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
к.п.н.

_____ А.Ю. Эвнин

“___”_____ 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 – 010400.62.2014.08-036-1570.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

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы ВМИ-456 Сафонову Александру Юрьевичу,
обучающемуся по направлению 010400.62 «Информационные технологии»

1. **Тема работы** (утверждена приказом ректора от 12.03.2014 № 368)

Реализация теста Graph500 для параллельной СУБД PargreSQL.

2. Срок сдачи студентом законченной работы: 04.06.2014.

3. Исходные данные к работе

3.1. 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.
3.2. Соколинский Л.Б. Параллельные системы баз данных. – М.: Издательство Московского университета, 2013. – 184 с.
3.3. Jose J. Designing Scalable Graph500 Benchmark with Hybrid MPI+OpenSHMEM Programming Models / J. Jose, S. Potluri, K. Tomko, D.K. Pandas // 28th International Supercomputing Conference (ISC 2013), Leipzig, Germany, 16.06 – 20.06, 2013. Germany: Proceedings, 2013. – P. 109–124.

4. Перечень подлежащих разработке вопросов

4.1. Изучить архитектуру параллельной СУБД PargreSQL и спецификацию теста Graph500.

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

5. Дата выдачи задания: 08.02.2014.

Научный руководитель

к.ф.-м.н., доцент М.Л. Цымблер

Задание принял к исполнению А.Ю. Сафонов

Оглавление

Введение

Актуальность исследования

Большинство существующих тестов производительности суперкомпьютеров направлены на измерение производительности вычислительных мощностей (тесты LINPACK [12], 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, который учитывает специфику задач с интенсивной обработкой данных. В 2010 г. 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 [11]. 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 [10].

Параллельная СУБД PargreSQL [4], разработанная на кафедре системного программирования факультета Вычислительной математики и информатики ЮУрГУ позволяет работать со сверхбольшими базами данных. Эта возможность достигается за счет использования фрагментного параллелизма[9].

Перспективным является использование реляционных СУБД для интеллектуального анализа графов [13], однако в настоящее время в этом направлении существует малое количество опубликованных работ. Напри-

мер, в работе [8] описан SQL-ориентированный алгоритм DB FSG, осуществляющий поиск часто встречающихся подграфов.

Таким образом, актуальной является задача оценки возможности использования параллельной СУБД PargreSQL для реализации теста Graph500.

Цель и задачи работы

Целью данной 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 29 страниц, объем библиографии – 15 источников.

Содержание работы

Первая глава, «Анализ предметной области», описывает предметную область, в рамках которой ведется данная работа.

Вторая глава, «Алгоритмы, реализующие тест 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. Анализ предметной области

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.

1.1. СУБД PargreSQL

Базисной концепцией параллельной обработки запросов в реляционных системах баз данных является фрагментный параллелизм [5]. Принципиальная схема обработки запроса с использованием фрагментного параллелизма выглядит следующим образом (см. рис. 1).

