому атрибуту фрагментирован результат операции. Соответствующие случаи, которые требуют вставки оператора обмена, представлены на рис. 27.

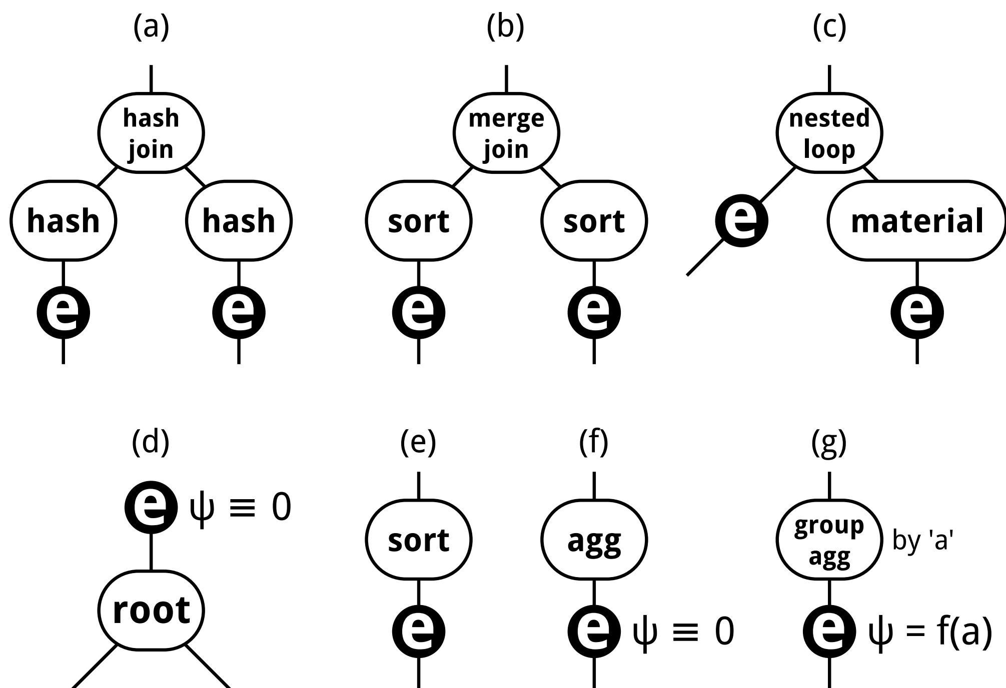


Рис. 27. Вставка оператора *exchange*

При формировании плана запроса в СУБД PostgreSQL используются следующие типы узла, реализующего операцию соединение: соединение хешированием (HashJoin) [91], соединение слиянием (MergeJoin) [93] и соеди-

нение вложенными циклами (NestedLoop) [92]. Вставка оператора обмена в каждом из этих случаев имеет следующие особенности.

Операция HashJoin предполагает формирование хеш-таблицы для каждого из отношений, участвующих в соединении. Узел *HashJoin* имеет два узла-потомка типа *Hash*, каждый из которых выполняет создание хеш-таблицы для своего сына. Вставка оператора обмена осуществляется между узлом *Hash* и его поддеревом (см. рис. 27a), чтобы создание хеш-таблицы осуществлялось после того, как будут получены кортежи, отправленные другими вычислительными узами с помощью оператора *exchange* на текущий вычислительный узел.

Операция MergeJoin предполагает предварительную сортировку отношений, участвующих в соединении. Узел *MergeJoin* имеет два узла-потомка типа *Sort*, каждый из которых выполняет сортировку данных своего сына. Вставка оператора обмена осуществляется между узлом *Sort* и его поддеревом (см. рис. 27b), чтобы сортировка осуществлялась после того, как будут получены кортежи, отправленные другими вычислительными узами с помощью оператора *exchange* на текущий вычислительный узел.

Операция NestedLoop предполагает, что правое отношение полностью считано в память для осуществления его многократного сканирования во внутреннем цикле соединения. Правый сын узла *NestedLoop* — это узел *Material*, выполняющий загрузку данных своего сына в память. Вставка оператора обмена осуществляется между узлом *Material* и его поддеревом (см. рис. 27c), чтобы загрузка данных в память осуществлялась после того, как будут получены кортежи, отправленные другими вычислительными узами с помощью оператора *exchange* на текущий вычислительный узел. Вставка оператора обмена над узлом *Material* приведет к тому, что пересылка кортежей правого отношения будет осуществляться столько раз, сколько кортежей в левом отношении. Это приводит к взаимной блокировке в случае, когда фрагменты левого отношения на разных узлах кластера содержат различное количество кортежей.

На рис. 27d показана вставка оператора обмена в корень плана запроса. В данном случае оператор обмена обеспечивает слияние частичных результатов запроса, полученных на различных вычислительных узлах кластера, на одном вычислительном узле. Оператор обмена, вставляемый в корень плана запроса, имеет функцию пересылки, тождественно равную номеру вычислительного узла, на который осуществляется слияние частичных результатов (например, нулю).

Для получения корректного параллельного плана, помимо вставки оператора обмена в случае операции соединения отношений, требуется также вставка оператора обмена при обработке узлов плана, выполняющих сортировку и агрегацию кортежей.

Операция Sort используется для сортировки поступающих из поддерева кортежей, и, если поместить сразу над ней операцию *exchange*, порядок кортежей нарушится и сортировка будет выполнена впустую. Поэтому в таких случаях (см. рис. 27e) операция *exchange* сдвигается на уровень глубже — под узел *Sort*. Таким образом, сортировка выполняется после обменов и в итоге получается корректный результат.

Операция Agg используется для вычисления агрегирующих функций без группировки в запросах вида `select sum(a) from t`. Поскольку операция агрегации должна обработать кортежи, находящиеся во всех фрагментах отношения, то для получения корректных результатов под узел *Agg* вставляется операция *exchange* (см. рис. 27f) с функцией обмена, тождественно равной номеру вычислительного узла, на котором осуществляется агрегация (например, нулю). Это обеспечивает пересылку всех кортежей на один узел и корректный подсчет агрегирующей функции.

Операция GroupAgg используется для вычисления агрегирующих функций с группировкой в запросах вида `select a, sum(b) from t group by a`. В отличие от предыдущего случая, для правильного выполнения данной операции необходимо обработать не все кортежи отношения — достаточно обработать каждую группу целиком. Поэтому для получения корректных

результатов под узел *Agg* вставляется операция *exchange* с функцией обмена, зависящей от атрибута группировки (см. рис. 27g). Это обеспечивает пересылку всех кортежей из одной группы на один узел и, соответственно, корректное вычисление агрегирующей функции для каждой группы.

2.2.5. Запросы на изменение данных

Запросы на изменение данных (INSERT, DELETE, UPDATE) выполняются СУБД PostgreSQL так же, как и выборки, за исключением финального действия над кортежами, которые получаются в результате. В случае запросов SELECT это действие заключалось в пересылке кортежа клиенту. В случае INSERT — нужно вставить кортеж в таблицу. В случае DELETE — удалить. Запросы UPDATE совмещают в себе удаление старой версии кортежа и вставку новой. Таким образом, алгоритмы, приведенные в разделе 2.2.3, работают только для случая запроса SELECT, а для запросов на изменение данных эти алгоритмы требуется подвергнуть модификации.

В модификации данных алгоритмов используется следующая особенность СУБД PostgreSQL. При исполнении запросов UPDATE или DELETE результирующие кортежи, поступающие из корня плана, имеют системный скрытый атрибут CTID. Атрибут CTID представляет собой адрес кортежа внутри хранилища СУБД PostgreSQL, с помощью которого система определяет, какой кортеж удалить или обновить. Другие атрибуты содержат обновленные значения или, в случае запроса DELETE, просто отсутствуют.

Чтобы запросы DELETE корректно обрабатывались системой PargreSQL, алгоритмы изменять не нужно, однако запросы UPDATE и INSERT требуют новых алгоритмов, поскольку кортеж нужно вставлять только на одном узле, а измененный кортеж может потребовать перемещения с одного узла на другой.

Для реализации запросов UPDATE были внесены дополнительные изменения в операцию *Split* и в код исполнителя.

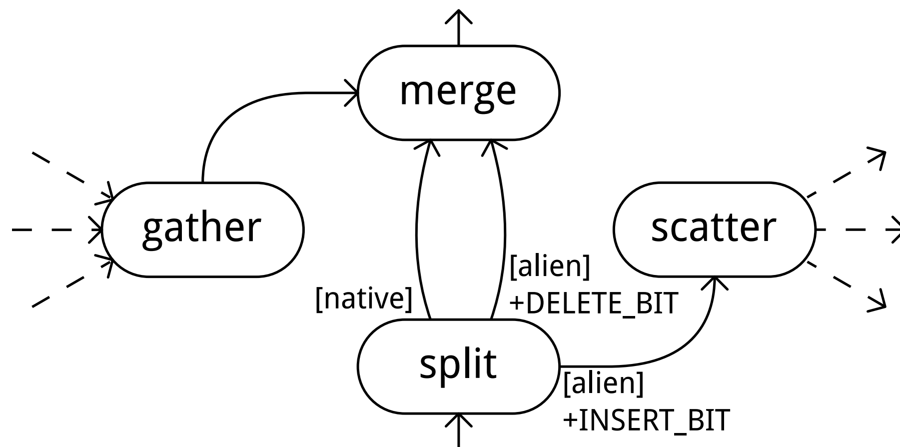


Рис. 28. Поток кортежей в операторе *exchange* при запросе UPDATE

Исполнитель, получая кортежи из корня плана, проверяет биты «delete me» и «insert me» и поступает с кортежами соответствующим образом (удаляя или вставляя их). Таким образом, кортеж, который в ходе изменения атрибутов стал «чужим», будет удален на текущем узле и вставлен — на другом. Если ни один из битов не установлен, то вызывается оригинальная процедура обновления кортежа и СУБД PargreSQL производит обновление локально, как в СУБД PostgreSQL.

Для обычных запросов INSERT параллелизатор добавляет дополнительное условие в корень плана, которое эквивалентно SQL-предложению `WHERE fragattr % nodes == this_node`. С таким условием кортеж будет вставлен только на том узле, где он считается «своим».

Сложные запросы на вставку вида `INSERT INTO dest SELECT columns FROM src` не требуют такого дополнительного условия для правильной работы.

2.2.6. Запросы на определение данных

Чтобы предоставить фрагментацию данных в PargreSQL, в метаданные таблиц СУБД PostgreSQL вводится новый атрибут *fragattr*. Данный

атрибут имеет строковый тип и задает имя атрибута, от которого зависит функция фрагментации соответствующей таблицы. Администратор базы данных должен при создании таблицы задавать значение данного атрибута, иначе невозможно корректно выполнить вставку данных. Атрибут *fragattr* указывается в запросе `CREATE TABLE` с помощью конструкции `WITH`. Пример запроса приведен на рис. 29.

```
create table Person (
    id int,
    name varchar(30),
    gender char(1),
    birth date
) with (fragattr = id);
```

Рис. 29. Создание таблиц в PargreSQL

Атрибут под именем, которое указано в параметре *fragattr* таблицы, будет использован при обработке запросов `UPDATE` и `INSERT` для обеспечения фрагментации по формуле $\varphi(t) = t.\mathrm{fragattr} \bmod N$, где N — количество узлов в кластерной системе, а mod — операция взятия остатка от деления.

2.3. Архитектура параллельной СУБД PargreSQL

Данный раздел содержит описание архитектуры параллельной СУБД PargreSQL, разработанной путем внедрения фрагментного параллелизма в последовательную СУБД PostgreSQL, и организован следующим образом. В разделе 2.3.1 описаны базовые клиентские и серверные процессы СУБД PargreSQL. В разделе 2.3.2 приведена схема параллельной обработки запроса. Раздел 2.3.3 содержит схему модульной структуры и описание семантики подсистем СУБД PargreSQL. Раздел 2.3.4 содержит описание размещения подсистем СУБД PargreSQL на клиенте и сервере.

