ПРОГРАММИРОВАНИЕ, 2017, № 3, с. 3–21

УДК 004.657

МЕТОДЫ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ СВЕРХБОЛЬШИХ БАЗ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ РАСПРЕДЕЛЕННЫХ КОЛОНОЧНЫХ ИНДЕКСОВ *

© 2017 г. Е.В. Иванова, Л.Б. Соколинский,
Южно-Уральский государственный университет
454080 Челябинск, пр. им. В. И. Ленина, д. 76
E-mail: Elena.Ivanova@susu.ru, Leonid.Sokolinsky@susu.ru
Поступила в редакцию 20.04.2016

Статья посвящена вопросам разработки и исследования эффективных методов параллельной обработки сверхбольших баз данных с использованием колоночного представления информации, ориентированных на кластерные вычислительные системы. Предлагается подход, позволяющий сочетать преимущества реляционных и колоночных СУБД. Вводится новый вид распределенных колоночных индексов, фрагментируемых на основе доменно-интервального принципа. Колоночные индексы являются вспомогательными структурами, постоянно хранимыми в распределенной оперативной памяти кластерной вычислительной системы. Для сопоставления элементов колоночного индекса кортежам исходного отношения используются суррогатные ключи. Ресурсоемкие реляционные операции выполняются не над исходными отношениями базы данных, а над соответствующими колоночными индексами. В результате получается таблица предварительных вычислений. С использованием этой таблицы СУБД осуществляет реконструкцию результирующего отношения. Для всех основных реляционных операций над распределенными колоночными индексами предложены методы их параллельной декомпозиции, не требующие массовых обменов данных между процессорными узлами. Указанный подход позволяет ускорить выполнение запросов класса OLAP в сотни раз.

1. ВВЕДЕНИЕ

В настоящее время одним из феноменов, оказывающих существенное влияние на область технологий обработки данных, являются сверхбольшие данные. Согласно прогнозам аналитической компании IDC, количество данных в мире удваивается каждые два года и к 2020 г. достигнет 44 Зеттабайт, или 44 триллионов гигабайт [1]. Сверхбольшие данные путем очистки и структурирования преобразуются в сверхбольшие базы и хранилища данных. По оценкам корпорации Microsoft 62% крупных американских компаний имеют хранилища данных, объем которых превышает 100 Терабайт [2]. При этом современные технологии баз данных не могут обеспечить обработку столь крупных объемов данных. По оценке IDC в 2013 г. из всего объема существующих данных потенциально полезны 22%, из которых менее 5% были подвергнуты анализу. К 2020 году процент потенциально полезных данных может вырасти до 35%, преимущественно за счет данных от встроенных систем [1].

По мнению одного из ведущих специалистов мира в области баз данных М. Стоунбрейкера (Michael Stonebraker) для решения проблемы обработки сверхбольших данных необходимо использовать технологии СУБД [3]. СУБД предлагает целый спектр полезных сервисов, не предоставляемых файловой системой, включая схему (для управления семантикой данных), язык запросов (для организации до-

*Работа выполнена при финансовой поддержке Министерства образования и науки РФ (государственное задание 2.7905.2017) и Правительства РФ в соответствии с Постановлением № 211 от 16.03.2013 г. (соглашение № 02.A03.21.0011).

ступа к частям базы данных), сложные системы управления доступом (грануляция данных), сервисы обеспечения согласованности данных (управление целостностью данных и механизм транзакций), сжатие (для уменьшения размера базы данных) и индексирование (для ускорения обработки запросов).

Для обработки больших данных необходимы высокопроизводительные вычислительные системы [4, 5]. В этом сегменте вычислительной техники сегодня доминируют системы с кластерной архитектурой, узлы которых оснащены многоядерными ускорителями. Кластерные вычислительные системы занимают 85% списка TOP500 самых мощных суперкомпьютеров мира [6] (ноябрь 2015 г.). При этом в первой сотне списка более 50% систем оснащены многоядерными ускорителями. Самый мощный суперкомпьютер мира Tianhe-2 (Национальный суперкомпьютерный центр в Гуанчжоу, КНР) также имеет кластерную архитектуру и оснащен многоядерными ускорителями (сопроцессорами) Intel Xeon Phi. Его производительность составляет 33.9 PFLOP/S на тесте LINPACK, суммарный объем оперативной памяти — 1 петабайт. Недавние исследования показывают, что кластерные вычислительные системы могут эффективно использоваться для хранения и обработки сверхбольших баз данных [7, 8, 9, 10]. В разработке технологий параллельных систем баз данных [11, 12] к настоящему времени достигнут большой прогресс. Некоторый обзор научных публикаций по этой теме можно найти в [5, 13]. Однако в этой области остается целый ряд нерешенных масштабных научных задач, в первую очередь связанных с проблемой больших данных [3].