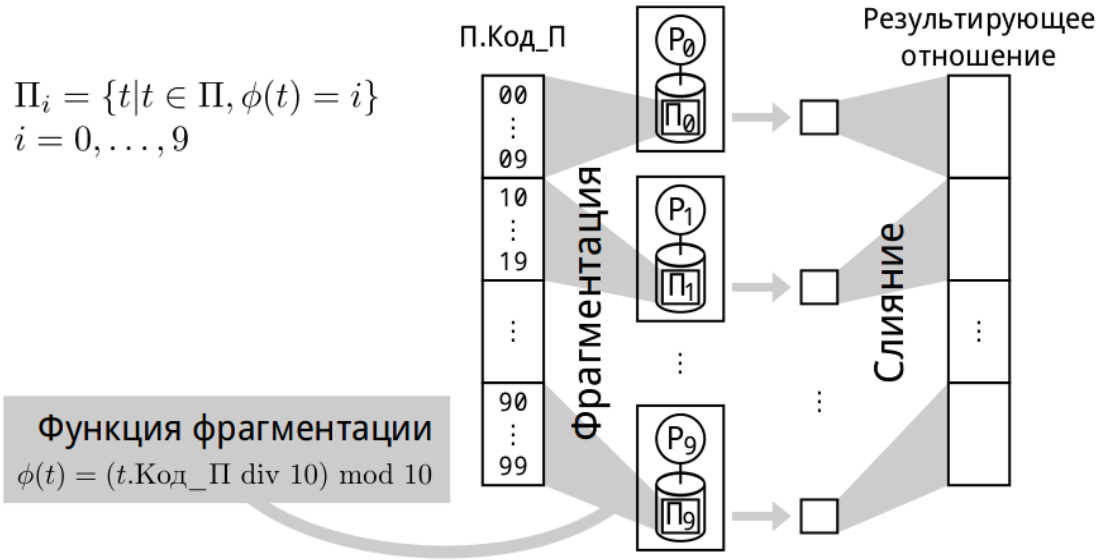

Рис. 1. Параллельная обработка запроса на основе фрагментного параллелизма

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

Функция фрагментации отношения R по формуле 1 вычисляет номер процессорного узла, на котором должен храниться кортеж.

$$\varphi: R \to \{0, 1, \dots, k-1\} \quad (1)$$

Величина k (количество процессорных узлов) называется степенью фрагментации. Для фрагментации целесообразно использовать атрибутную фрагментацию (формула 2).

$$\forall r \in R\big(\varphi_R(r) = f_A(r.A)\big), \quad (2)$$

где $f_A: D_A \to \{0, \dots, k-1\}$ – некоторая функция, определенная 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 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 за-

проса произвольное количество операторов *exchange* (для каждого оператора указывается свой уникальный порт обмена).

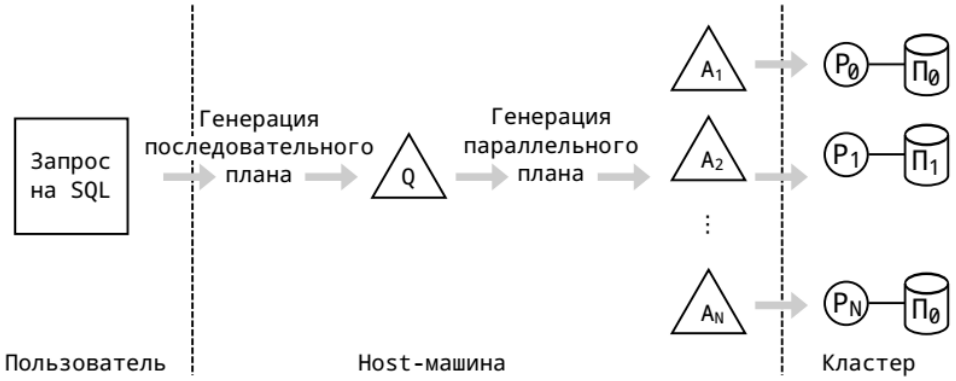

Рис. 2. Схема обработки запроса в параллельной СУБД:
Q – последовательный физический план, A_i – параллельный агент,
P_i – вычислительный узел, D_i – диск

Общая схема организации обработки запросов в параллельной СУБД выглядит следующим образом [1]. Мы будем полагать, что вычислительная система представляет собой кластер из N вычислительных узлов (см. рис. 2), 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 [3].

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 корне-

вым оператором *exchange* на нулевом узле, откуда передаются на host-машину.

Роль host-машины может играть любой узел вычислительного кластера.

1.2. Тест вычислительной производительности LINPACK

Тест состоит в решении системы линейных уравнений с помощью LU-факторизации. Основное время затрачивается на векторные операции типа FMA (умножение и сложение). Методика измерения производительности LINPACK базируется на методе декомпозиции, который широко применяется при высокопроизводительных вычислениях [12]. 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. Производительность определяется как количество вычислительных операций над этими числами в расчете на 1 секунду, и выражается в GFLOPS (миллиардах операций с плавающей точкой в секунду) [14].

1.3. 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) [15]. 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.

1.4. 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 [10]. 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 [11]. Общий алгоритм теста Graph500 приведен на рис. 3.

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.

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 ко-

личества ребер графа к числу вершин и определяет количество ребер графа.

