МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования

**«Южно-Уральский государственный университет»**

**(национальный исследовательский университет)**

**Факультет Вычислительной математики и информатики**

**Кафедра системного программирования**

|  |  |
| --- | --- |
| РАБОТА ПРОВЕРЕНА  Рецензент  <ученая степень, ученое звание>  \_\_\_\_\_\_\_\_\_\_ <И.О. Фамилия рецензента>  “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2014 г. | ДОПУСТИТЬ К ЗАЩИТЕ  Заведующий кафедрой, д.ф.-м.н., профессор  \_\_\_\_\_\_\_\_\_\_ Л.Б. Соколинский  “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2014 г. |

**Реализация теста GRAPH500 для параллельной СУБД PargreSQL**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ЮУрГУ – 010400.62.2014.08-036-15701.ВКР

|  |  |
| --- | --- |
|  | Научный руководитель  кандидат физ.-мат. наук, доцент  \_\_\_\_\_\_\_\_\_\_\_\_\_М. Л. Цымблер  Автор работы, студент группы ВМИ-456  \_\_\_\_\_\_\_\_\_\_\_\_\_А. Ю. Сафонов  Ученый секретарь  (нормоконтролер)  \_\_\_\_\_\_\_\_\_\_\_\_\_ О. Н. Иванова  “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2014 г. |

Челябинск-2014

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования

**«Южно-Уральский государственный университет»**

**(национальный исследовательский университет)**

**Факультет Вычислительной математики и информатики**

**Кафедра системного программирования**

УТВЕРЖДАЮ

Зав. кафедрой СП

\_\_\_\_\_\_\_\_\_\_ Л.Б. Соколинский

08.02.2014

**ЗАДАНИЕ1**

**на выполнение выпускной квалификационной работы бакалавра**

студенту группы ВМИ-356 Сафонову Александру Юрьевичу,

обучающемуся по направлению 010400.62 «Информационные технологии»

1. **Тема работы** (утверждена приказом ректора от <дата и номер приказа>)

Реализация теста Graph500 для параллельной СУБД PargreSQL.

1. **Срок сдачи студентом законченной работы:** 04.06.2014.
2. **Исходные данные к работе2**
3. Пан К.С., Цымблер М.Л. Использование параллельной СУБД PargreSQL для интеллектуального анализа сверхбольших графов // Суперкомпьютерные технологии в науке, образовании и промыш-ленности, 2012. № 1. С. 125-134.
4. Соколинский Л.Б. Параллельные системы баз данных. М.: Издательство Московского университета, 2013. 184 с.
5. **Перечень подлежащих разработке вопросов3**
6. Изучить архитектуру параллельной СУБД PargreSQL и спецификацию теста Graph500.
7. Разработать схему базы данных для хранения графа и промежуточных данных в соответствии со спецификацией теста Graph500.
8. Выполнить проектирование и разработку алгоритмов на языке SQL, реализующих тест Graph500 для параллельной СУБД PargreSQL.
9. Провести вычислительные эксперименты на суперкомпьютере “Торнадо ЮУрГУ”, исследующие эффективность параллельной СУБД PargreSQL на тесте Graph500.
10. **Дата выдачи задания:** 08.02.2014.

**Научный руководитель4**

Зам. заведующего кафедрой СП,

кандидат физ.-мат. наук М. Л. Цымблер

**Задание принял к исполнению** А. Ю. Сафонов

Оглавление

[Введение 5](#_Toc388968320)

[1. Описание существующих технологий 7](#_Toc388968321)

[1.1. СУБД PargreSQL 7](#_Toc388968322)

[1.2. Тест вычислительной производительности LINPACK 10](#_Toc388968323)

[1.3. Тест оценки производительности систем баз данных TPC 10](#_Toc388968324)

[1.4. Тест Graph500 11](#_Toc388968325)

[2. Алгоритмы, реализующие тест Graph500 для параллельной СУБД PargreSQL 15](#_Toc388968326)

[2.1. Разработка требований к системе Pargraph500 15](#_Toc388968327)

[2.2. Модульная структура 16](#_Toc388968328)

[2.3. Схема базы данных 17](#_Toc388968329)

[2.4. Описание таблицы на языке SQL 18](#_Toc388968330)

[2.5. Генерация списка ребер 19](#_Toc388968331)

[2.6. Генерация внутреннего представления графа 19](#_Toc388968332)

[2.7. Алгоритм обхода графа в ширину 21](#_Toc388968333)

[3. Вычислительные эксперименты 24](#_Toc388968334)

[3.1. Аппаратная платформа экспериментов 24](#_Toc388968335)