Оперативная память в качестве основного места хранения данных становится все более привлекательной в результате экспоненциального снижения отношения стоимость/размер [14, 15, 16]. Согласно отчету компании Gartner, СУБД в оперативной памяти через 2–5 лет получат очень широкое распространение [17].

Одним из наиболее важных классов приложений, связанным с обработкой сверхбольших баз данных, являются хранилища данных [18, 19, 20, 21], для которых характерны запросы типа OLAP. Исследования показали [22, 23, 24], что для таких приложений выгодно использовать колоночную модель представление данных, позволяющую получить на порядок лучшую производительность по сравнению с традиционными системами баз данных, использующими строчную модель представления данных. Эта разница в производительности объясняется тем, что колоночные хранилища позволяют выполнять меньше обменов с дисками при выполнении запросов на выборку данных, поскольку с диска (или из основной памяти) считываются значения только тех атрибутов, которые упоминаются в запросе [25]. Дополнительным преимуществом колоночного представления является возможность использования эффективных алгоритмов сжатия данных, поскольку в одной колонке таблицы содержатся данные одного типа. Сжатие может привести к повышению производительности на порядок, поскольку меньше времени занимают операции ввода-вывода [26]. Недостатком колоночной модели представления данных является то, что в колоночных СУБД затруднено применение техники эффективной оптимизации SQL-запросов, хорошо зарекомендовавшей себя в реляционных СУБД. Кроме этого, колоночные СУБД значительно уступают строковым по производительности на запросах класса OLTP.

В соответствии с выше сказанным, актуальной является задача разработки новых эффективных методов параллельной обработки сверхбольших баз данных в оперативной памяти на кластерных вычислительных системах, оснащенных многоядерными ускорителями, которые позволили бы совместить преимущества реляционной модели с колоночным представлением данных.

Статья организована следующим образом. В разделе 2 дается обзор работ по колоночной модели хранения данных и анализируются публикации, наиболее близко относящихся к теме настоящей статьи. Раздел 3 посвящен формальному описанию доменно-колоночной модели представления данных. На основе этой модели в разделе 4 предлагается декомпозиция основных реляционных операций над распределенными колоночными индексами. В разделе 5 описывается архитектура и реализации колоночного сопроцессора для реляционных СУБД. Раздел 6 содержит результаты вычислительных экспериментов по исследованию эффективности разработанных моделей, методов и алгоритмов

при обработке сверхбольших баз данных. В заключении суммируются основные результаты нашей работы, приводятся их отличия от ранее выполненных родственных работ других авторов, даются рекомендации по возможному практическому применению колоночных индексов и обозначаются направления дальнейших исследований.

2. КОЛОНОЧНАЯ МОДЕЛЬ ХРАНЕНИЯ ДАННЫХ

Колоночное представление данных предполагает хранение таблиц базы данных в виде отдельных столбцов. Колоночная модель хранения данных приобрела в последнее время большую популярность у исследователей, занимающихся проблематикой оперативной аналитической обработки информации в хранилищах данных [27, 28, 29, 30]. Колоночное представление в отличие от традиционному строкового представления оказывается намного более эффективным при выполнении запросов класса OLAP [25]. Это объясняется тем, что при выполнении запросов колоночная СУБД считывает с диска только те атрибуты, которые необходимы для выполнения запроса, что сокращает объем операций ввода-вывода и, как следствие, уменьшает время выполнения запроса. Недостатком колоночного представления является низкая эффективность при выполнении стоково-ориентированных операций, таких, например, как добавление или удаление кортежей. Вследствие этого колоночные СУБД могут проигрывать по производительности строковым при выполнении запросов класса OLTP. На рис. 1 схематично изображено основное отличие в физической организации колоночных и строчных хранилищ [29]: показаны три способа представления отношения Sales (Продажи), содержащего пять атрибутов. При колоночно-ориентированном подходе (рис. 1 (a) и (b)) каждая колонка хранится независимо как отдельный объект базы данных. Поскольку данные в СУБД записываются и считываются поблочно, колоночно-ориентированный подход предполагает, что каждый блок, содержащий информацию из таблицы продаж, включает в себя данные только по некоторой единственной колонке. В этом случае, запрос, вычисляющий, например, число продаж определенного продукта за июль месяц, должен получить доступ только к колонкам prodid (идентификатор продукта) и date (дата продажи). Следовательно, СУБД достаточно загрузить в оперативную память только те блоки, которые содержат данные из этих колонок. С другой стороны, при строчно-ориентированном подходе (рис. 1 (c)) существует единственный объект базы данных, содержащий все необходимые данные, то есть каждый блок на диске, содержащий информацию из таблицы Sales, включает в себя данные из всех колонок этой таблицы. В этом случае отсутствует возможность выборочно считать конкретные атрибуты, необходимые для конкретного запроса, без считывания всех остальных атрибутов отношения. Принимая во внимание тот факт, что затраты на обмены с диском (либо обмены между процессорным кэшем и оперативной памятью) являются узким место в системах баз данных, а схемы баз данных становятся все более сложными с включением широких таблиц с сотнями атрибутов, колоночные хранилища способны обеспечить существенный прирост в производительности при выполнении запросов класса OLAP.