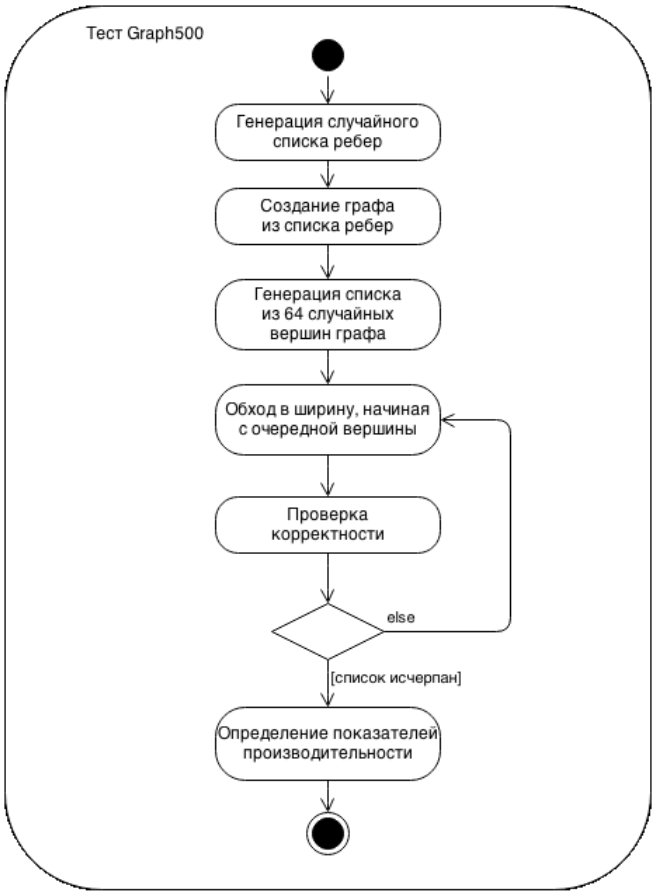

Рис. 3. Алгоритм теста Graph500

Табл. 1. Характеристики классов задач

Класс задачи	**Количество вершин**	**Количество ребер**	**Объем памяти**
Игрушечный (уровень 10)	2^{26}	2^{30}	17 ГБ
Минимальный (уровень 11)	2^{29}	2^{33}	137 ГБ
Маленький (уровень 12)	2^{32}	2^{36}	1 ТБ
Средний (уровень 13)	2^{36}	2^{40}	17 ТБ
Большой (уровень 14)	2^{39}	2^{43}	140 ТБ
Огромный (уровень 15)	2^{42}	2^{46}	1 ПБ

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 – 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, пока-

зывающая число пройденных ребер в секунду (TEPS). Показатель TEPS определяется во время работы второго ядра по формуле 3:

$$TEPS = \frac{m}{time_{K2}}, \quad (3)$$

где m – это количество входных ребер в пределах компоненты связности, на которой осуществлялся обход, включая петли и кратные ребра, а $time_{K2}$ – это время выполнения второго ядра.

Создателями теста Graph500 специфицированы параметры, которые должны быть использованы при непосредственной реализации алгоритма теста, однако, существует ряд реализаций, которые удовлетворяют всем предъявляемым к реализации требованиям и одобрены разработчиками теста Graph500. Такие реализации называются *референсными реализациями* и доступны на сайте спецификации теста [10].

2. 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.

2.1. Разработка требований к системе Pargraph500

На рис. 4 представлена диаграмма вариантов использования системы Pargraph500. Единственным актером в системе является *Исследователь*, который выполняет тест Graph500 на какой-либо вычислительной системе. Система предоставляет Исследователю следующие варианты использования: генерировать граф и запустить тест Graph500. *Генерация графа* подразумевает создание списка ребер графа. Второй вариант использования системы предполагает *запуск теста Graph500* на основе существующего списка ребер.