2.3.1. Взаимодействие процессов СУБД

Архитектура клиент-серверного взаимодействия параллельной СУБД PargreSQL представлена на рис. 30. В отличие от последовательной СУБД PostgreSQL, предполагается, что клиент может взаимодействовать с двумя и более серверами одновременно.

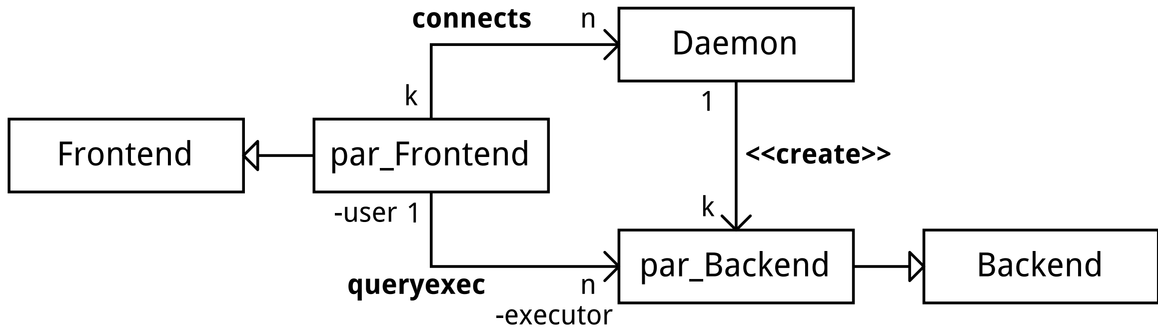


Рис. 30. Процессы СУБД PargreSQL

Реализация компонентов `par_Backend` и `par_Frontend` осуществляется на основе оригинальных компонентов `Backend` и `Frontend` СУБД PostgreSQL соответственно. Компонент `Backend` расширяется в сторону обеспечения обменов кортежами, а в компонент `Frontend` добавляется прозрачное тиражирование запросов для одновременной работы со множеством `Backend`-ов.

Порядок взаимодействия клиентского приложения и СУБД PargreSQL представлен на рис. 31. Взаимодействие осуществляется следующим обра-

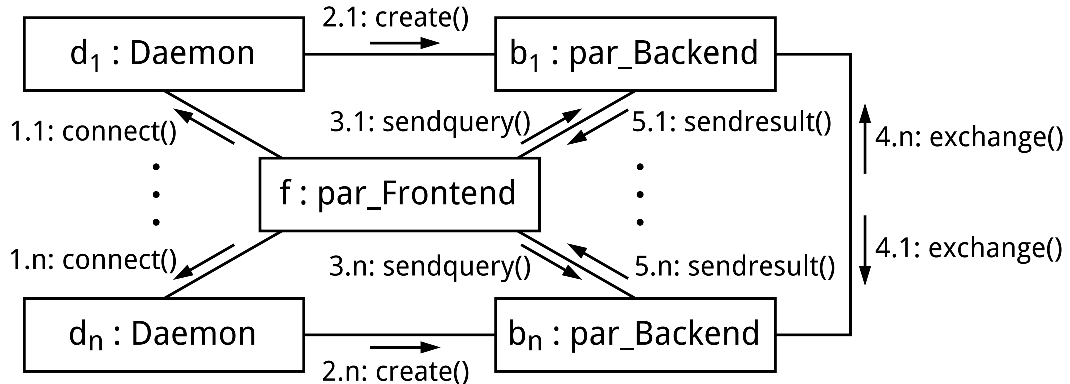


Рис. 31. Взаимодействие клиента и серверов PargreSQL

зом. Клиентское приложение подключается последовательно ко всем демонам СУБД. Результатом подключения является запуск `par_Backend` на каждом узле. Затем клиент последовательно отправляет каждому из этих компонентов запрос. Получив запрос, каждый экземпляр `par_Backend` выполняет его над своим фрагментом базы данных, при этом, возможно, обмениваясь данными с другими экземплярами с помощью оператора *exchange*. По завершении обработки запроса клиентское приложение получает от экземпляров результаты и агрегирует их. Далее мы более подробно рассмотрим, каким образом компонент `par_Backend` осуществляет обработку запроса.

2.3.2. Этапы обработки запроса

Схема параллельной обработки запроса в СУБД PargreSQL представлена на рис. 32.

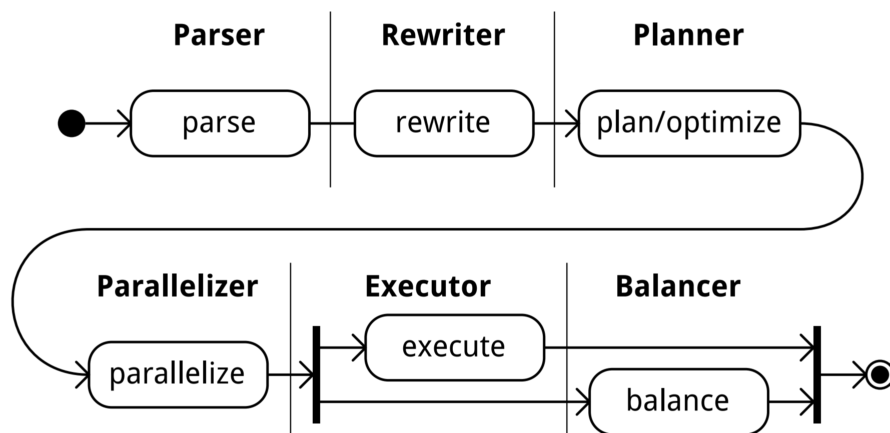


Рис. 32. Обработка запроса в СУБД PargreSQL

Шаги *parse*, *rewrite*, *plan/optimize* и *execute* выполняются так же, как и в последовательной версии СУБД.

На шаге *parallelize* выполняется формирование параллельного плана запроса путем вставки в нужные места последовательного плана операторов *exchange*. Места для вставки оператора *exchange* выбираются паралле-

лизатором после оптимизации и формирования плана запроса, перед его исполнением. Алгоритмы работы параллелизатора описаны в разделе 2.2.4

На шаге *balance* осуществляется балансировка загрузки серверных процессов [26]. Данный шаг осуществляется параллельно с шагом выполнения плана запроса и заключается в переопределении некоторых параметров узлов плана в ходе его выполнения.

2.3.3. Модульная структура

В данном разделе рассмотрена структура подсистем СУБД PargreSQL, реализующих вышеперечисленные шаги параллельной обработки запроса, и их связь с подсистемами оригинальной СУБД PostgreSQL. Модульная структура СУБД PargreSQL представлена на рис. 33. Оригинальная СУБД PostgreSQL рассматривается как подсистема в рамках СУБД PargreSQL.

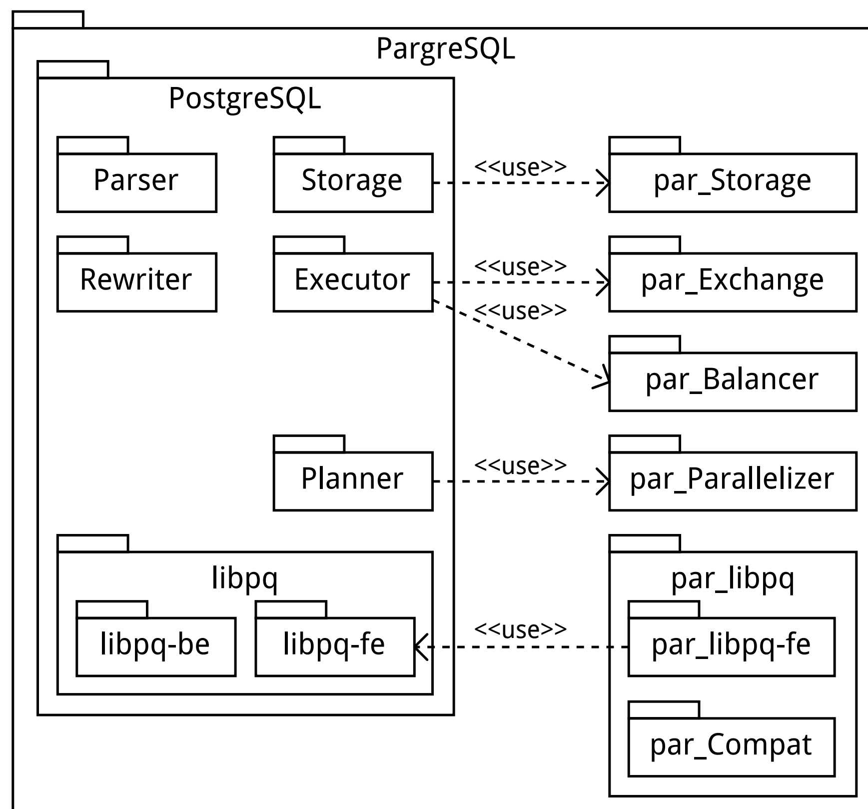


Рис. 33. Архитектура СУБД PargreSQL

Для реализации СУБД PargreSQL необходимо внести изменения в ис-

ходный код следующих подсистем СУБД PostgreSQL: `Storage`, `Executor` и `Planner`. Данные изменения обеспечивают внедрение следующих новых подсистем:

— `par_Storage` — подсистема, обеспечивающая хранение и обработку метаданных о фрагментации отношений;

— `par_Exchange` — подсистема, реализующая оператор *exchange* [12], который обеспечивает пересылку кортежей между экземплярами СУБД во время выполнения запроса;

— `par_Parallelizer` — подсистема, выполняющая добавление в нужные места последовательного плана запроса операторов *exchange*;

— `par_Balancer` — подсистема, выполняющая динамическую балансировку загрузки серверных процессов.

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

— `par_libpq-fe` — надстройка над стандартной библиотекой `libpq-fe` СУБД PostgreSQL, реализующая тиражирование запроса (отправку запроса множеству серверов, на которых запущены экземпляры СУБД PargreSQL);

— `par_Compat` — подсистема портирования, которая реализует прозрачное для приложения подключение библиотеки `par_libpq-fe` и обеспечивает тем самым возможность использования параллельной СУБД PargreSQL вместо оригинальной последовательной СУБД PostgreSQL без существенного изменения исходного текста пользовательского приложения.

2.3.4. Размещение компонентов

Размещение компонентов СУБД PargreSQL приведено на рис. 34.

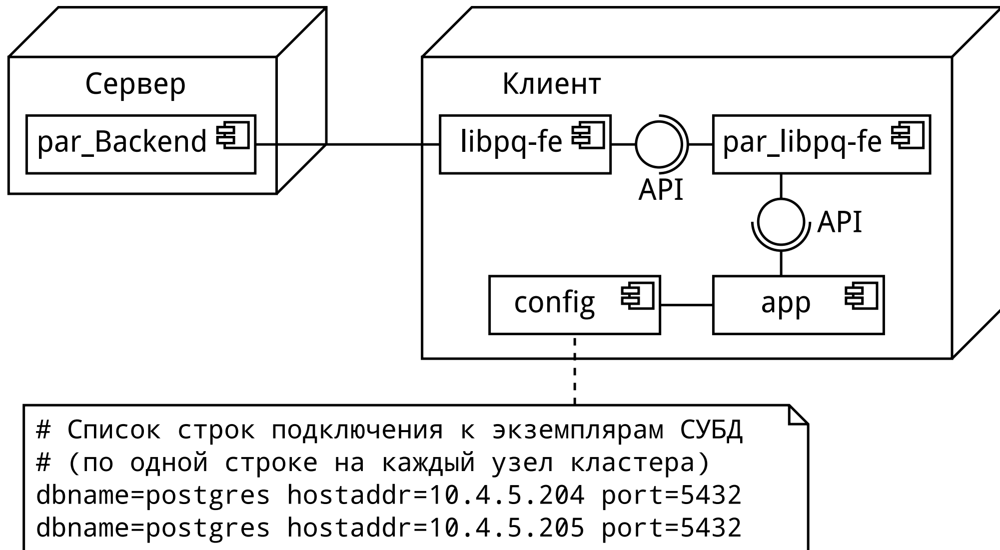


Рис. 34. Размещение компонентов СУБД PargreSQL

На клиенте размещаются библиотеки `par_libpq` и `libpq-fe`, которые пользователь должен подключить к своему приложению, чтобы работать с базой данных. Пользователь также должен создать на клиенте конфигурационный файл, в котором перечисляются параметры доступа к узлам кластера (адрес, порт, имя пользователя, пароль, имя базы данных и др.). Компонент `Backend` размещается на узлах сервера.