[3.2. План экспериментов 24](#_Toc388968336)

[3.3. Результаты 25](#_Toc388968337)

[Заключение 26](#_Toc388968338)

[Литература 27](#_Toc388968339)

Введение

Актуальность исследования

Большинство существующих тестов производительности суперкомпьютеров направлены на измерение производительности вычислительных мощностей (тесты LINPACK, HPCC RandomAccess). Однако они не позволяют адекватно измерить производительность на задачах с интенсивной обработкой данных (Data Intensive). Примерами таких задач могут служить задачи на сверхбольших графах. Чтобы иметь возможность улучшать производительность на задачах данного типа, требовалось разработать новый сравнительный тест (бенчмарк), который бы учитывал специфику задач с интенсивной обработкой данных. В 2010 около 30 международных экспертов в области высокопроизводительных вычислений из научной сферы и сформировали специальный комитет, с помощью которого была разработана спецификация подобного теста, названного Graph500. Целью данного теста является разработка приложения, состоящего из нескольких основных анализирующих частей (так называемых «ядер»), которые обращаются к единой структуре данных, представляющей собой взвешенный, ненаправленный граф. Текущая спецификация теста содержит 2 ядра, одно ядро из входного списка ребер создает представление графа, которое будет использоваться в дальнейшем, а второе — производит вычисления на этом графе.

Цель и задачи работы

Целью моей работы является оценка эффективности параллельной СУБД PargreSQL на задачах интенсивной обработки данных с помощью сравнительного теста Graph500.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучить архитектуру параллельной СУБД PargreSQL и спецификацию теста Graph500.
2. Разработать схему базы данных для хранения графа и промежуточных данных в соответствии со спецификацией теста Graph500.
3. Выполнить проектирование и разработку алгоритмов на языке SQL, реализующих тест Graph500 для параллельной СУБД PargreSQL.
4. Провести вычислительные эксперименты на суперкомпьютере «Торнадо ЮУрГУ», исследующие эффективность параллельной СУБД PargreSQL на тесте Graph500.

Структура и объем работы

Работа состоит из введения, трех глав, заключения и библиографии. Объем работы составляет ? страниц, объем библиографии – ? источников.

Содержание работы

Первая глава, «Описание существующих технологий», описывает предметную область, в рамках которой ведется данная работа.

Вторая глава, «Алгоритмы, реализующие теста Graph500 для параллельной СУБД PargreSQL», описывает разработку реализации теста Graph500.

Третья глава, «Вычислительные эксперименты», описывает проведенные эксперименты и объясняет полученные результаты.

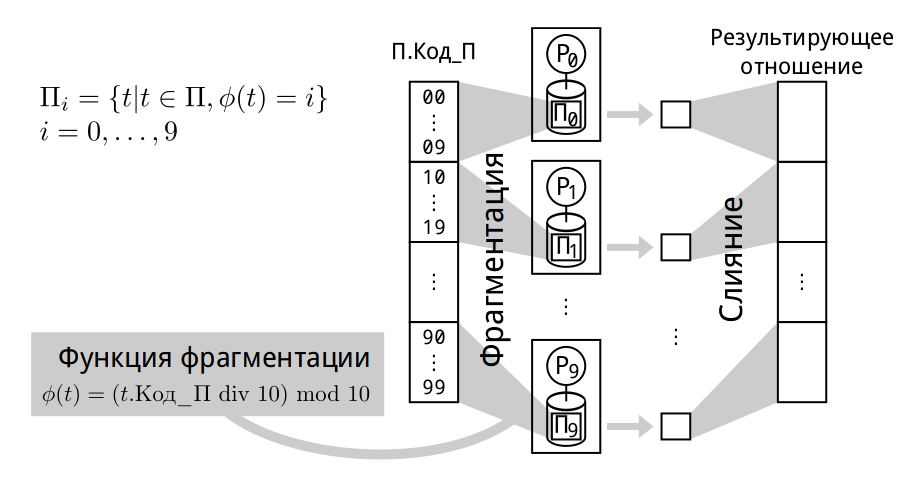
В заключении описываются основные результаты, полученные при выполнении выпускной квалификационной работы.

1. Описание существующих технологий

В данном разделе рассмотрена параллельная СУБД PargreSQL а также некоторые из существующих технологий оценки производительности систем: тесты LINPACK и TPC. Наиболее подробно рассмотрена спецификация теста Graph500.

* 1. СУБД PargreSQL

Базисной концепцией параллельной обработки запросов в реляционных системах баз данных является фрагментный параллелизм [9]. Принципиальная схема обработки запроса с использованием фрагментного параллелизма выглядит следующим образом (см. ).