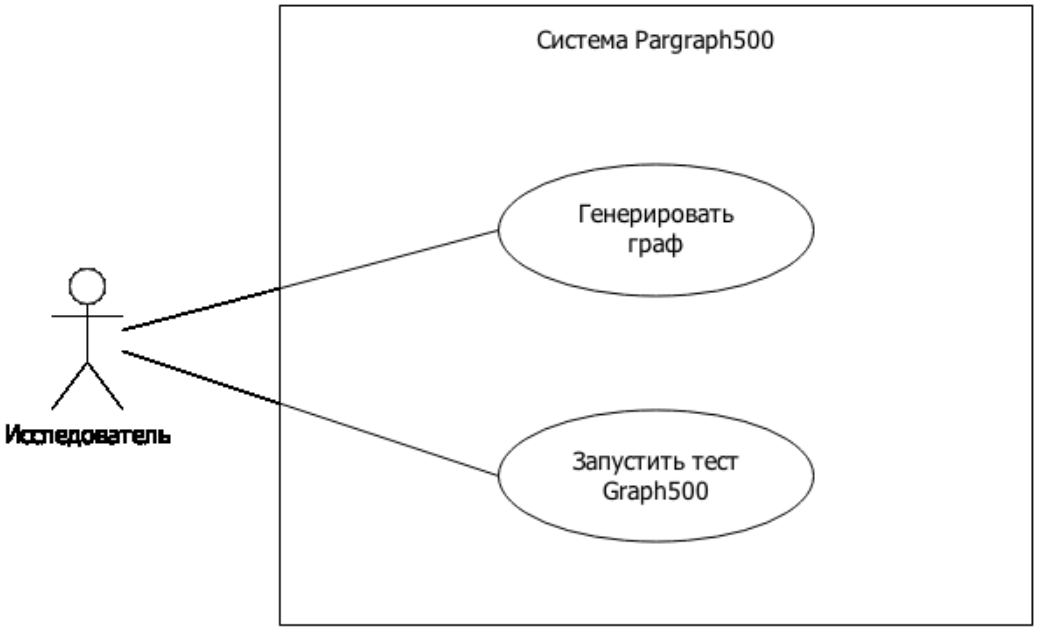

Рис. 4. Варианты использования

К системе Pargraph500 предъявляется 2 функциональных требования:

– система должна быть совместима с семейством операционных систем Linux, так как тестирование и отладку системы планируется производить на суперкомпьютере «Торнадо ЮУрГУ», на котором установлена ОС Linux CentOS 6.2;

– система должна быть совместима с параллельной СУБД PargreSQL, поддерживающей фрагментный параллелизм, исходя из условий поставленной задачи.

2.2. Модульная структура

На рис. 5 представлена модульная структура системы. Система Pargraph500 состоит из 3 частей:

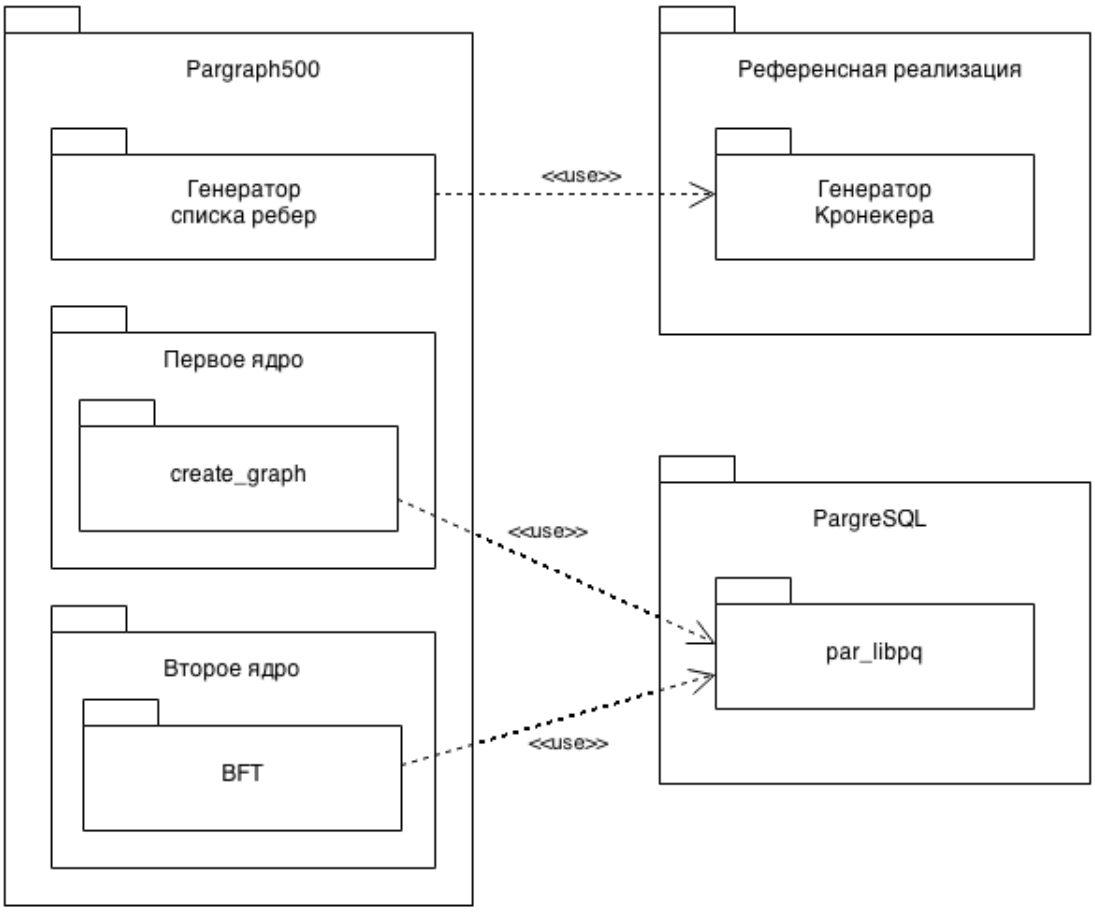

Рис. 5. Структура системы

– *генератор списка ребер* – модуль, который на основе входных дан-

ных, 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 [7].

Первое и второе ядро используют API PargreSQL, в том числе параллельную библиотеку parlib_pq.

2.3. Схема классов

Предлагается следующая схема классов предметной области, которая представленна на рис. 6. Схема классов состоит из двух сущностей (*Ребро* и *Вершина*) и класса реализации *Очередь*.

Сущность «Вершина» имеет семантику вершины графа. Так как предметная область предполагает присвоение отдельного номера каждой вершине, сущность «Вершина» обладает атрибутом *Number*, обозначающим порядковый номер вершины.

Сущность «Ребро» имеет семантику ребра графа и содержит два атрибута: *A* – номер первой вершины и *B* – номер второй вершины.

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

Связь между сущностями «Ребро» и «Вершина» означает, что согласно предметной области, каждая вершина может быть инцидентна любому количеству ребер (в том числе, могут быть изолированные вершины) и каждое ребро может быть инцидентно ненулевому количеству ребер (граф может содержать петли).

Унарная связь сущности «Вершина» с собой означает, что у каждой вершины может быть ровно 1 или 0 предков в построенном BFS-дереве. Причем у каждой вершины может быть несколько потомков (в том числе ни одного).

Связь между классом «Очередь» и сущностью «Вершина» означает, что В очереди может содержаться любое количество вершин, но каждая вершина может быть включена в очередь не более одного раза.

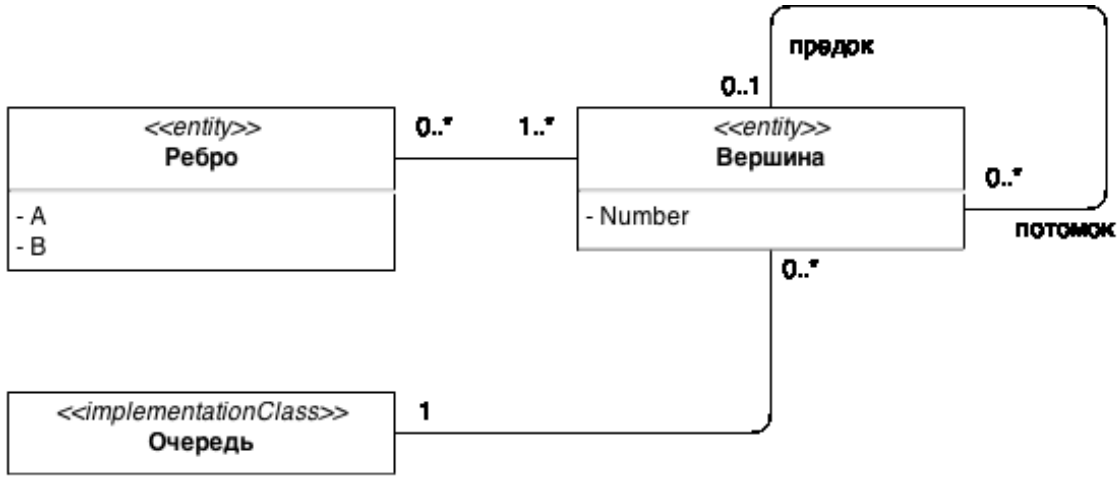

Рис. 6. Схема классов

2.4. Описание таблицы на языке SQL

Для хранения графа и промежуточных данных теста Graph500 была разработана следующая схема базы данных. База данных состоит из одной таблицы (см. рис. 7).

```
create table graph (
    id bigint,                -- порядковый номер дуги
    a_id bigint,              -- номер начальной вершины
    b_id bigint,              -- номер конечной вершины
    parent bigint,            -- номер вершины родителя a_id
    queue bigint              -- номер в очереди вершины a_id
) with (fragattr = id);       -- функция фрагментации
```

Рис. 7. Создание таблицы

На рис. 7 представлен запрос, создающий таблицу, в которой хранится внутреннее представление графа. Это представление будет исполь-

зоваться всеми следующими ядрами. Таблица имеет следующую структуру:

– *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.