2.4. Выводы по главе 2

В данной главе описаны новые методы построения параллельной СУБД для кластерных систем путем внедрения концепции фрагментного параллелизма в свободную СУБД. Данные методы применяется к свободной СУБД PostgreSQL, на основе которой создается параллельная СУБД PargreSQL.

В рамках представленных методов последовательная СУБД рассматривается как подсистема параллельной СУБД. Методы предполагают как внесение изменений в подсистемы оригинальной СУБД, так и реализацию ряда новых подсистем.

Изменениям подвергаются следующие основные подсистемы: подсисте-

ма хранения данных, подсистема построения плана запроса и исполнитель запросов. Изменения в подсистеме хранения данных направлены на поддержку метаданных о фрагментации отношений. Изменения в подсистеме построения плана запроса обеспечивают построение параллельного плана запроса путем вставки в нужные места последовательного плана операторов *exchange*. Оператор *exchange* инкапсулирует в себе параллелизм, реализуя пересылку кортежей между экземплярами СУБД во время выполнения запроса. Модификация исполнителя запросов предполагает реализацию выполнения динамической балансировки загрузки серверных процессов на различных узлах кластера.

Основные новые подсистемы, которые не требуют изменений в исходных текстах подсистем оригинальной СУБД, следующие: подсистема тиражирования запроса и подсистема портирования исходных текстов пользовательских программ. Подсистема тиражирования запроса реализует отправку запроса множеству серверов, на которых запущены экземпляры параллельной СУБД. Подсистема портирования реализует прозрачное подключение подсистемы тиражирования запроса к пользвательским приложениям, написанным для последовательной СУБД.

Описаны алгоритмы реализации оператора обмена и методы его использования при выполнении запросов на выборку, изменение, вставку и удаление данных.

Основные результаты, изложенные в данной главе, опубликованы в работах [4, 5, 7, 8, 10, 70, 71]. На параллельную СУБД PargreSQL получено свидетельство Роспатента об официальной регистрации программы для ЭВМ [15].

Глава 3. Применение параллельной СУБД PargreSQL для интеллектуального анализа графов

В данной главе представлен новый подход к решению задачи разбиения сверхбольших графов, которые состоят из миллионов вершин и ребер, ориентированный на реляционные СУБД с фрагментным параллелизмом. Глава организована следующим образом.

В разделе 3.1 дается краткое описание проблематики интеллектуального анализа графов и приводится формальное определение задачи разбиения графа. В разделе 3.2 описаны существующие в настоящее время подходы к решению данной задачи. Раздел 3.3 содержит описание алгоритма разбиения сверхбольшого графа посредством параллельной СУБД PargreSQL. В разделе 3.4 суммируются основные результаты, полученные в данной главе.

3.1. Определение задачи разбиения графа

Под *интеллектуальным анализом данных (Data Mining)* понимают совокупность методов, алгоритмов и программного обеспечения для обнаружения в данных ранее неизвестных и практически полезных знаний, необходимых для принятия решений в различных сферах человеческой деятельности [43]. Одной из областей приложения технологий Data Mining являются задачи интеллектуального анализа сверхбольших графов, имеющих миллионы вершин и (или) ребер, которые возникают при моделировании сложных структур: химические соединения, белковые структуры, биологические и социальные сети, Web, потоки работ, XML документы и др.

Эффективное разбиение графов имеет большое значение в ряде тео-

ретических и практических задач. В качестве примера теоретических задач можно привести задачи раскраски графа, определения числа и состава компонент связности графа и представления графа в виде ярусно-параллельной формы. Примерами практических задач, в которых необходимо разбиение графа, являются проектирование сложных электронных схем, БИС (больших интегральных схем) и ПЛИС (программируемых логических интегральных схем), проектирование топологии локальной сети, конечно-элементное моделирование и др. Существующие последовательные и параллельные алгоритмы решения данной задачи предполагают возможность размещения графов и промежуточных данных обработки в оперативной памяти и неприменимы для случая графов, имеющих миллионы вершин и (или) ребер.

Задача *разбиения графов (graph partitioning)* представляет собой одну из актуальных задач интеллектуального анализа графов и определяется следующим образом [36]. Пусть имеется граф $G = (N, E)$, где N – множество взвешенных вершин, E – множество взвешенных ребер, и целое число $p > 0$. Требуется найти подмножества вершин N_1, N_2, …, N_p из N такие, что

— $\cup_{i=1}^{p} N_i = N$ и $N_i \cap N_j = \varnothing$ для $i \neq j$;

— $W(i) \approx W/p, i = 1, 2, \ldots, p$, где $W(i)$ и W представляют собой суммы весов вершин N_i и N соответственно;

— величина *разреза (cut size)*, т.е. сумма весов ребер, соединяющих вершины разных подмножеств, минимальна.

3.2. Обзор существующих решений задачи разбиения графа

В настоящее время в области интеллектуального анализа графов опубликовано достаточно большое количество работ и достигнуты значитель-

ные результаты, в том числе в задаче разбиения графов [18].

Классический алгоритм нахождения оптимального разбиения графов предложен в работе [51]. Многоуровневый подход к разбиению графов предложен в работе [48]. Разработано большое количество как последовательных, так и параллельных алгоритмов решения задачи [18]. Один из первых параллельных алгоритмов разбиения графа на основе многоуровневого разбиения предложен в работах [49, 50]. Подход к решению задачи разбиения графа на основе генетических алгоритмов исследован в работе [52]. Алгоритм, не использующий принцип многоуровневого разбиения, описан в работе [34].

В работе [86] обсуждается параллельный алгоритм разбиения графов для систем с общей памятью на основе многоядерных процессоров. В работе [87] представлен параллельный алгоритм многоуровневого разбиения графа, использующий хранение данных на диске. Работа [31] описывает подход к решению задачи разбиения сверхбольших графов на базе концепции облачных вычислений. Распределенный алгоритм разбиения графа представлен в работе [82].

Различные алгоритмы разбиения графа воплощены в программных пакетах METIS и ParMETIS [47], Chaco [46], SCOTCH [75], KaFFPa [81] и др.

Следует отметить также системы обработки графов, использующие распределение данных по узлам многопроцессорной вычислительной системы. Система ParallelGDB [21] представляет собой параллельную систему обработки графов без совместного использования ресурсов, реализованную в виде надстройки над системой DEX [57]. Корпорацией Google разработана вычислительная программная модель Pregel [56] для обработки сверхбольших графов.

Существующие алгоритмы интеллектуального анализа сверхбольших графов сталкиваются с трудностями, связанными с ограниченным размером доступной оперативной памяти для хранения графа и промежуточных

данных алгоритма. Одним из подходов к решению данной проблемы является *использование реляционных СУБД*. Данный подход предполагает перемещение алгоритмов к данным, хранимым в реляционных базах данных, взамен перемещения хранимых данных к утилитам интеллектуального анализа данных третьих разработчиков. Таким образом исключаются накладные расходы, связанные с предварительным экспортом анализируемых данных из базы данных во внешнюю аналитическую утилиту и импортом результатов работы этой утилиты обратно в базу данных. Кроме того, использование реляционных СУБД для интеллектуального анализа данных позволяет получить без накладных расходов все предоставляемые ею сервисы: эффективное управление буферным пулом, быстрый поиск на основе индексирования данных, оптимизация запросов и др. Обратной стороной этого преимущества является сложность разработки структур данных и алгоритмов интеллектуального анализа на языке баз данных SQL с получением громоздких исходных текстов.

Исследования в области использования реляционных СУБД для интеллектуального анализа графов представлены следующими работами. В работе [67] описан масштабируемый подход к анализу структуры графов на основе использования SQL. В работе [84] описан фреймворк, основанный на применении реляционной СУБД для нахождения полного подграфа неориентированного графа. Алгоритмы поиска часто встречающихся подграфов в графе, ориентированные на использование SQL, предложены в работах [29, 38]. Исследования, направленные на поиск циклов в графе с помощью реляционной СУБД, описаны в работах [20, 67]. Работа [58] описывает комбинированный подход к разбиению графов, использующий встраивание запросов SQL в реализацию алгоритма обработки графа на языке программирования высокого уровня.

Для эффективной обработки сверхбольших графов, хранящихся в реляционных базах данных, требуется использование *параллельных* систем баз данных [13], которые обеспечивают параллельную обработку запросов

на многопроцессорных вычислительных системах. Однако проведенный нами обзор литературы показывает, что в настоящее время отсутствует опыт применения параллельных реляционных СУБД для разбиения сверхбольших графов.

3.3. Разбиение графа с помощью СУБД PargreSQL

В данном разделе мы рассмотрим подход к бисекции сверхбольшого графа с помощью параллельной СУБД PargreSQL. *Бисекция* предполагает разбиение графа на два подграфа и является частным случаем исходной задачи (p=2). Разбиение графа на большее количество подграфов выполняется рекурсивным применением бисекции к найденным подграфам [36].

Схема выполнения бисекции сверхбольшого графа с помощью СУБД PargreSQL представлена на рис. 35. Нами используется схема с многоуровневым разбиением [48], состоящая из трех последовательно выполняемых стадий: огрубление, начальное разбиение и уточнение.

Целью стадии *огрубления (coarsening)* является редукция исходного графа до размеров, позволяющих разместить граф и промежуточные данные в оперативной памяти [48]. Огрубление графа выполняется с помощью СУБД PargreSQL. Реляционная таблица, представляющая граф в виде списка ребер, фрагментируется по узлам кластерной системы, и СУБД PargreSQL выполняет запросы SQL, которые сокращают количество вершин и ребер графа.

На стадии *начального разбиения (initial partitioning)* огрубленный граф экспортируется из базы данных и подается на вход сторонней утилиты Chaco [46], которая выполняет начальное разбиение с помощью одного из известных алгоритмов, выбираемых пользователем (классический алгоритм Кернигана и Лина [51], инерциальный алгоритм [83], спектральный алгоритм [76] и др.). Результат начального разбиения импортируется в базу

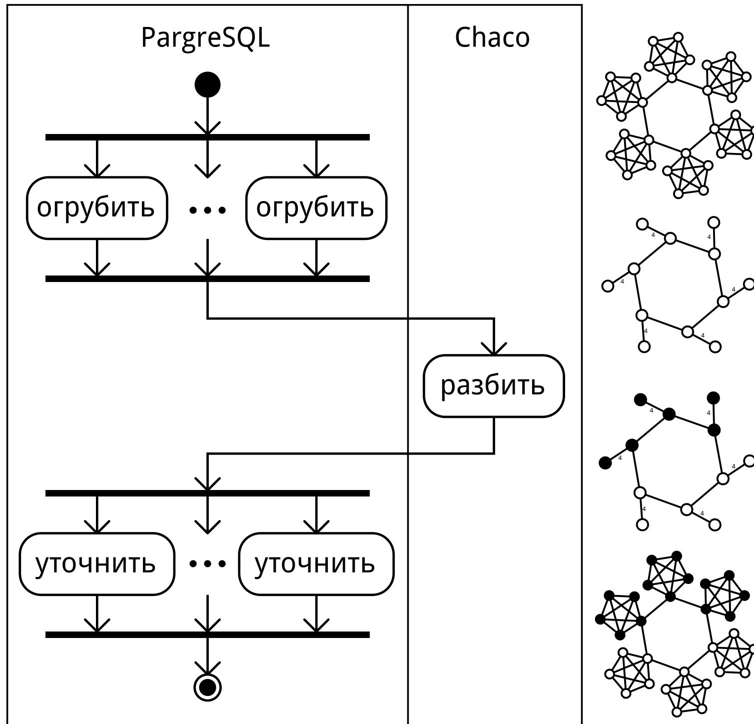


Рис. 35. Бисекция графа с помощью параллельной СУБД PargreSQL

данных в виде реляционной таблицы, содержащей список вершин графа с указанием номера подграфа, которому принадлежит каждая вершина. Далее для удобства изложения вместо номера подграфа, которому принадлежит вершина после бисекции, мы будем говорить о белом или черном *цвете вершины* графа.