**Рис. 1.** Параллельная обработка запроса на основе фрагментного параллелизма

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 [3], каждый из которых обрабатывает отдельный фрагмент отношения на выделенном ему процессорном узле. Полученные агентами результаты сливаются в результирующее отношение.

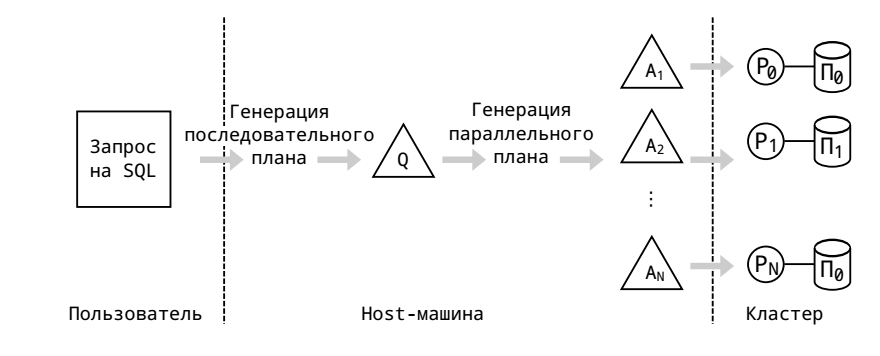
Функция фрагментации отношения

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

где является некоторой функцией, определенной на домене атрибута . 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 получения корректного результата необходимо выполнять пересылки кортежей между процессорными узлами. Для организации таких пересылок в соответствующие места дерева плана запроса вставляется оператор *exchange*.

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).



**Рис. 2.** Схема обработки запроса в параллельной СУБД.  − последовательный физический план,  − параллельный агент,  ‑‑ вычислительный узел,  − диск

Общая схема организации обработки запросов в параллельной СУБД выглядит следующим образом [3]. Мы будем полагать, что вычислительная система представляет собой кластер из вычислительных узлов (см. ), 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.

На первом этапе SQL-запрос передается пользователем на выделенную host-машину, где транслируется в некоторый последовательный физический план [8].

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. Тест вычислительной производительности LINPACK

Методика измерения производительности LINPACK базируется на методе декомпозиции, который широко применяется при высокопроизводительных вычислениях [4]. Для реализации элементарных операций над векторами, которые включают умножение векторов на скаляр, сложение векторов, скалярное произведение векторов, выделяется базовый уровень системы, называемый BLAS (Basic Linear Algebra Subprograms). Исходные данные для тестирования представляются в виде вещественных чисел двойной точности. Полученные результаты выражаются в GFLOPS.

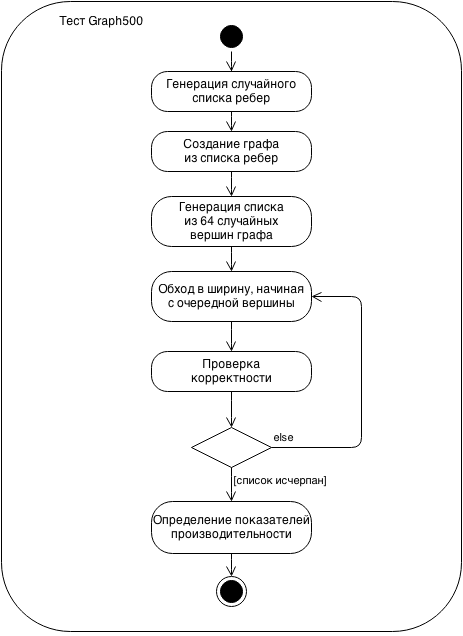
* 1. Тест оценки производительности систем баз данных TPC

Методика оценки производительности систем баз данных была разработана Советом по оценке производительности обработки транзакций (TPC – Transaction Processing Performance Council) [5]. Основной задачей этой организации является точное определение тестовых пакетов для оценки систем обработки транзакций и баз данных, а также для распространения объективных, проверяемых данных в промышленности. Под термином «*транзакция*» (transaction) понимается коммерческий обмен товарами, услугами или деньгами.

TPC публикует спецификации тестовых пакетов, которые регулируют вопросы, связанные с работой тестов. Эти спецификации гарантируют, что покупатели имеют объективные значения данных для сравнения производительности различных вычислительных систем. Хотя реализация спецификаций оценочных тестов оставлена на усмотрение индивидуальных спонсоров тестов, сами спонсоры, объявляя результаты TPC, должны представить TPC детальные отчеты, документирующие соответствие всем спецификациям. Эти отчеты, в частности, включают конфигурацию системы, методику калькуляции цены, диаграммы значений производительности и документацию, показывающую, что тест соответствует требованиям атомарности, согласованности, изолированности и долговечности (ACID - atomicity, consistency, isolation, and durability), которые гарантируют, что все транзакции из оценочного теста обрабатываются должным образом.