Данная структура была выбрана по следующим причинам:

– количество вершин и ребер постоянно;

– позволяет уменьшить количество хранимой информации;

– позволяет уменьшить накладные расходы на операцию соединения.

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

Графы, генерируемые для спецификации теста Graph500, содержат от 2^{26} вершин и 2^{30} ребер (см. табл. 1). Предложенное представление позволяет хранить только список ребер со вспомогательными полями и не выполнять на каждом шаге операцию соединения, следовательно, уменьшить накладные расходы.

Чтобы обеспечить фрагментацию данных по узлам, в запрос добавлена информация об атрибуте, по которому СУБД PargreSQL должен обес-

печить разбиение. В данном случае фрагментация будет осуществлена по полю id.

2.5. 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 [7].

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).

2.6. Первое ядро

Созданный на предыдущем шаге список ребер переводится в представление, которое будет использоваться при дальнейшей работе алгоритма и которое не может быть изменено во время его работы. Интерфейс функции перевода графа из списка ребер в базу данных приведен на рис. 8.

```
/**
Функция для создания внутреннего представления графа.
Входные параметры: указатель на установленное соединение с базой данной и
имя файла, из которого должен читаться список ребер.
Выходные параметры: количество добавленных ребер или -1 в случае ошибки.
**/
int create_graph (PGconn *conn, const char *filename);
```

Рис. 8. Генерация внутреннего представления графа

2.7. Второе ядро

Обход в ширину является стандартным алгоритмом, реализуемым на

графе [2]. На рис. 9 приведен алгоритм на С-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 с 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.

Интерфейс функции, реализующей алгоритм поиска в ширину на языке С с использованием API PargreSQL, приведен на рис. 10. 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.

```
/**
Функция, реализующая обход в ширину на графе.
Входные параметры: указатель на установленное соединение с базой данной и
имя узла, с которого осуществляется обход в ширину.
Выходные параметры: информация об успешности выполнения обхода. 0, если об-
ход осуществлен успешно или -1 в случае ошибки.
**/
int BFT (PGconn *conn, const char *startNode);
```

Рис. 10. Программа, осуществляющая поиск в ширину

Подобная замена позволяет проследить связь между функциями в языке и запросами SQL. В табл. 2 приведены названия макросов, и для каждого из них обозначена семантика, которой он обладает в рамках алгоритма обхода в ширину.

Табл. 2. Семантика макросов в тексте программы

Макрос	Семантика
SET_ALL_NO_PARENT	Сбросить значения полей parent и queue
QUEUE_PUSH	Добавить в очередь вершину, связанную с текущей
QUEUE_SIZE	Узнать количество элементов в очереди
QUEUE_IS_EMPTY	Проверить, пуста ли очередь
QUEUE_FRONT	Получить первый элемент из очереди
GET_NODE_PARENT	Получить значение поля parent нужной вершины
QUEUE_POP	Перейти к следующему элементу в очереди

3. Вычислительные эксперименты

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

3.1. Аппаратная платформа экспериментов

Для исследования эффективности разработанной реализации теста Graph500 была проведена серия экспериментов. В качестве аппаратной платформы был выбран суперкомпьютер «Торнадо ЮУРГУ» [6], технические характеристики которого приведены в табл. 3.

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 сопроцессора	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

3.2. План экспериментов

Для оценки эффективности работы второго ядра алгоритма теста Graph500 было запланировано провести серию вычислительных экспериментов на графах с различным числом дуг. Кроме того, эксперименты проводились на суперкомпьютере «Торнадо ЮУрГУ» с использованием различного числа узлов.

Тесты проводились на графах двух размеров:

– 64 вершины, 2048 дуг;

– 256 вершин, 8192 дуги.

Прежде всего, была измерена производительность последовательной реализации алгоритма теста Graph500 при работе на одном узле с использованием последовательной СУБД PostgreSQL.

Затем была измерена производительность параллельной реализации алгоритма теста с использованием параллельной СУБД PargreSQL на 2 и на 4 узлах.