На стадии *уточнения разбиения (uncoarsening)* параллельная СУБД PargreSQL с помощью запросов SQL к таблице, полученной на предыдущей стадии, формирует реляционную таблицу из двух столбцов: номер вершины графа и ее цвет.

Ниже мы рассмотрим предлагаемую нами реализацию стадий огрубления и уточнения с помощью параллельной СУБД PargreSQL.

3.3.1. Реляционная схема данных

Данные, обрабатываемые в процессе разбиения графа, представляются в виде реляционных таблиц, указанных в табл. 3. Встречающиеся в данной

таблице термины *наибольшее паросочетание* и *функция выгоды* разъясняются ниже в разделах 3.3.2 и 3.3.3 соответственно.

Табл. 3. Реляционная схема данных, используемых в разбиении графа

№	Реляционная таблица (имя и поля)	Семантика
1	**GRAPH**(A, B, W)	Исходный граф в виде списка ребер A,B: номера концевых вершин ребра, W: вес ребра
2	**MATCH**(A, B)	Наибольшее паросочетание исходного графа A,B: номера концевых вершин ребра
3	**COARSE_GRAPH**(A, B, W)	Огрубленный граф в виде списка ребер A,B: номера концевых вершины ребра, W: вес ребра
4	**COARSE_PARTITIONS**(A, P)	Начальное разбиение огрубленного графа A: номер вершины, P: цвет вершины
5	**PARTITIONS**(A, P, G)	Разбиение исходного графа A: номер вершины, P: цвет вершины, G: значение функции выгоды

При обработке с помощью параллельной СУБД PargreSQL указанные таблицы равномерно распределяются по узлам кластерной вычислительной системы с помощью функции фрагментации $\varphi(t) = \lfloor \frac{t.A \times n}{|E|} \rfloor$, где атри-

бут A используется в качестве атрибута фрагментации и n – это количество узлов в системе. Каждый набор фрагментов этих таблиц на отдельном узле кластера обрабатывается соответствующим экземпляром ядра СУБД.

3.3.2. Огрубление графа

Процедура огрубления графа выполняется нами на основе эвристики, предложенной в работе [48], и состоит из двух последовательно выполняемых шагов: поиск и стягивание. Схема огрубления графа представлена на рис. 36.

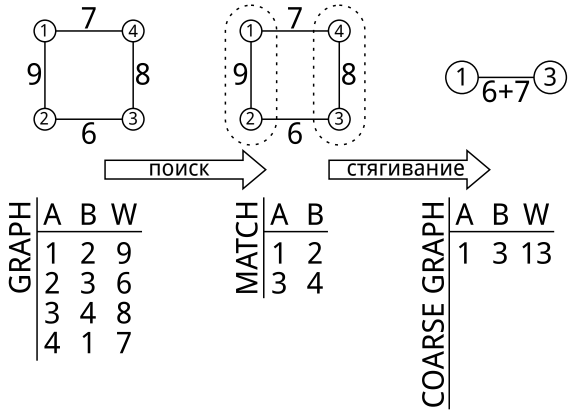


Рис. 36. Схема выполнения огрубления графа

На шаге *поиска* осуществляется нахождение наибольшего паросочетания исходного графа, которое имеет вес, близкий к максимальному. *Паросочетание* – это множество попарно несмежных ребер графа. *Наибольшее паросочетание* представляет собой такое паросочетание, которое не содержится ни в каком другом паросочетании этого графа. Исходный текст реализации шага поиска представлен на рис. 37.

На шаге *стягивания* осуществляется удаление ребер, найденных на шаге поиска. При этом концы удаляемого ребра заменяются одной вершиной, петли уничтожаются, а кратные ребра преобразуются в одно ребро, имеющее вес, равный сумме весов кратных ребер. Исходный текст реализации шага поиска представлен на рис. 38.

```
for edge in (
  select A, B
  from GRAPH
  order by W desc)
loop
  if not exists(
      select *
      from visited
      where A=edge.A or A=edge.B) then
    insert into visited values (edge.A);
    insert into visited values (edge.B);
    insert into MATCH values (edge.A, edge.B);
  end if;
end loop;
```

Рис. 37. Реализация поиска паросочетания

Огрубление повторяется многократно, пока граф не перестанет быть сверхбольшим, например, пока не будет выполнено условие $|E| < L$, где пороговое значение L наперед задано.

В оригинальном алгоритме [48] огрубление осуществляется несколько иным способом: *многократно* выполняется поиск и стягивание ребра с наибольшим весом. Заметим, что поскольку в оригинальном алгоритме при поиске игнорируются ребра, концевые вершины которых являются результатом ранее выполненных операций стягивания, то найденные данным алгоритмом ребра являются паросочетанием. Таким образом, предлагаемый нами алгоритм эквивалентен оригинальному. При этом использование СУБД позволяет выполнить шаг стягивания *однократно* (как один запрос на удаление записей из таблицы).

Огрубленный граф экспортируется из базы данных и подается на вход

```
select
  least(newA, newB) as A,
  greatest(newA, newB) as B,
  sum(W) as W
from (
  select
    coalesce(match2.A, GRAPH.A) as newA,
    coalesce(MATCH.A, GRAPH.B) as newB,
    GRAPH.W
  from
    GRAPH left join MATCH on GRAPH.B=MATCH.B
    left join MATCH as match2 on GRAPH.A=match2.B)
where newA<>newB
group by A, B;
```

Рис. 38. Реализация шага стягивания

сторонней утилиты Chaco [46], которая реализует стадию начального разбиения с помощью алгоритма Кернигана и Лина [51]. Результатом стадии начального разбиения является таблица **COARSE_PARTITIONS** из двух столбцов, в которой для каждой вершины огрубленного графа указан ее цвет.

3.3.3. Уточнение разбиения графа

Процедура уточнения разбиения графа состоит из трех последовательно выполняемых шагов [51]: окрашивание, оценка качества разбиения и улучшение разбиения. Схема уточнения разбиения графа представлена на рис. 39.

На шаге *окрашивания* концам стянутых ребер исходного графа присваи-

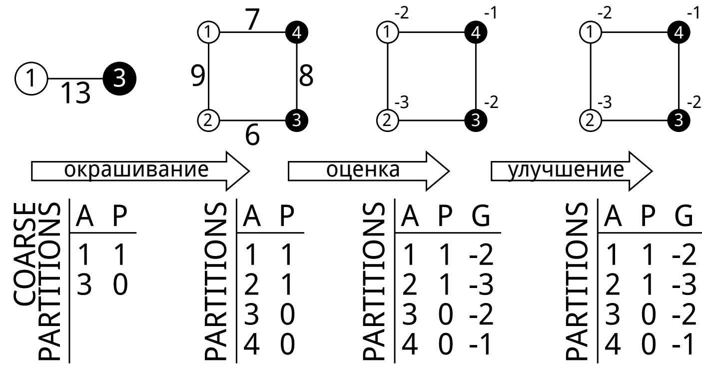


Рис. 39. Схема выполнения уточнения разбиения графа

вается цвет соответствующей вершины огрубленного графа. Окрашивание реализуется как запрос, представленный на рис. 40.

```
insert into PARTITIONS
  select A, P
  from COARSE_PARTITIONS
 union
  select MATCH.B, COARSE_PARTITIONS.P
  from MATCH, COARSE_PARTITIONS
  where MATCH.A = COARSE_PARTITIONS.A;
```

Рис. 40. Реализация шага окрашивания

На шаге *оценки качества разбиения* для каждой вершины графа, полученного на предыдущем шаге, вычисляется *функция выгоды (gain)* [51]:

$$\mathrm{gain}(v) = \mathrm{ext}(v) - \mathrm{int}(v),$$

где

$$\mathrm{ext}(v) = \sum_{(v,u)\in E, P(v)\neq P(u)} w(v,u),$$

$$\mathrm{int}(v) = \sum_{(v,u)\in E, P(v)=P(u)} w(v,u).$$

Значение данной функции показывает, насколько выгодно изменить цвет вершины v на противоположный: если $\mathrm{gain}(v) > 0$, то цвет вер-

шины необходимо изменить. Оценка качества разбиения реализуется как запрос на языке PL/pgSQL, показанный на рис. 41.

```
select
  PARTITIONS.A,
  PARTITIONS.P,
  sum(subgains.Gain) as Gain
from
  PARTITIONS left join (
    select
      GRAPH.A, GRAPH.B,
      case when ap.P=bp.P then -GRAPH.W
        else GRAPH.W
      end as Gain
    from GRAPH left join PARTITIONS as ap
      on GRAPH.a=ap.A
    left join PARTITIONS as bp
    on GRAPH.b=bp.A
  ) as subgains
  on PARTITIONS.A=subgains.A
    or PARTITIONS.A=subgains.B
group by PARTITIONS.A, PARTITIONS.P
```

Рис. 41. Подсчет функции выгоды

На шаге *улучшения разбиения* осуществляется поиск вершины с максимальным положительным значением функции выгоды и инвертирование ее цвета. Поиск и инвертирование повторяются многократно, пока такие вершины существуют. Данные операции реализуются с помощью запросов на языке PL/pgSQL, представленных на рис. 42.

Таким образом, результатом выполнения улучшения разбиения являет-

```
select *
from PARTITIONS
where
  P=current and
  G=(
    select max(G)
    from PARTITIONS
    where P=current)
limit 1
into V
```

(a) поиск вершины

```
update PARTITIONS
  set G = G + W *
    (case when P=V.P then
      2 else -2 end)
from (
  select
    case when A=V.A then
      B else A
    end,
    W
  from GRAPH
  where B=V.A or A=V.A)
    as neighbors
where neighbors.A=PARTITIONS.A;
update PARTITIONS set G=-G, P=1-P
where A=V.A;
```

(b) инвертирование цвета вершины

Рис. 42. Реализация шага улучшения разбиения

ся таблица **PARTITIONS**, столбцы которой покажут номер и цвет соответствующей вершины исходного графа.

3.4. Выводы по главе 3

В данной главе представлен новый подход к решению задачи разбиения сверхбольших графов, которые состоят из миллионов вершин и ребер, основанный на использовании параллельной СУБД PargreSQL. Данный подход может найти применение в решении ряда теоретических и практических задач: раскраска графа, определения числа и состава компонент связности

графа, проектирование больших интегральных схем и топологии локальной сети и др.

Предложенное решение позволяет преодолеть трудности, связанные с ограниченным размером доступной оперативной памяти для хранения графа и промежуточных данных алгоритма, с которыми сталкиваются существующие алгоритмы интеллектуального анализа сверхбольших графов.

Разработанный алгоритм предполагает представление графа в виде реляционной таблицы (списка ребер), распределяемой по узлам кластерной системы. Разбиение состоит из трех последовательно выполняемых стадий: огрубление, начальное разбиение и уточнение. Стадия огрубления выполняется с помощью СУБД PargreSQL и обеспечивает редукцию исходного графа до размеров, позволяющих разместить граф и промежуточные данные в оперативной памяти. На стадии начального разбиения огрубленный граф экспортируется из базы данных и подается на вход сторонней утилиты, которая выполняет начальное разбиение графа в оперативной памяти с помощью одного из известных алгоритмов. Результат начального разбиения импортируется в базу данных в виде реляционной таблицы. На стадии уточнения разбиения параллельная СУБД PargreSQL с помощью запросов к таблице, полученной на предыдущей стадии, формирует финальное разбиение графа в виде реляционной таблицы из двух столбцов: номер вершины графа и номер подграфа, которому принадлежит эта вершина.

Основные результаты, полученные в данной главе, опубликованы в работах [6, 9, 72].