* 1. Тест Graph500

Целью данного теста является разработка приложения, состоящего из нескольких основных анализирующих частей (так называемых «*ядер*»), которые обращаются к единой структуре данных, представляющей собой взвешенный, ненаправленный граф [2]. Текущая спецификация теста содержит 2 ядра, одно ядро из входного списка ребер создает представление графа, которое будет использоваться в дальнейшем, а второе — производит вычисления на этом графе. Общий алгоритм теста Graph500 приведен на .



**Рис. 3.** Алгоритм теста Graph500

Тест Graph500 включает в себя масштабируемый генератор данных, который генерирует список ребер, где для каждого ребра указана начальная и конечная вершины. Согласно размеру входных данных задача разделяется на 6 классов, описанных в .

**Табл. 1.** Характеристики классов задач

|  |  |  |  |
| --- | --- | --- | --- |
| **Класс задачи** | **Количество вершин** | **Количество ребер** | **Объем памяти** |
| Игрушечный (уровень 10) | 226 | 230 | 17 ГБ |
| Минимальный (уровень 11) | 229 | 233 | 137 ГБ |
| Маленький  (уровень 12) | 232 | 236 | 1 ТБ |
| Средний  (уровень 13) | 236 | 240 | 17 ТБ |
| Большой  (уровень 14) | 239 | 243 | 140 ТБ |
| Огромный  (уровень 15) | 242 | 246 | 1 ПБ |

На основе указанных классов определяются два основных параметра генератора графа: *scale* и *edgefactor*. Параметр *scale* равен логарифму по основанию 2 от числа вершин графа и определяет число вершин и число ребер в сгенерированном графе. Параметр *edgefactor* равен отношению количества ребер графа к числу вершин и определяет количество ребер графа.

После генерации списка ребер, первое ядро программы переводит его во внутреннее представление, которое будет использоваться всеми последующими ядрами и которое не может быть изменено в ходе работы алгоритма.

Перед началом работы второго ядра в графе случайно выбирается 64 вершины со степенью не меньше 1 (не считая петли), чтобы исключить работу алгоритма на изолированных вершинах.

Для каждой из выбранных вершин второе ядро строит так называемое *BFS‑дерево*. BFS‑дерево получается в результате обхода графа в ширину (алгоритм BFS), причем для каждой вершины указывается ее родитель, то есть вершина, благодаря которой данная вершина была добавлена в результирующее BFS‑дерево.

После того, как BFS‑дерево построено, оно проходит процедуру проверки (валидации). Чтобы убедиться, что построенное BFS‑дерево корректно, оно должно удовлетворять следующим критериям:

1. Полученное BFS-дерево является деревом и не содержит циклов.
2. Каждое ребро BFS-дерева соединяет вершины, чей уровень при обходе в ширину различается ровно на 1.
3. Каждое ребро из входного списка ребер соединяет вершины, уровень которых в BFS-дереве различается не более чем на единицу или же обе эти вершины не включены в BFS-дерево (принадлежат другой компоненте связности).
4. В BFS-дерево входят все вершины данной компоненты связности.
5. Каждая вершина соединена ребром с ее родителем в исходном графе.

Для того чтобы сравнить производительность реализаций теста Graph500, написанных на различных аппаратных платформах, моделях разработки и языках программирования, вводится новая метрика для измерения производительности. Авторами теста вводится новая метрика, показывающая число пройденных ребер в секунду (TEPS). Показатель TEPS определяется во время работы второго ядра по следующей формуле:

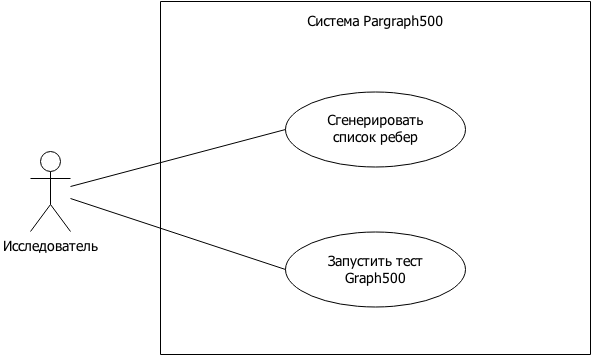
где – это количество входных ребер в пределах компоненты связности, на которой осуществлялся обход, включая петли и кратные ребра, а – это время выполнения второго ядра.