3.3. Результаты

На рис. 11 представлен график, отражающий зависимость количества пройденных дуг в секунду от количества узлов. Производительность измеряется в тысячах пройденных дуг в секунду (KTEPS).

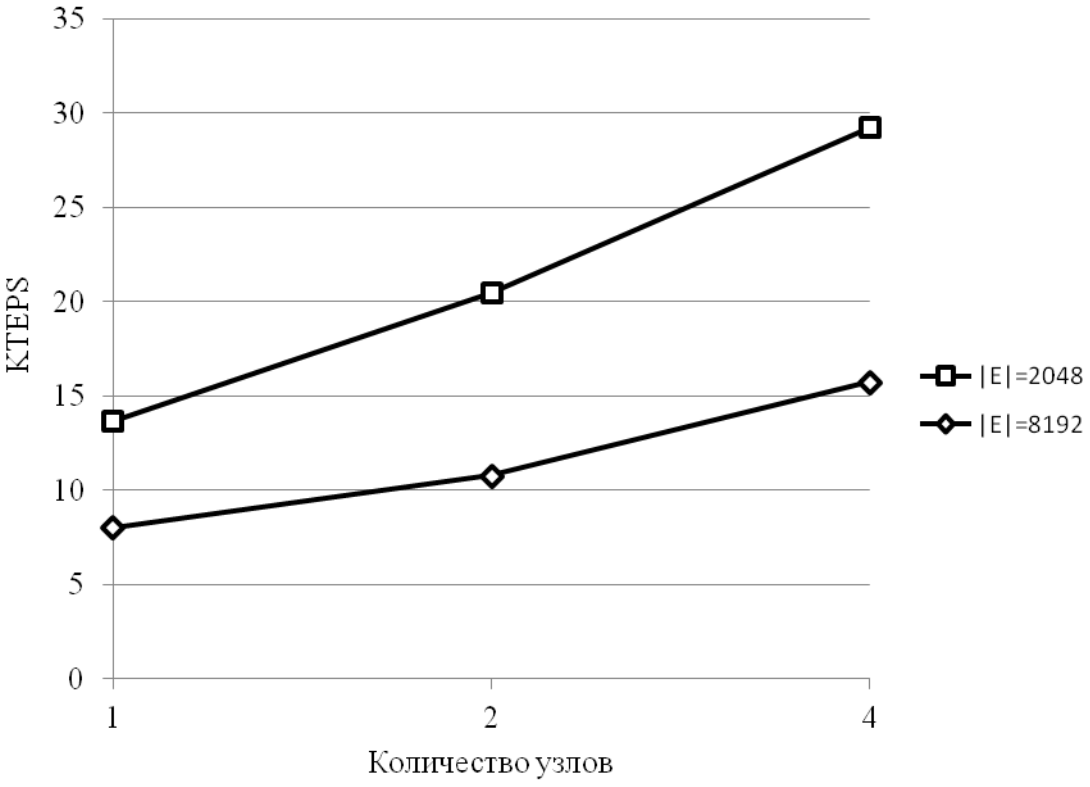

Рис. 11. Зависимость числа пройденных дуг в секунду от количества узлов

На рис. 12 показано ускорение работы алгоритма в зависимости от

количества узлов и размера входных данных.

Как можно заметить, разработанная реализация алгоритма не дает существенного ускорения при работе на 2 или 4 узлах.