Глава 4. Вычислительные эксперименты

В данной главе отражены результаты вычислительных экспериментов, выполненных с использованием параллельной СУБД PargreSQL, разработанной на основе внедрения фрагментного параллелизма в свободную последовательную СУБД PostgreSQL. Глава организована следующим образом.

В разделе 4.1 описаны план проведения и аппаратная платформа экспериментов. Раздел 4.2 посвящен экспериментам, исследующим ускорение и расширяемость СУБД PargreSQL. В разделе 4.3 приведены результаты экспериментов с СУБД PargreSQL на стандартных тестах производительности, разработанных консорциумом TPC (Transaction Processing Council). Раздел 4.4 отражает эксперименты, в которых с помощью СУБД PargreSQL выполнялось разбиение сверхбольших графов (состоящих из миллионов вершин и ребер). В разделе 4.5 суммируются основные результаты, полученные в данной главе.

4.1. План и аппаратная платформа экспериментов

Целью вычислительных экспериментов являлась оценка эффективности предложенных методов и алгоритмов, реализованных в параллельной СУБД PargreSQL, при обработке синтетических и реальных данных на платформе многопроцессорной вычислительной системы с кластерной архитектурой. В соответствии с этим нами было проведено три серии экспериментов: исследование масштабируемости СУБД PargreSQL, сравнение производительности СУБД PargreSQL с производительностью имеющихся

в настоящее время аналогичных систем и оценка эффективности разбиения сверхбольших графов с помощью СУБД PargreSQL.

Мотивацией проведения экспериментов на оценку масштабируемости и производительности СУБД PargreSQL является то, что основными требованиями к параллельной СУБД являются высокая масштабируемость и высокая производительность [14]. В число основных требований также входит высокая доступность данных, однако эта характеристика не рассматривалась в экспериментах, поскольку основным фактором ее обеспечения является аппаратная отказоустойчивость [14], исследование которой выходит за рамки данной работы.

Масштабируемость означает адекватное увеличение производительности при добавлении в систему дополнительных процессоров, модулей памяти дисков и других аппаратных компонент. *Сравнение производительности* предполагает выполнение СУБД PargreSQL и другими системами некоторого эталонного теста на одной и той же аппаратной платформе. В качестве эталонного теста использовалась спецификация стандартного теста производительности TPC-C, разработанных консорциумом TPC (Transaction Processing Council) [60].

В экспериментах *по оценке эффективности разбиения сверхбольших графов* с помощью СУБД PargreSQL нами исследовалось ускорение и качество разбиения реальных графов, представляющих собой карты дорог Люксембурга и Бельгии и содержащих около 10^5 и 10^6 вершин соответственно.

4.2. Ускорение и расширяемость

В данном разделе приведены результаты экспериментов, исследующих масштабируемость СУБД PargreSQL. Масштабируемость любой многопроцессорной системы определяется эффективностью распараллеливания. Существуют две основные качественные характеристики эффективности рас-

Табл. 4. Аппаратная платформа экспериментов

Характеристика	Значение
Число выч. узлов/процессоров/ядер	736/1472/8832
Тип процессора	Intel Xeon X5680
Оперативная память	3 Тб
Производительность пиковая	117 TFlops
Производительность LINPACK	100.4 TFlops

параллеливания: ускорение и расширяемость, определяемые следующим образом [14].

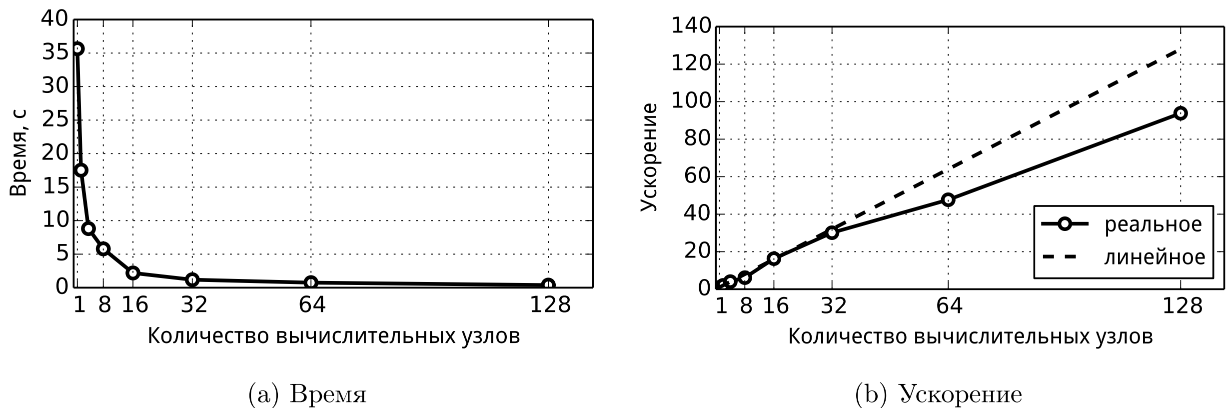
Пусть A и B – две различные конфигурации параллельной машины баз данных с фиксированной архитектурой, различающиеся количеством процессоров и ассоциированных с ними устройств (при этом все конфигурации предполагают пропорциональное наращивание модулей памяти и дисков) и задан некоторый тест Q. Тогда *ускорение* a_{AB}, получаемое при переходе от конфигурации A к конфигурации B, определяется формулой $a_{AB} = t_{QA}/t_{QB}$, где t_{QA} и t_{QB} – это время, затраченное конфигурациями A и B соответственно на выполнение теста Q. Ускорение позволяет определить эффективность наращивания системы на сопоставимых задачах.

Пусть теперь задан набор тестов Q_1, Q_2, ..., количественно превосходящих некоторый фиксированный тест Q в i раз, где i – номер соответствующего теста и конфигурации параллельной машины баз данных A_1, A_2, ..., превосходящие по степени параллелизма (количеству процессоров) некоторую минимальную конфигурацию A в j раз, где j – номер соответствующей конфигурации. Тогда *расширяемость* e_{km}, получаемая при переходе от конфигурации A_k к конфигурации A_m $(k < m)$, определяется формулой $e_{km} = t_{Q_k A_k}/t_{Q_m A_m}$. Расширяемость позволяет измерить эффективность наращивания системы на бóльших задачах.

Говорят, что параллельная система хорошо масштабируема, если она демонстрирует ускорение и расширяемость, близкие к линейным. *Линей-*

ное ускорение означает, что существует константа $k > 0$ такая, что $a_{AB} = kd_B/d_A$ для любых конфигураций A и B (где d – количество процессоров в соответствующей конфигурации). *Линейная расширяемость* означает, что расширяемость остается равной единице для всех конфигураций данной системной архитектуры.

В экспериментах на исследование ускорения СУБД PargreSQL исполняла запрос, предполагающий выполнение операции естественного соединения двух отношений по общему атрибуту. Соединяемые отношения имели размеры 300 млн. и 7.5 млн. кортежей соответственно, распределяемые равномерно по узлам кластера. Результаты данных экспериментов представлены на рис. 43.



(a) Время (b) Ускорение

Рис. 43. Ускорение СУБД PargreSQL

Как видно, СУБД PargreSQL демонстрирует ускорение, близкое к линейному: от 75% до 100% от линейного.

В экспериментах, исследовавших расширяемость, СУБД PargreSQL исполняла запрос, предполагающий выполнение операции естественного соединения двух отношений по общему атрибуту. Кортежи отношений равномерно распределены по узлам кластера. Размеры соединяемых отношений увеличивались пропорционально увеличению количества используемых узлов кластера с множителем 12 млн. и 0.3 млн. кортежей соответственно (т.е. при использовании 128 узлов осуществлялось соединение отношений

из 1 536 млн. и 38.4 млн. кортежей соответственно). Результаты данных экспериментов отображены на рис. 44.

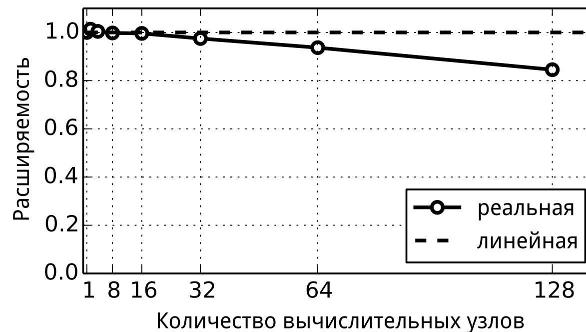


Рис. 44. Расширяемость СУБД PargreSQL

Эксперименты показывают, что расширяемость СУБД PargreSQL близка к линейной: от 85% до 100% от линейной.

4.3. Производительность на тестах TPC

Вторая серия экспериментов была направлена на исследование эффективности СУБД PargreSQL на стандартном тесте производительности TPC-C, разработанном консорциумом TPC (Transaction Processing Council) [60].

Тест TPC-C измеряет производительность СУБД в ходе обработки смеси коротких транзакций, осуществляя моделирование деятельности типичного склада (прием заказов, управлением учетом и распространением товаров и др.). В качестве меры производительности в тесте TPC-C используется коммерческая пропускная способность, отражающая количество обработанных в минуту заказов. Мера производительности выражается пиковой скоростью выполнения транзакций tpm-C (transactions-per-minute-C, количество транзакций в минуту)

В тесте использовалось от 1 до 30 параллельно работающих клиентов, выполняющих запросы к СУБД PargreSQL, запущенной на 12 узлах. Размер базы данных составлял 12 «складов». Результаты исследования эффективности СУБД PargreSQL на тесте TPC-C приведены в табл. 5 в поряд-

ке убывания показателя tpm-C. Данный результат позволяет СУБД PargreSQL попасть в пятерку лидеров рейтинга TPC-C среди параллельных СУБД для кластеров на сентябрь 2013 г. (см. табл. 6).

Табл. 5. Результаты теста TPC-C

К-во клиентов	tpm-C ↓	К-во клиентов	tpm-C ↓	К-во клиентов	tpm-C ↓	К-во клиентов	tpm-C ↓
29	2202531	24	2165413	16	1882353	8	1156626
26	2197183	23	2156250	15	1747572	7	1150684
30	2195122	22	2146341	14	1647058	5	857142
32	2194285	20	2068965	13	1529411	6	847058
27	2189189	19	2054054	12	1358490	4	657534
31	2188235	18	2037735	11	1346938	3	444444
28	2181818	21	2016000	10	1290322	2	328767
25	2173913	17	1961538	9	1270588	1	150000

4.4. Исследование разбиения сверхбольших графов

В данном разделе описаны результаты экспериментов по исследованию эффективности и качества разбиения сверхбольших графов (состоящих из миллионов вершин и ребер)с помощью СУБД PargreSQL.

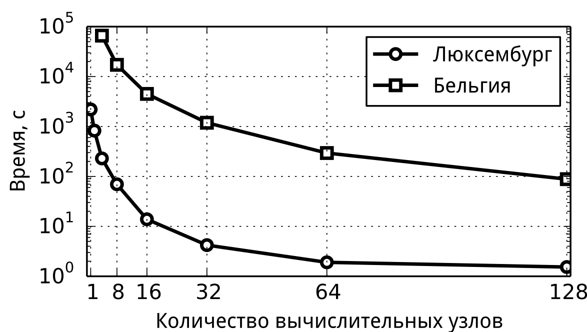
Для разбиения нами были взяты два графа, представляющие собой карты дорог Люксембурга и Бельгии в формате OpenStreetMap[1] и содержащие около 10^5 и 10^6 вершин соответственно. На рис. 45 представлены данные о времени и ускорении разбиения.