1. Алгоритмы, реализующие тест Graph500 для параллельной СУБД PargreSQL

В данном разделе приведено описание системы Pargraph500, разработанной для реализации теста Graph500 для параллельной СУБД PargreSQL. В первом разделе описан анализ требований к системе, во втором разделе рассмотрена модульная структура системы Pargraph500, в третьем разделе подробно описаны алгоритмы, позволяющие реализовать тест Graph500 с помощью параллельной СУБД PargreSQL в рамках разрабатываемой системы Pargraph500.

* 1. Разработка требований к системе Pargraph500

На представлена диаграмма вариантов использования системы Pargraph500. Единственным актером в системе является Исследователь, то есть человек, который исследует производительность теста Graph500 на какой-либо вычислительной системе. Система предоставляет Исследователю следующие варианты использования: сгенерировать список ребер и запустить тест Graph500. Генерация списка ребер подразумевает создание списка смежности вершин графа. Второй вариант использования системы предполагает запуск теста Graph500 на основе существующего списка ребер.

****

**Рис. 4.** Варианты использования

Важным требованием является совместимость системы Pargraph500 с семейством операционных систем Linux, так как более 90% мощнейших суперкомпьютеров по версии рейтинга TOP500 [3] созданы на базе операционных систем семейства Linux.

Еще одним требованием является совместимость системы с параллельной СУБД PargreSQL, поддерживающей фрагментный параллелизм.

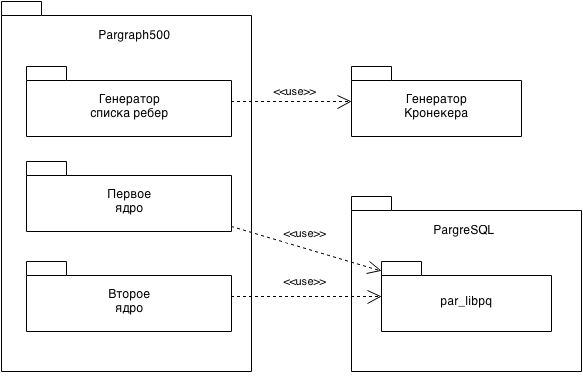
* 1. Модульная структура

На представлена модульная структура системы. Система Pargraph500 состоит из 3 частей:

* *Генератор списка ребер* – модуль, который генерирует список ребер графа.
* *Первое ядро* – модуль, который реализует первое ядро сравнительного теста Graph500, а именно генерацию внутреннего представления графа.
* *Второе ядро* – модуль, который реализует второе ядро сравнительного теста Graph500, а именно обход в ширину на графе.

Генератор списка ребер использует референсную реализацию генератора Кронекера [1].

Первое и второе ядро используют API PargreSQL, в том числе параллельную библиотеку parlib\_pq.



**Рис. 5.** Структура системы

* 1. Схема базы данных

На представлена схема данных.

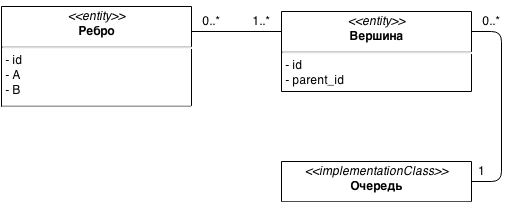
Сущность «Дуга» содержит следующие поля:

* id – номер дуги
* a\_id – номер первой вершины
* b\_id – номер второй вершины

Сущность «Вершина» содержит следующие поля:

* id – номер вершины
* parent\_id – номер родительской вершины при построении BFS‑дерева

Класс реализации «Очередь» содержит основные методы работы с очередью: добавить в очередь, извлечь из очереди, проверить очередь на пустоту.



**Рис. 6.** Схема данных

* 1. Описание таблицы на языке SQL

Спецификация теста Graph500 не предполагает изменение графа во время проведения теста: то есть в граф не добавляются новые вершины и дуги, и ни дуги, ни вершины не удаляются из графа.