Наиболее простой организацией колоночного хранилища является добавление к каждой колонке столбца суррогатных ключей [31], представляющих собой идентификатор строки в соответствующем отношении. В этом случае мы получаем так называемое двухстолбцовое представление с суррогатным ключом (рис. 1 (b)). В работе [32] предложен механизм виртуальных ключей. В этом случае роль суррогатного ключа играет порядковый номер значения в столбце. Учитывая тот факт, что все значения столбца имеют одинаковый тип (занимают одинаковое количество байт) и хранятся в непрерывной области памяти, по порядковому номеру значения можно определить область памяти, занимаемой этим значением. Реконструкция i-того кортежа отношения осуществляется путем выборки и соединения i-тых значений столбцов, принадлежащих этому отношению. В этом случае мы приходим к одностолбцовому представлению (рис. 1 (a)), что позволяет экономить память.

Эффективность сжатия данных в колоночных хранилищах была исследована в работах [26, 33]. Поскольку в колонке все значения имеют

Рис. 1.
Физическая организация колоночных и строчных хранилищ.

одинаковый тип, то для каждой колонки может быть подобран наиболее эффективный метод сжатия, специфичный для данного типа. Максимальный эффект от сжатия достигается на отсортированных данных, когда столбец содержит большие группы повторяющихся значений. Для повышения скорости выполнения запросов наиболее подходит "легковесное" сжатие данных (lightweight data compression) [33], при котором нагрузка на процессор для сжатия/распаковки данных не перевешивает выгоду от уменьшения времени на передачу сжатых данных [34, 35]. В [26, 36, 37] показано, что выполнение операций над данными в сжатом формате позволяет на порядок повысить производительность обработки запросов.

Одним из главных преимуществом строчных хранилищ является наличие в строковых СУБД мощных процедур оптимизации запросов, разработанных на базе реляционной модели. Строковые СУБД также имеют большое преимущество в скорости обработки запросов класса OLTP. В соответствии с этим в исследовательском сообществе баз данных были предприняты интенсивные усилия по интеграции преимуществ столбцовой модели хранения данных в строковые СУБД [25].

В работе [38] предложена новая схема зеркалирования данных, получившая название "разбитое зеркало" (fractured mirror). Этот подход предполагает гибридную схему хранения данных, включающую в себя и строковое и колоночное представления. На базе строкового представления выполняются операции модификации базы данных, а на базе колоночного — операции чтения и анализа данных. Изменения, производимые в строковом представлении, переносятся в колоночное представление с помощью фонового процесса. Все таблицы в строчном представлении делятся на фрагменты, распределяемые по различным дискам. Каждому строчному фрагменту на этом же диске сопоставляется его зеркальная копия в столбцовом представлении с использованием виртуальных ключей. Указанная схема хорошо подходит для параллельного выполнения запросов, не требующего обмена данными между узлами. Однако, при выполнении сложных запросов, требующих массовых обменов данными между процессорными узлами, данная схема приводит к большим накладным расходам на передачу сообщений. Предложенный подход предполагает для каждого запроса создание гибридного плана, состоящего фактически из комбинации двух планов: один — для строчного представления и второй — для колоночно представления. Оптимизация таких гибридных запросов порождает большое количество трудно решаемых проблем.