Полученное нами сверхлинейное ускорение обусловлено тем, что предложенное решение задачи разбиения графов относится к классу *embarras-*

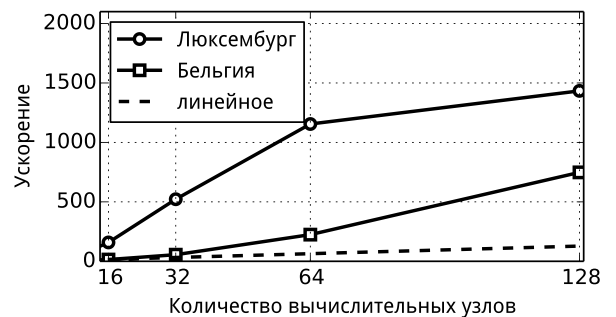
[1] `http://www.cc.gatech.edu/dimacs10/archive/streets.shtml` (дата обращения: 26.04.2013)

Табл. 6. Лидеры TPC-C среди ПСУБД на кластерах

№ п/п	Кластер	СУБД	К-во узлов	К-во клиентов	tpm-C
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



(a) Время



(b) Ускорение

Рис. 45. Время и ускорение разбиения графов

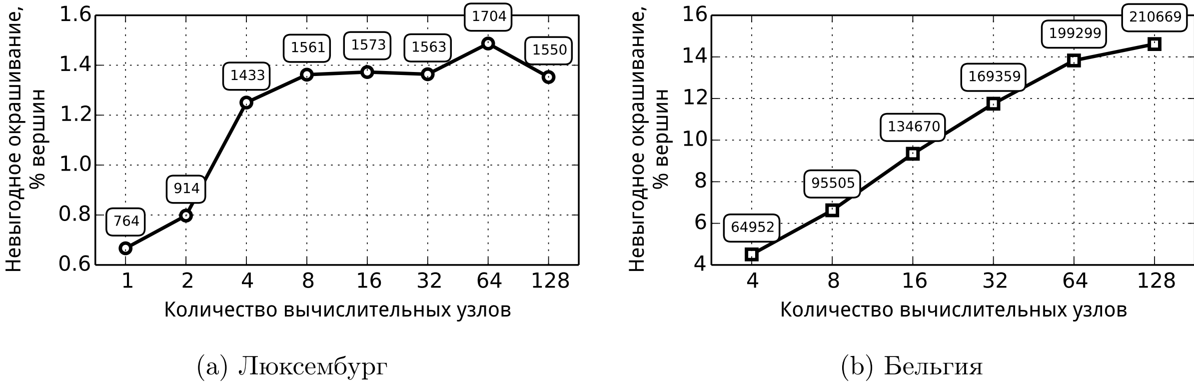
singly parallel («ошеломляюще параллельных») [37], то есть допускает разбиение исходной задачи на неограниченно большое число подзадач, независимых между собой по данным и управлению. Действительно, предложенное решение предполагает, что в соответствии с концепцией фрагментного параллелизма каждый экземпляр СУБД PargreSQL осуществляет обработ-

ку «своих» фрагментов реляционных таблиц, представленных в табл. 3, независимо от других экземпляров, и требуется лишь одна операция пересылки данных для получения итогового результата путем слияния частичных результатов. Поскольку оценка сложности разбиения всего графа совпадает с оценкой сложности разбиения одного фрагмента этого графа и составляет $\underline{O}(|N|^3)$ [36], то при увеличении количества узлов кластерной системы мы получаем сверхлинейное увеличение ускорения.

Качество разбиения графов оценивалось с помощью метрики Кернигана—Лина [51]

$$\text{gain}(v) = \sum_{v,u \in E, P(v) \neq P(u)} W(u,v) - \sum_{v,u \in E, P(v) = P(u)} W(u,v),$$

которая вычисляет для вершины v потенциальную выгоду от перемещения ее в другой подграф. На рис. 46 представлена зависимость количества неверно окрашенных вершин исходного графа (т.е. для которых в результате разбиения было получено отрицательное значение функции выгоды) от количества узлов кластерной системы. Результаты экспериментов показывают, что, хотя качество разбиения обратно пропорционально зависит от количества узлов используемой для разбиения кластерной системы, при этом доля неверно окрашенных вершин является низкой.



(a) Люксембург (b) Бельгия

Рис. 46. Качество разбиения графов

4.5. Выводы по главе 4

В данной главе представлены результаты вычислительных экспериментов с разработанной параллельной СУБД PargreSQL. Эксперименты проводились на 128 узлах суперкомпьютера «Торнадо ЮУрГУ», который входит в список 500 самых мощных вычислителей мира (471 место по версии списка в июне 2013 г.).

В первой серии экспериментов исследовались основные показатели масштабируемости параллельной СУБД: ускорение и расширяемость СУБД PargreSQL. В данных экспериментах СУБД PargreSQL исполняла запрос, предполагающий выполнение операции естественного соединения двух отношений по общему атрибуту.

В экспериментах на исследование ускорения соединяемые отношения имели размеры 300 млн. и 7.5 млн. кортежей соответственно, распределяемые по узлам кластера. Эксперименты показали, что СУБД PargreSQL демонстрирует ускорение, близкое к линейному: от 75% до 100% от линейного.

В экспериментах, исследовавших расширяемость, размеры соединяемых отношений увеличивались пропорционально увеличению количества используемых узлов кластера с множителем 12 млн. и 0.3 млн. кортежей соответственно (т.е. при использовании 128 узлов осуществлялось соединение отношений из 1 536 млн. и 38.4 млн. кортежей соответственно). В ходе экспериментов установлено, что расширяемость СУБД PargreSQL близка к линейной: от 85% до 100% от линейной.

Вторая серия экспериментов была направлена на исследование эффективности СУБД PargreSQL на стандартном тесте производительности TPC-C, разработанном консорциумом TPC (Transaction Processing Council). Тест TPC-C измеряет производительность СУБД на последовательности OLTP-запросов.

В третьей серии экспериментов изучались ускорение и качество раз-

биения реальных сверхбольших графов, выполненного с помощью СУБД PargreSQL. В данных экспериментах выполнялась бисекция двух графов, представляющих собой карты дорог Люксембурга и Бельгии и содержащих 10^5 и 10^6 соответственно. Эксперименты показали, что разбиение выполняется с ускорением, существенно превышающим линейное. Это является ожидаемым результатом, поскольку предложенное решение задачи разбиения графа на основе использования параллельной СУБД относится к классу *embarrassingly parallel* («ошеломляюще параллельных»). При этом, однако, имеет место приемлемое качество разбиения исследуемых графов: не более 1.5% и не более 15% от общего количества вершин графа соответственно были причислены к неверным подграфам.

Таким образом, в ходе экспериментов установлено приемлемое ускорение параллельной СУБД PargreSQL, хотя и меньшее в сравнении с коммерческими аналогами. При этом по сравнению с данными системами СУБД PargreSQL демонстрирует более высокую расширяемость. Проведенные эксперименты также показывают успешное применение СУБД PargreSQL для решения задачи разбиения реальных сверхбольших графов.

Мы можем заключить, что параллельная СУБД PargreSQL представляет собой эффективное и относительно недорогое решение для организации хранения и обработки сверхбольших объемов данных, обладающее хорошей масштабируемостью.

Основные результаты, полученные в данной главе, опубликованы в работах [6, 11, 71, 72].

Заключение

В диссертационной работе была рассмотрена проблема разработки эффективной и масштабируемой параллельной системы управления реляционными базами данных (СУБД) для многопроцессорных вычислительных систем с кластерной архитектурой на основе внедрения фрагментного параллелизма в последовательную СУБД.

Исследованы современные подходы к организации параллельных систем баз данных: фрагментный параллелизм и кластеры баз данных, и рассмотрены соответствующие параллельные СУБД (Oracle Real Application Cluster, MySQL Cluster, ParGRES, HadoopDB и др.). Проведен обзор наиболее популярных на сегодняшний день последовательных реляционных СУБД, свободно распространяемых на уровне исходных текстов (SQLite, MariaDB, MaxDB, Ingres, HSQLDB, BerkeleyDB, Firebird, MySQL и PostgreSQL). На основе данного обзора выбрана СУБД PostgreSQL для внедрения в нее фрагментного параллелизма.

Разработаны новые методы построения параллельной СУБД для кластерных систем путем внедрения фрагментного параллелизма в свободную СУБД. В рамках данных методов последовательная СУБД рассматривается как подсистема параллельной СУБД. Методы предполагают как внесение изменений в подсистемы оригинальной СУБД, так и реализацию ряда новых подсистем.

Изменениям подвергаются следующие основные подсистемы: подсистема хранения данных, подсистема построения плана запроса и исполнитель запросов. Изменения в подсистеме хранения данных направлены на поддержку метаданных о фрагментации отношений. Изменения в подсистеме построения плана запроса обеспечивают построение параллельного плана запроса путем вставки в нужные места последовательного плана операторов *exchange*. Оператор *exchange* инкапсулирует в себе параллелизм, реализуя пересылку кортежей между экземплярами СУБД во время выпол-

нения запроса. Модификация исполнителя запросов предполагает реализацию выполнения динамической балансировки загрузки серверных процессов на различных узлах кластера.

Основные новые подсистемы, которые не требуют изменений в исходных текстах подсистем оригинальной СУБД, следующие: подсистема тиражирования запроса и подсистема портирования исходных текстов пользовательских программ. Подсистема тиражирования запроса реализует отправку запроса множеству серверов, на которых запущены экземпляры параллельной СУБД. Подсистема портирования реализует прозрачное подключение подсистемы тиражирования запроса к пользвательским приложениям, написанным для последовательной СУБД.

На основе данных методов путем внедрения фрагментного параллелизма в свободную СУБД PostgreSQL разработана параллельная СУБД для кластерных вычислительных систем, получившая название PargreSQL. Отлаженный код системы PargreSQL составил около 10 000 строк на языке C (не включая исходные тексты оригинальной СУБД PostgreSQL).

Предложено применение параллельной СУБД PargreSQL для решения задачи разбиения сверхбольших графов (состоящих из миллионов вершин и ребер), имеющей большое значение в ряде теоретических и практических задач (раскраска графа, определения числа и состава компонент связности графа, проектирование больших интегральных схем и программируемых логических интегральных схем, проектирование топологии локальной сети и др.). Разработанный алгоритм предполагает представление графа в виде реляционной таблицы (списка ребер), распределяемой по узлам кластерной системы. Разбиение состоит из трех последовательно выполняемых стадий: огрубление, начальное разбиение и уточнение. Стадия огрубления выполняется с помощью СУБД PargreSQL и обеспечивает редукцию исходного графа до размеров, позволяющих разместить граф и промежуточные данные в оперативной памяти. На стадии начального разбиения огрубленный граф экспортируется из базы данных и подается на вход сторонней утили-

ты, которая выполняет начальное разбиение графа в оперативной памяти с помощью одного из известных алгоритмов. Результат начального разбиения импортируется в базу данных в виде реляционной таблицы. На стадии уточнения разбиения параллельная СУБД PargreSQL с помощью запросов к таблице, полученной на предыдущей стадии, формирует финальное разбиение графа в виде реляционной таблицы из двух столбцов: номер вершины графа и номер подграфа, которому принадлежит эта вершина.

Проведены вычислительные эксперименты, исследующие характеристики разработанной СУБД PargreSQL на реальной кластерной системе с использованием 128 вычислительных узлов. Эксперименты, исследующие масштабируемость СУБД PargreSQL, показали ускорение и расширяемость, близкие к линейным. Эксперименты по разбиению реальных сверхбольших графов с помощью СУБД PargreSQL показали сверхлинейное ускорение предложенного алгоритма при сохранении приемлемого качества разбиения.

Гранты

Работа выполнялась при поддержке Российского фонда фундаментальных исследований (проекты 12-07-31217 мол_а, 12-07-00443, 09-07-00241) и Президента Российской Федерации (стипендия СП-5427.2013.5).

В заключение перечислим основные полученные результаты диссертационной работы, приведем данные о публикациях и апробациях, и рассмотрим направления дальнейших исследований в данной области.

Основные результаты диссертационной работы

На защиту выносятся следующие новые научные результаты:

1. Разработаны новые методы внедрения фрагментного параллелизма в свободную СУБД с открытым исходным кодом.

2. На основе разработанных методов предложены архитектурные подходы и алгоритмы, реализующие фрагментный параллелизм в рамках последовательной СУБД с открытым исходным кодом, на базе которых выполнена параллелизация СУБД PostgreSQL.