В связи с этим предлагается хранить информацию о графе в виде реляционной таблицы, представляющей собой список дуг с некоторыми вспомогательными полями.

create table graph (

id bigint, -- порядковый номер дуги

a\_id bigint, -- номер первой вершины

b\_id bigint, -- номер второй вершины

parent bigint, -- номер родительской вершины

queue bigint -- номер в очереди

) with (fragattr = id); -- функция фрагментации

**Рис. 7.** Создание таблицы

На представлен запрос, создающий таблицу, в которой хранится внутреннее представление графа. Это представление будет использоваться всеми следующими ядрами. Таблица имеет следующие поля:

* *id* – порядковый номер дуги *(a\_id, b\_id)*. Выражается целым 64‑разрядным числом, номер каждой дуги уникален.
* *a\_id* – номер начальной вершины дуги с номером *id*. Выражается целым 64‑разрядным числом.
* *b\_id* – номер конечной вершины дуги с номером *id*. Выражается целым 64‑разрядным числом.
* *parent* – номер вершины, являющейся родительской для вершины *a\_id*. Выражается целым 64‑разрядным числом. Это поле используется во время работы второго ядра.
* *queue* – порядковый номер в очереди вершины *a\_id*. Выражается целым 64‑разрядным числом. Это поле используется во время работы второго ядра.

При этом существует функциональная зависимость , при которой значения полей *parent* и *queue* дублируются для всех одинаковых значений *a\_id*.

Так же подобная денормализация необходима, чтобы уменьшить количество хранимой информации и уменьшить накладные расходы, связанные с операцией соединения.

После ключевого слова with, расширяющего возможности создания таблиц в СУБД PargreSQL, идет ключевое слово fragattr и название поля, по которому будет осуществляться фрагментация. В данном случае фрагментация будет осуществлена по полю id.

* 1. Генерация списка ребер

Для генерации списка ребер используется рефернесная реализация *генератора Кронекера*, который представляет собой генератор списка ребер на основе алгоритма Кронекера, рекурсивного алгоритма генерации матриц R‑MAT [].

Генератор принимает на вход два параметра: *scale* и *edgefactor*. Параметр *scale* равен логарифму по основанию 2 от числа вершин графа и определяет число вершин и число ребер в генерируемом графе. Параметр *edgefactor* равен отношению количества ребер графа к числу вершин и определяет количество ребер графа.

Результатом работы генератора является файл, содержащий список ребер сгенерированного графа. Каждая строка списка представляет собой набор из двух целых положительных чисел, разделенных пробелом (номеров вершин).

* 1. Генерация внутреннего представления графа

Созданный на предыдущем шаге список ребер переводится в представление, которое будет использоваться при дальнейшей работе алгоритма и которое не может быть изменено во время работы алгоритма. Текст программы на языке C приведен на . Из файла последовательно читаются строки и каждое ребро вставляется в таблицу SQL с помощью запроса *insert*. Так как в SQL‑таблице хранятся дуги, каждое ребро должно быть добавлено в виде двух дуг – из первой вершины во вторую и из второй вершины в первую. Каждому ребру присваивается номер, на основе которого параллельная СУБД PargreSQL будет осуществлять фрагментацию данных на узлах.

#include <parlibpq-fe.h>

#include <stdio.h>

int main(int argc, char const \*argv[]) {

PGconn \*conn;

PGresult \*res;

long int a,b,i;

char query[200];

char \*edgeList; // файл, содержащий список ребер

if (argc != 2) {

puts("Please specify file name!");

exit(1);

}

edgeList = argv[1];

FILE \*in=freopen(edgeList,"r",stdin);

conn = PQconnectdb("");

if (PQstatus(conn) == CONNECTION\_BAD) {

puts(PQerrorMessage(conn));

puts("We were unable to connect to the database");

exit(0);

}

PQexec(conn,"drop table if exists graph");

PQexec(conn,"create table graph (id bigint,a\_id bigint, b\_id bigint,

parent bigint, queue bigint) with (fragattr = id);");

i = 0;

while (scanf("%ld %ld",&a,&b) != EOF ) {

sprintf(query,"insert into graph(id,a\_id,b\_id,parent,queue)

values(%d,%d,%d,NULL,NULL)",i,a,b);

PQexec(conn,query);

++i;

sprintf(query,"insert into graph(id,a\_id,b\_id,parent,queue)

values(%ld,%ld,%ld,NULL,NULL)",i,b,a);

PQexec(conn,query);

++i;

}

PQclear(res);

PQfinish(conn);

return 0;

}

**Рис. 8.** Генерация внутреннего представления графа

* 1. Алгоритм обхода графа в ширину

Обход в ширину является стандартным алгоритмом, реализуемым на графе [7]. На приведен алгоритм на C-подобном языке, выполняющий обход в ширину на графе, заданном с помощью списка смежности.

void BFT(NODE \*startNode, NODE \*parents) {

NODE currentNode;

Queue <NODE> queue;

foreach node in nodes do {

parents[node] = NO\_PARENT;