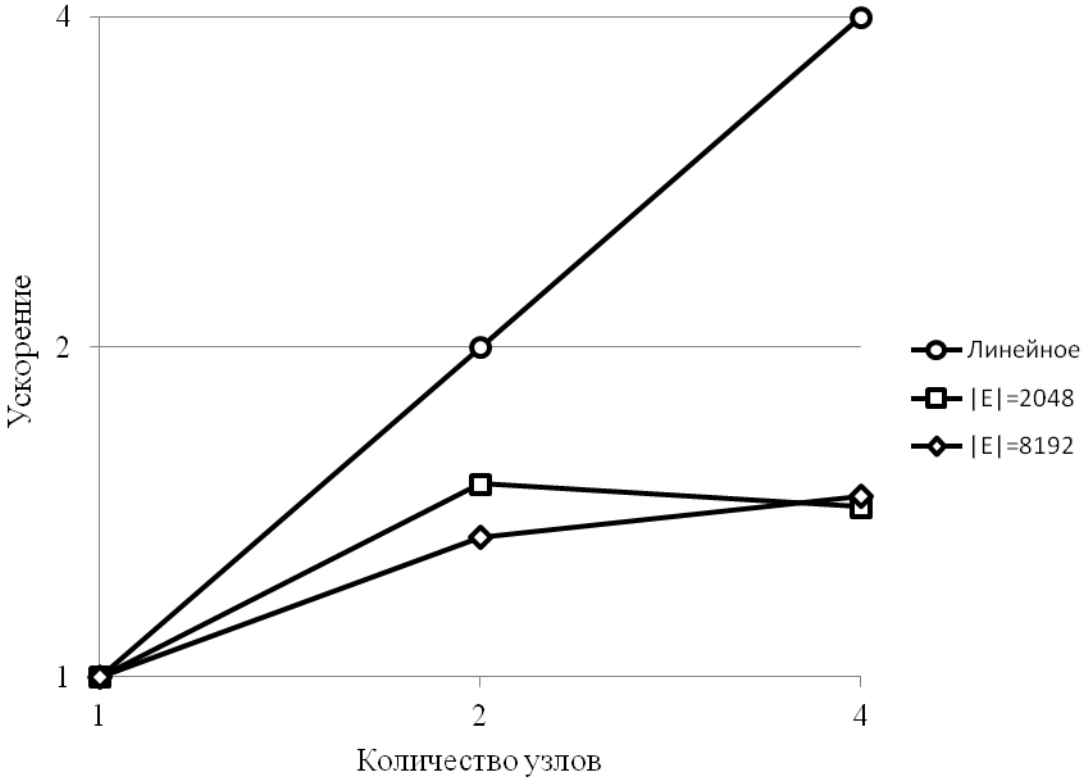

Рис. 12. Ускорение в зависимости от количества узлов

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. 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.

2. Мокрозуб В.Г. Графовые структуры и реляционные базы данных в автоматизированных интеллектуальных информационных системах. – М.: Издательский дом "Спектр", 2011. – 108 с.

3. 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.

4. 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.

5. Соколинский Л.Б. Параллельные системы баз данных. М.: Издательство Московского университета, 2013. 184 с.

6. Суперкомпьютер «Торнадо ЮУрГУ». [Электронный ресурс] URL: http://supercomputer.susu.ac.ru/computers/tornado/ (дата обращения: 14.03.2014)

7. Chakrabarti D., Zhan Y., Faloutsos C. R-MAT: A recursive model for graph mining // SDM 2004. P. 442–446.

8. 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 2008, Turin, Italy, September 1-5, 2008. Italy: Springer, 2008. – P. 684–692.

9. Gounaris A., Sakellariou R., Paton N., Fernandes A. A Novel Approach to Resource Scheduling for Parallel Query Processing on Computational Grids // Distributed and Parallel Databases, 2006, Hingham, USA, 2006. USA: Kluwer Academic Publishers, 2006. – Vol. 19, No. 2. – P. 87–106.

10. Graph500 Benchmark. [Электронный ресурс] URL: http://www.graph500.org/specifications (дата обращения: 14.03.2014)

11. Jose J. Designing Scalable Graph500 Benchmark with Hybrid MPI+OpenSHMEM Programming Models / J. Jose, S. Potluri, K. Tomko, D.K. Pandas // 28th International Supercomputing Conference (ISC 2013), Leipzig, Germany, 16.06 – 20.06, 2013. Germany: Proceedings, 2013. – P. 109–124.

12. LINPACK. [Электронный ресурс] URL: http://www.netlib.org/linpack/ (дата обращения 15.03.2014).

13. Srihari S., Chandrashekar S., Parthasarathy S. A Framework for SQL-based Mining of Large Graphs on Relational Databases // Advances in knowledge discovery and data mining. Lecture Notes in Computer Science. 2010. Vol. 6119. P. 160–167.

14. Tomić D., Gjenero L., Imamagić E. Semidefinite optimization of High Performance Linpack on Heterogeneous Cluster // The 36th international convention on information and communication technology, electronics and microelectronics, Opatija, Adriatic Coast, Croatia, May 20-24, 2013. – IEEE, 2013. – P. 177–182.

15. Transaction Processing Performance Council. [Электронный ресурс] URL: http://www.tpc.org/ (дата обращения 21.04.2014).