3. Разработан новый алгоритм разбиения сверхбольших графов, состоящих из миллионов вершин и ребер, ориентированный на реляционные СУБД с фрагментным параллелизмом.

4. Проведены вычислительные эксперименты с СУБД PargreSQL, которые показали успешное применение данной СУБД для решения задач классов OLAP и OLTP, связанных с обработкой сверхбольших баз данных.

Публикации по теме диссертации

1. *Пан К.С.* Разработка параллельной СУБД на основе PostgreSQL // Труды Института системного программирования РАН. 2011. Т. 21. С. 357–370.

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

3. *Pan C., Zymbler M.* Taming Elephants, or How to Embed Parallelism into PostgreSQL // Database and Expert Systems Applications – 24th International Conference, DEXA 2013, Prague, Czech Republic, August 26–29, 2013. Proceedings, Part I. Springer, 2013. Lecture Notes in Computer Science. Vol. 8055. P. 153–164.

4. *Pan C., Zymbler M.* Very Large Graph Partitioning by Means of Parallel DBMS // Proceedings of the 17th East-European Conference on Advances in Databases and Information Systems, ADBIS'2013, September 1–4, 2013, Genoa, Italy. Lecture Notes in Computer Science. Springer, 2013. Vol. 8133. P. 388–399.

5. *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.

6. *Пан К.С., Цымблер М.Л.* Исследование эффективности параллельной СУБД PargreSQL // Научный сервис в сети Интернет: все грани параллелизма: Труды Международной суперкомпьютерной конференции (Новороссийск, 23–28 сентября 2013 г.). М.: Изд-во МГУ, 2013. С. 148–149.

7. *Пан К.С.* Разработка параллельной СУБД на основе свободной СУБД PostgreSQL // Научный сервис в сети Интернет: экзафлопсное будущее: Труды Международной суперкомпьютерной конференции (Новороссийск, 19–24 сентября 2011 г.). М.: Изд-во МГУ, 2011. С. 566–571.

8. *Пан К.С.* Подход к разбиению сверхбольших графов с помощью параллельных СУБД // Вестник ЮУрГУ. Серия «Вычислительная математика и информатика». 2012. № 47(306). Вып. 2. С. 127–132.

9. *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.

10. *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 суперкомпьютерной конференции (Новороссийск, 20–25 сентября 2010 г.). М.: Изд-во МГУ, 2010. С. 308–313.

11. *Пан К.С., Цымблер М.Л.* Использование параллельной СУБД PargreSQL для интеллектуального анализа сверхбольших графов // Суперкомпьютерные технологии в науке, образовании и промышленности. 2012. № 1. С. 125–134.

12. *Соколинский Л.Б., Цымблер М.Л., Пан К.С., Медведев А.А.* Свидетельство Роспатента о государственной регистрации программы для ЭВМ «Параллельная СУБД PargreSQL» № 2012614599 от 23.05.2012.

Работы 1, 2, 3 и 4 опубликованы в изданиях, включенных ВАК в перечень журналов, в которых должны быть опубликованы основные результаты диссертаций на соискание ученой степени доктора наук.

Апробация работы

Основные положения диссертационной работы, разработанные методы, алгоритмы и результаты вычислительных экспериментов докладывались на следующих международных научных конференциях:

— на международной научной конференции DEXA'2013 (The 24th International Conference on Database and Expert Systems Applications) (Чешская Республика, Прага, 26–30 августа 2013 г.);

— на международной научной конференции ADBIS'2013 (The 17th East-European Conference on Advances in Databases and Information Systems) (Италия, Генуя, 1–4 сентября 2013 г.);

— на международной научной конференции SYRCoDIS'2011 (The 7th Spring Researchers Colloquium on Databases and Information Systems) (Москва, 2–3 июня 2011 г.);

— на Международной суперкомпьютерной конференции «Научный сервис в сети Интернет: все грани параллелизма» (Новороссийск, 23–28 сентября 2013 г.);

— на международной научной конференции «Параллельные вычислительные технологии (ПаВТ'2011)» (Москва, 28 марта – 1 апреля 2011 г.);

— на Международной суперкомпьютерной конференции «Научный сервис в сети Интернет: экзафлопсное будущее» (Новороссийск, 19–24 сентября 2011 г.);

— на Втором Московском суперкомпьютерном форуме (МСКФ'2011), (Москва, 26–27 октября 2011 г.);

— на Международной суперкомпьютерной конференции «Научный сервис в сети Интернет: суперкомпьютерные центры и задачи» (Новороссийск, 20–25 сентября 2010 г.).

Направления дальнейших исследований

Теоретические исследования и практические разработки, выполненные в рамках этой диссертационной работы, предполагается продолжить по следующим направлениям.

1. Аналитическое исследование предложенного алгоритма разбиения графа: получение аналитических оценок качества разбиения графа с учетом характеристик этого графа (например, средняя степень вершины), количества задействованных в разбиении узлов кластера и функции фрагментации.

2. Проведение вычислительных экспериментов по разбиению сверхбольших графов из различных предметных областей (например, социальные сети, химические структуры и др.).

3. Разработка алгоритмов для решения других задач интеллектуального анализа сверхбольших графов посредством использования параллельной СУБД PargreSQL (например, поиск часто встречающихся подграфов).

Литература

1. Когаловский М.Р., Новиков Б.А. Электронные библиотеки – новый класс информационных систем // Программирование. 2000. № 3. С. 3–8.

2. Костенецкий П.С., Лепихов А.В., Соколинский Л.Б. Некоторые аспекты организации параллельных систем баз данных для мультипроцессоров с иерархической архитектурой // Алгоритмы и программные средства параллельных вычислений: (Сб. науч. тр.). УрО РАН. 2006. № 9. С. 42–84.

3. Кузнецов С.Д. MapReduce: внутри, снаружи или сбоку от параллельных СУБД? // Труды Института системного программирования РАН. 2010.

4. Пан К.С. Разработка параллельной СУБД на основе PostgreSQL // Труды Института системного программирования РАН. 2011. Т. 21. С. 357–370.

5. Пан К.С. Разработка параллельной СУБД на основе свободной СУБД PostgreSQL // Научный сервис в сети Интернет: экзафлопсное будущее: Труды международной научной конференции (Новороссийск, 19–24 сентября 2011 г.). Издательство МГУ, 2011. С. 566–571.

6. Пан К.С. Подход к разбиению сверхбольших графов с помощью параллельных СУБД // Вестник Южно-Уральского государственного университета. Серия «Вычислительная математика и информатика». 2012. № 47(306). С. 127–132.

7. 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 междуна-

родной научной конференции (Новороссийск, 20–25 сентября 2010 г.). Издательство МГУ, 2010. С. 308–313.

8. 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.). Издательский центр ЮУрГУ, 2011. С. 577–584.

9. Пан К.С., Цымблер М.Л. Использование параллельной СУБД PargreSQL для интеллектуального анализа сверхбольших графов // Суперкомпьютерные технологии в науке, образовании и промышленности. 2012. № 1. С. 125–134.

10. Пан К.С., Цымблер М.Л. Разработка параллельной СУБД на основе последовательной СУБД PostgreSQL с открытым исходным кодом // Вестник Южно-Уральского государственного университета. Серия «Математическое моделирование и программирование». 2012. № 18(277). С. 112–120.

11. Пан К.С., Цымблер М.Л. Исследование эффективности параллельной СУБД PargreSQL // Научный сервис в сети Интернет: все грани параллелизма: Труды международной научной конференции (Новороссийск, 23–28 сентября 2013 г.). 2013. С. 148–149.

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

13. Соколинский Л.Б. Обзор архитектур параллельных систем баз данных // Программирование. 2004. № 6. С. 49–63.

14. Соколинский Л.Б. Параллельные системы баз данных. Издательство Московского университета, 2013. 184 с.

15. Соколинский Л.Б., Цымблер М.Л., Пан К.С., Медведев А.А. Свидетельство Роспатента о государственной регистрации программы для ЭВМ «Параллельная СУБД PargreSQL» № 2012614599 от 23.05.2012.

16. MySQL: The world's most popular open source database. URL: `http://www.mysql.com/` (дата обращения: 19.08.2013).

17. Abouzeid A., Bajda-Pawlikowski K., Abadi D. et al. HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads // Proc. VLDB Endow. 2009. AUG. Vol. 2, No. 1. P. 922–933.

18. Aggarwal C.C., Wang H. Managing and Mining Graph Data. 1st edition. Springer Publishing Company, Incorporated, 2010. P. 608.

19. Akal F., Böhm K., Schek H. OLAP Query Evaluation in a Database Cluster: A Performance Study on Intra-Query Parallelism // Proceedings of the 6th East European Conference on Advances in Databases and Information Systems. ADBIS '02. London, UK, UK : Springer-Verlag, 2002. P. 218–231.

20. Balachandran R., Padmanabhan S., Chakravarthy S. Enhanced DB-Subdue: supporting subtle aspects of graph mining using a relational approach // Proceedings of the 10th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining. PAKDD'06. Berlin, Heidelberg : Springer-Verlag, 2006. P. 673–678.

21. Bargunó L., Muntés-Mulero V., Dominguez-Sal D., Valduriez P. ParallelGDB: a parallel graph database based on cache specialization // Proceedings of the 15th Symposium on International Database Engineering Applications. IDEAS '11. New York, NY, USA : ACM, 2011. P. 162–169.

22. Baru C.K., Fecteau G., Goyal A. et al. An Overview of DB2 Parallel Edition // SIGMOD Conference. 1995. P. 460–462.

23. Bgelsack A., Gradl S., Mayer M., Krcmar H. SAP MaxDB Administration. SAP PRESS, 2009.

24. Bonchi F., Castillo C., Gionis A., Jaimes A. Social Network Analysis and Mining for Business Applications // ACM TIST. 2011. Vol. 2, No. 3. P. 22.

25. Borrie H. The Firebird Database Developer's Guide. APress, 1970.

26. Bouganim L. Query Load Balancing in Parallel Database Systems // Encyclopedia of Database Systems / Ed. by L. Liu, M.T. Özsu. Springer US, 2009. P. 2268–2272.

27. Bukhres O., Chen J., Elmagarmid A. et al. InterBase: a multidatabase prototype systems // SIGMOD Rec. 1993. JUN. Vol. 22, No. 2. P. 534–539.

28. Cecchet E., Marguerite J., Zwaenepole W. C-JDBC: flexible database clustering middleware // Proceedings of the annual conference on USENIX Annual Technical Conference. ATEC '04. Berkeley, CA, USA : USENIX Association, 2004. P. 26–26.

29. Chakravarthy S., Pradhan S. DB-FSG: An SQL-Based Approach for Frequent Subgraph Mining // Proceedings of the 19th international conference on Database and Expert Systems Applications. DEXA '08. Berlin, Heidelberg : Springer-Verlag, 2008. P. 684–692.

30. Chaudhuri S. An Overview of Query Optimization in Relational Systems // PODS. 1998. P. 34–43.

31. Chen R., Yang M., Weng X. et al. Improving large graph processing on partitioned graphs in the cloud // Proceedings of the Third ACM Symposium on Cloud Computing. SoCC '12. New York, NY, USA : ACM, 2012. P. 3:1–3:13.

32. 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.

33. Dean J., Ghemawat S. MapReduce: simplified data processing on large clusters // Commun. ACM. 2008. JAN. Vol. 51, No. 1. P. 107–113.

34. Delling D., Goldberg A.V., Razenshteyn I., Werneck R.F. Graph Partitioning with Natural Cuts // Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium. IPDPS '11. Washington, DC, USA : IEEE Computer Society, 2011. P. 1135–1146.

35. Evdoridis T., Tzouramanis T. A Generalized Comparison of Open Source and Commercial Database Management Systems // Database Technologies: Concepts, Methodologies, Tools, and Applications. 2009. P. 13–27.

36. Fjällström P. Algorithms for graph partitioning: A survey // Linköping Electronic Articles in Computer and Information Science. 1998. Vol. 3, No. 10. URL: http://www.ep.liu.se/ea/cis/1998/010/cis98010.pdf (online; accessed: 26.04.2013).