}

queue.push(startNode);

do {

if (queue.isEmpty())

break;

currentNode = queue.front();

foreach nextNode.adjacent(currentNode) do {

if (parent[nextNode] == NO\_PARENT) {

queue.push(nextNode);

parent[nextNode] = currentNode;

}

}

queue.pop();

}

while (true);

}

**Рис. 9.** Алгоритм обхода в ширину

Алгоритм начинается прохода по массиву родительских вершин с присвоением каждой из них специального значения, обозначающего, что вершина не имеет родителя. На этом шаге BFS‑дерево пусто. Далее в очередь добавляется *корневая* вершина, с которой начинается построение дерева. На следующем шаге для каждой вершины в очереди, пока очередь не пуста, в очередь добавляются все вершины, связанные с текущей, которые не были добавлены ранее. После выполнения алгоритма обхода массив *parent* содержит информацию о родителе каждой вершины.

Для реализации данного алгоритма на языке SQL необходимо заменить все операции, использующие очередь, на соответствующие обращения к базе данных.

#include <parlibpq-fe.h>

#include <stdio.h>

#include "graph500.h"

int main (int argc, char const \*argv[]) {

PGconn \*conn;

PGresult \*res,\*res\_parent; // результат выполнения запроса

long res\_count; // количество записей в полученном ответе

long i; // счетчик цикла

char currentNode[20], // текущий обрабатываемый узел

nextNode[20], // очередной потомок текущего узла

startNode[20],goalNode[20];

char query[200];

long nextQueueNumber = 0, // номер в очереди следующего узла

newQueueNumber = 0;// номер нового добавляемого в очередь узла

conn = PQconnectdb("");

if (PQstatus(conn) == CONNECTION\_BAD) {

puts(PQerrorMessage(conn));

puts("We were unable to connect to the database");

exit(0);

}

if (argc != 2) {

puts("Please specify startNode!");

exit(1);

}

strcpy(startNode,argv[1]);

SET\_ALL\_NO\_PARENT(query);

QUEUE\_PUSH(query,newQueueNumber,startNode,startNode);

do {

QUEUE\_SIZE(query,res,res\_count,nextQueueNumber);

if (QUEUE\_IS\_EMPTY(res\_count)) {

// очередь пуста

break;

}

// очередь не пуста

QUEUE\_FRONT(res,currentNode);

for (i=0; i < res\_count; ++i) {

strcpy(nextNode,PQgetvalue(res, i, 1));

GET\_NODE\_PARENT(query,res\_parent,nextNode);

if (NODE\_HAS\_NO\_PARENT(res\_parent)) { // нет родителя

++newQueueNumber; // push

QUEUE\_PUSH(query,newQueueNumber,

currentNode,nextNode);

}

}

QUEUE\_POP(nextQueueNumber);

} while (true);

PQclear(res);

PQfinish(conn);

return 0;

}

**Рис. 10.** Программа, осуществляющая поиск в ширину

Программа, реализующая алгоритм поиска в ширину на языке C с использованием API PargreSQL, приведена на . В тексте программы все SQL‑запросы к БД заменены на макросы, которые определены в файле *graph500.h*. Имена этих макросов имеют названия, схожие с основными шагами, выполняемыми при реализации поиска в ширину без использования языка SQL. Подобная замена позволяет проследить связь между функциями в языке и запросами SQL (см. ).

**Табл. 2.** Связь между SQL‑запросами и макросами

|  |  |  |
| --- | --- | --- |
| **Макрос** | **Запрос** | **Семантика** |
| SET\_ALL\_NO\_PARENT | update graph  set parent=-1,queue=NULL | Сбросить значения полей parent и queue |
| QUEUE\_PUSH | update graph  set parent=currentNode, queue=newQueueNumber  where parent=-1  and a\_id=nextNode | Добавить в очередь вершину nextNode, связанную с currentNode, если она не была добавлена ранее |
| QUEUE\_SIZE | select distinct a\_id,b\_id from graph where queue=nextQueueNumber | Узнать количество элементов в очереди |
| QUEUE\_IS\_EMPTY | Используется результат QUEUE\_SIZE | Проверить, пуста ли очередь |
| QUEUE\_FRONT | Используется результат QUEUE\_SIZE | Получить первый элемент из очереди |
| GET\_NODE\_PARENT | select distinct parent  from graph  where a\_id=nextNode | Получить значение поля parent вершины nextNode, чтобы узнать, были ли посещена вершина ранее |
| NODE\_HAS\_NO\_PARENT | Используется результат GET\_NODE\_PARENT | Вершина не была посещена ранее |
| QUEUE\_POP | Запрос не производится | Перейти к следующему элементу в очереди |

1. Вычислительные эксперименты

В данном разделе описана аппаратная платформа, на которой проводились эксперименты, реализующие тест Grap500, приведен план экспериментов и описаны основные достигнутые результаты.

* 1. Аппаратная платформа экспериментов

Для исследования эффективности разработанной реализации теста Graph500 была проведена серия экспериментов. В качестве аппаратной платформы был выбран суперкомпьютер «Торнадо ЮУРГУ» [10], технические характеристики которого приведены в .

**Табл. 3.** Характеристики суперкомпьютера «Торнадо ЮУрГУ»

|  |  |
| --- | --- |
| Число вычислительных узлов | 480 |
| Тип процессора | Intel Xeon X5680 (Gulftown, 6 ядер по 3.33 GHz) — 960 шт. |
| Тип coпроцессора | Intel Xeon Phi SE10X (61 ядро по 1.1 GHz) — 384 шт. |
| Оперативная память | 16.9 TB |
| Дисковая память | 150 TB, твердотельные накопители SSD Intel, параллельная система хранения данных Panasas ActiveStor 11 |
| Тип системной сети | InfiniBand QDR (40 Gbit/s) |
| Тип управляющей сети | Gigabit Ethernet |
| Пиковая производительность комплекса | 473.6 TFlops |
| Производительность на тесте LINPACK | 288.2 TFlops |
| Операционная система | Linux CentOS 6.2 |

* 1. План экспериментов

Провести эксперименты на тестовом графе из 16 вершин на 1 узле с использованием последовательной СУБД PostgreSQL.

Провести эксперименты на тестовом графе из 16 вершин на 4 узлах с использованием параллельной СУБД PargreSQL.

Провести эксперименты на полноразмерном графе, соответствующем классу сложности «Игрушечный» на 64 узлах с использованием параллельной СУБД PargreSQL.

* 1. Результаты

Программа работает на небольшом объеме данных.

А на большом не работает.

Заключение

Работа посвящена оценке эффективности параллельной СУБД PargreSQL на задачах интенсивной обработки данных с помощью сравнительного теста Graph500. В ходе работы получены следующие основные результаты:

1. Изучена архитектура параллельной СУБД PargreSQL и спецификация теста Graph500.
2. Разработана схема базы данных для хранения графа и промежуточных данных в соответствии со спецификацией теста Graph500.
3. Выполнено проектирование и разработка алгоритмов на языке SQL, реализующих тест Graph500 для параллельной СУБД PargreSQL.
4. Проведены вычислительные эксперименты на суперкомпьютере «Торнадо ЮУрГУ», исследующие эффективность параллельной СУБД PargreSQL на тесте Graph500.

По результатам данной работы был выполнен доклад на научной конференции:

* Сафонов А. Ю. Реализация теста Graph500 для реляционной СУБД на основе фрагментного параллелизма. 67 студенческая научная конференция ЮУрГУ (Челябинск, 23 апреля 2014 г.).

Литература

1. Chakrabarti D., Zhan Y., Faloutsos C. R-MAT: A recursive model for graph mining // SDM 2004. P. 442–446.
2. Graph500 Benchmark [Электронный ресурс] // Graph500 [сайт]. [2014]. URL: http://www.graph500.org/specifications (дата обращения: 14.03.2014)
3. Graph500 Benchmark [Электронный ресурс] // Graph500 [сайт]. [2014]. URL: http://www.graph500.org/specifications (дата обращения: 14.03.2014)
4. LINPACK [Электронный ресурс] // LINPACK [сайт]. [2014]. URL: http://www.netlib.org/linpack/ (дата обращения 15.03.2014).
5. Transaction Processing Performance Council [Электронный ресурс] // TPC [сайт]. [2014]. URL: http://www.tpc.org/ (дата обращения 21.04.2014).
6. 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.]. -Екатеринбург: УрО РАН, 2006. С. 42-83.
7. Мокрозуб В.Г. Графовые структуры и реляционные базы данных в автоматизированных интеллектуальных информационных системах. – М.: Издательский дом "Спектр", 2011. 108 c.
8. 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.
9. Соколинский Л.Б. Параллельные системы баз данных. М.: Издательство Московского университета, 2013. 184 с.
10. Суперкомпьютер «Торнадо ЮУрГУ» // Лаборатория суперкомпьютерного моделирования ЮУрГУ. 2014. URL: http://supercomputer.susu.ac.ru/computers/tornado/ (дата обращения: 14.03.2014)