37. Foster I.T. Designing and building parallel programs - concepts and tools for parallel software engineering. Addison-Wesley, 1995. P. I–XIII, 1–381.

38. Garcia W., Ordonez C., Zhao K., Chen P. Efficient algorithms based on relational queries to mine frequent graphs // PIKM. 2010. P. 17–24.

39. Grabs T., Böhm K., Schek H. PowerDB-IR — Scalable Information Retrieval and Storage with a Cluster of Databases // Knowl. Inf. Syst. 2004. JUL. Vol. 6, No. 4. P. 465–505.

40. Gropp W. MPI 3 and Beyond: Why MPI Is Successful and What Challenges It Faces // EuroMPI. 2012. P. 1–9.

41. 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.

42. Haldar S. Inside sqlite. O'Reilly Media, Inc., 2007.

43. Han J., Kamber M., Pei J. Data mining: concepts and techniques. Morgan kaufmann, 2006.

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

45. Held G., Stonebraker M., Wong E. INGRES: A Relational Data Base System // AFIPS National Computer Conference. 1975. P. 409–416.

46. Hendrickson B. Chaco // Encyclopedia of Parallel Computing / Ed. by D.A. Padua. Springer, 2011. P. 248–249.

47. Karypis G. METIS and ParMETIS // Encyclopedia of Parallel Computing / Ed. by D.A. Padua. Springer, 2011. P. 1117–1124.

48. Karypis G., Kumar V. Multilevel Graph Partitioning Schemes // ICPP (3). 1995. P. 113–122.

49. Karypis G., Kumar V. Multilevel k-way Partitioning Scheme for Irregular Graphs // J. Parallel Distrib. Comput. 1998. Vol. 48, No. 1. P. 96–129.

50. Karypis G., Kumar V. A Parallel Algorithm for Multilevel Graph Partitioning and Sparse Matrix Ordering // J. Parallel Distrib. Comput. 1998. Vol. 48, No. 1. P. 71–95.

51. Kernighan B.W., Lin S. An Efficient Heuristic Procedure for Partitioning Graphs // The Bell System Technical Journal. 1970. Vol. 49, No. 1. P. 291–307.

52. Kim J., Hwang I., Kim Y., Moon B. Genetic approaches for graph partitioning: a survey // Proceedings of the 13th annual conference on Genetic and evolutionary computation. GECCO '11. New York, NY, USA : ACM, 2011. P. 473–480.

53. Kotowski N., Lima A.A.B., Pacitti E. et al. Parallel query processing for OLAP in grids // Concurr. Comput. : Pract. Exper. 2008. DEC. Vol. 20, No. 17. P. 2039–2048.

54. Lee R., Zhou M. Extending PostgreSQL to Support Distributed/Heterogeneous Query Processing // DASFAA. 2007. P. 1086–1097.

55. 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.

56. Malewicz G., Austern M.H., Bik A.J. et al. Pregel: a system for large-scale graph processing // Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. SIGMOD '10. New York, NY, USA : ACM, 2010. P. 135–146.

57. Martínez-Bazan N., Muntés-Mulero V., Gómez-Villamor S. et al. Dex: high-performance exploration on large graphs for information retrieval // Proceedings of the sixteenth ACM conference on Conference on information and knowledge management. CIKM '07. New York, NY, USA : ACM, 2007. P. 573–582.

58. McCaffrey J.D. A Hybrid System for Analyzing Very Large Graphs // ITNG. 2012. P. 253–257.

59. Mennis J., Guo D. Spatial data mining and geographic knowledge discovery - An introduction // Computers, Environment and Urban Systems. 2009. Vol. 33, No. 6. P. 403–408.

60. Nambiar R.O., Poess M., Masland A. et al. TPC Benchmark Roadmap 2012 // TPCTC. 2012. P. 1–20.

61. Neumann T. Query Optimization (in Relational Databases) // Encyclopedia of Database Systems / Ed. by L. Liu, M.T. Özsu. Springer US, 2009. P. 2273–2278.

62. Newsome M., Pancake C., Hanus J. HyperSQL: Web-based Query Interfaces for Biological Databases // Proceedings of the 30th Hawaii International Conference on System Sciences: Information Systems Track – Internet and the Digital Economy - Volume 4. HICSS '97. Washington, DC, USA : IEEE Computer Society, 1997. P. 329–339.

63. Ngamsuriyaroj S., Pornpattana R. Performance Evaluation of TPC-H Queries on MySQL Cluster // Advanced Information Networking and Applications Workshops, International Conference on. 2010. P. 1035–1040.

64. Oliveira M.D.B., Gama J. An overview of social network analysis // Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery. 2012. Vol. 2, No. 2. P. 99–115.

65. Olson M.A., Bostic K., Seltzer M.I. Berkeley DB. // USENIX Annual Technical Conference, FREENIX Track. 1999. P. 183–191.

66. Pachev S. Understanding MySQL internals - discovering and improving a great database. O'Reilly, 2007. P. I–XVII, 1–234.

67. Padmanabhan S., Chakravarthy S. HDB-Subdue: A Scalable Approach to Graph Mining // Proceedings of the 11th International Conference on Data Warehousing and Knowledge Discovery. DaWaK '09. Berlin, Heidelberg : Springer-Verlag, 2009. P. 325–338.

68. Paes M., Lima A.A., Valduriez P., Mattoso M. High Performance Computing for Computational Science - VECPAR 2008 / Ed. by J.M. Palma,

P.R. Amestoy, M. Daydé et al. Berlin, Heidelberg : Springer-Verlag, 2008. P. 188–200.

69. Page J. A Study of a Parallel Database Machine and its Performance the NCR/Teradata DBC/1012 // BNCOD. 1992. P. 115–137.

70. Pan C. Development of a Parallel DBMS on the Basis of PostgreSQL // SYRCoDIS. 2011. P. 57–61.

71. Pan C.S., Zymbler M.L. Taming Elephants, or How to Embed Parallelism into PostgreSQL // Database and Expert Systems Applications - 24th International Conference, DEXA 2013, Prague, Czech Republic, August 26-29, 2013. Proceedings, Part I. 2013. P. 153–164.

72. Pan C.S., Zymbler M.L. Very Large Graph Partitioning by Means of Parallel DBMS // Advances in Databases and Information Systems — 17th East European Conference, ADBIS 2013, Genoa, Italy, September 1-4, 2013. Proceedings. 2013. P. 388–399.

73. Paulson L.D. Open Source Databases Move into the Marketplace // Computer. 2004. JUL. Vol. 37, No. 7. P. 13–15.

74. Pavlo A., Paulson E., Rasin A. et al. A comparison of approaches to large-scale data analysis // Proceedings of the 2009 ACM SIGMOD International Conference on Management of data. SIGMOD '09. New York, NY, USA : ACM, 2009. P. 165–178.

75. Pellegrini F., Roman J. SCOTCH: A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs // Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking. HPCN Europe 1996. London, UK, UK : Springer-Verlag, 1996. P. 493–498.

76. Pothen A., Simon H.D., Liou K. Partitioning sparse matrices with eigenvectors of graphs // SIAM J. Matrix Anal. Appl. 1990. MAY. Vol. 11, No. 3. P. 430–452.

77. Pruscino A. Oracle RAC: Architecture and Performance // SIGMOD Conference. 2003. P. 635.

78. Rinzivillo S., Turini F., Bogorny V. et al. Knowledge Discovery from Geographical Data // Mobility, Data Mining and Privacy. 2008. P. 243–265.

79. Ronstrom M., Thalmann L. MySQL cluster architecture overview. MySQL Technical White Paper. 2004.

80. Samokhvalov N. XML Support in PostgreSQL // SYRCoDIS. 2007.

81. Sanders P., Schulz C. Engineering Multilevel Graph Partitioning Algorithms // ESA. 2011. P. 469–480.

82. Sanders P., Schulz C. Distributed Evolutionary Graph Partitioning // ALENEX. 2012. P. 16–29.

83. Simon H.D. Partitioning of unstructured problems for parallel processing // Computing Systems in Engineering. 1991. Vol. 2, No. 2. P. 135–148.

84. Srihari S., Chandrashekar S., Parthasarathy S. A Framework for SQL-Based Mining of Large Graphs on Relational Databases // Proceedings of the 10th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining. PAKDD. 2010. P. 160–167.

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

86. Sui X., Nguyen D., Burtscher M., Pingali K. Parallel Graph Partitioning on Multicore Architectures // LCPC. 2010. P. 246–260.

87. Trifunovic A., Knottenbelt W.J. Towards a Parallel Disk-Based Algorithm for Multilevel k-way Hypergraph Partitioning // IPDPS. 2004.

88. Waas F.M. Beyond Conventional Data Warehousing - Massively Parallel Data Processing with Greenplum Database // BIRTE (Informal Proceedings). 2008.

89. White T. Hadoop: The definitive guide. O'Reilly Media, Inc., 2012.

90. Zhang M. Application of Data Mining Technology in Digital Library // JCP. 2011. Vol. 6, No. 4. P. 761–768.

91. Zhou J. Hash Join // Encyclopedia of Database Systems / Ed. by L. Liu, M.T. Özsu. Springer US, 2009. P. 1288–1289.

92. Zhou J. Nested Loop Join // Encyclopedia of Database Systems / Ed. by L. Liu, M.T. Özsu. Springer US, 2009. P. 1895.

93. Zhou J. Sort-Merge Join // Encyclopedia of Database Systems / Ed. by L. Liu, M.T. Özsu. Springer US, 2009. P. 2673–2674.

94. Zhou S., Williams M.H. Data placement in parallel database systems // Parallel Database Techniques. 1996. Vol. 10. P. 203–219.

95. About MariaDB. URL: `https://kb.askmonty.org/en/about-mariadb/` (online; accessed: 19.08.2013).

96. The Architecture Of SQLite. URL: `http://www.sqlite.org/arch.html` (online; accessed: 19.08.2013).

97. Encyclopedia of Database Systems, Ed. by L. Liu, M.T. Özsu. Springer US, 2009.

98. Encyclopedia of Parallel Computing, Ed. by D.A. Padua. Springer, 2011.

Приложение. Статистические данные о популярности современных свободных СУБД

Табл. 7. Количество найденных web-страниц

Система	Google (млн)	Job (млн)	Книги	Страницы Gogle Code	Проекты Sourceforge	Проекты Freshmeat	Проекты GitHub
PostgreSQL	36	24	1244	48000	625	546	2156
MySQL	237	68	4636	32000	2525	1631	8351
MariaDB	1.9	0.5	7	616	5	4	76
MaxDB	0.7	0	59	1290	2	6	8
SQLite	11.9	3.6	837	139000	575	223	2527
Firebird	22.8	5.8	64	13800	75	53	120
Ingres	3.2	1	198	3670	8	5	163
BerkeleyDB	1	0	23	5750	0	12	57
HSQLDB	0.9	0.1	122	45100	7	18	57

Табл. 8. Нормированные данные о популярности СУБД

Система	Google	Job	Книги	Страницы Gogle Code	Проекты Sourceforge	Проекты Freshmeat	Проекты GitHub	$\sum$
PostgreSQL	0.15	0.35	0.27	0.35	0.25	0.33	0.26	1.96
MySQL	1.00	1.00	1.00	0.23	1.00	1.00	1.00	6.23
MariaDB	0.01	0.01	0.00	0.00	0.00	0.00	0.01	0.03
MaxDB	0.00	0.00	0.01	0.01	0.00	0.00	0.00	0.03
SQLite	0.05	0.05	0.18	1.00	0.23	0.14	0.30	1.95
Firebird	0.10	0.09	0.01	0.10	0.03	0.03	0.01	0.37
Ingres	0.01	0.01	0.04	0.03	0.00	0.00	0.02	0.12
BerkeleyDB	0.00	0.00	0.00	0.04	0.00	0.01	0.01	0.06
HSQLDB	0.00	0.00	0.03	0.32	0.00	0.01	0.01	0.38