В работах [31, 39] была предложена модель хранения данных DSM (Decomposition Storage Model), которая предполагает декластеризацию каждого отношения на столбцы с использовани-

ем суррогатных ключей (см. рис. 1 (b)). В работе [25] была предпринята попытка эмулировать DSM в рамках реляционной СУБД System X. Каждое отношение разбивалось на колонки по числу атрибутов исходного отношения. Каждая колонка представлялась в виде бинарной (двухстолбцовой) таблицы. Первый столбец содержал суррогатный ключ, идентифицирующий соответствующую строку в исходной таблице, второй столбец содержал значение атрибута. При выполнении запроса с диска загружались только те столбцы, которые соответствуют атрибутам, указанным в запросе. Происходило их соединение по суррогатному ключу в "урезанные" таблицы, над которыми выполнялся запрос. В System X для этих целей по умолчанию используется хеш-соединение, которое оказалось очень дорогостоящим. Для исправления этого недостатка была предпринята попытка вычислять соединения с помощью кластеризованных индексов, создаваемых для каждого столбца, однако из-за обращения к индексам накладные расходы оказались еще выше.

У модели DSM имеются две проблемы. Во-первых, требуется хранение в каждой бинарной таблице столбца с суррогатными ключами, что приводит к излишним затратам дисковой памяти и дополнительным обменам с диском. Во-вторых, в большинстве строчных хранилищ вместе с каждым кортежем хранится относительно крупный заголовок, что также вызывает излишние расходы дисковой памяти (в колоночных хранилищах заголовки хранятся в отдельных столбцах во избежание этих накладных расходов). Для преодоления этих проблем в [25] был исследован подход, при котором отношения хранятся в обычном строчном виде, но на каждом столбце каждой таблицы определяется индекс в виде B-дерева. При выполнении SQL-запросов генерировались планы выполнения запросов, использующие только индексы (index-only plan). При выполнении таких планов никогда не производится доступ к реальным кортежам на диске. Хотя индексы явно хранят идентификаторы хранимых записей, каждое значение соответствующего столбца сохраняется только один раз, и обычно доступ к значениям столбцов через индексы сопровождается меньшими накладными расходами, поскольку в индексе не хранятся заголовки кортежей. Проведенные эксперименты показали, что такой подход демонстрирует существенно худшую производительность по сравнению с колоночными СУБД.

Еще один подход, рассмотренный в [25], использует материализованные представления. При применении этого подхода для каждого класса запросов создается оптимальный набор материализованных представлений, в котором содержатся только столбцы, требуемые в запросах этого класса. В этих представлениях не выполняется соединение столбцов разных таблиц. Цель этой стратегии состоит в том, чтобы дать возможность СУБД производить доступ только к тем данным на диске, которые действительно требуются. Этот подход работает лучше, чем подходы, базирующиеся на использовании DSM и index-only plan. Однако его применение требует предварительного знания рабочей нагрузки, что существенно ограничивает его использование на практике.

В [40] предпринимается еще одна попытка совместить преимущества строчного и колоночного хранилищ в рамках реляционной СУБД. Предлагается хранить данные в сжатой форме, используя отдельную таблицу (названную c-таблицей) для каждой колонки в исходной реляционной схеме (аналогично методу вертикальной декомпозиции отношений). Для сжатия данных используется метод кодирования по длинам периодов (RLE — Run-Length Encoding) [26]. На запросах класса OLTP данный метод показал производительность, сравнимую с колоночными СУБД. Однако данная схема предполагает создание c-таблицы для каждого атрибута, и, кроме этого, создание большого количества индексов для каждой c-таблицы, что приводит к большим затратам памяти, отводимой под хранилище. В дополнение к сказанному, зависимости между кортежами в c-таблицах приводят к неоправданно высокой стоимости операции добавления новых записей, даже для приложений с нечастыми обновлениями.

В [41] описан альтернативный подход к внедрению в строковую СУБД колоночных методов обработки запросов. В основе этого метода лежат планы выполнения запросов, использующие только индексы и специальные операторы

Index Merge, Index Merge Join, Index Hash Join, встраиваемые в ядро СУБД. Предлагаемые методы ориентированы на использование твердотельных накопителей (SSD) и многоядерных процессоров. Модифицированная таким образом строковая СУБД способна показать на OLAP-запросах производительность, сравнимую и даже превосходящую производительность колоночных СУБД.

В работе [42] описываются новые вспомогательные структуры данных — индексы колоночной памяти (column store indexes), внедренные в Microsoft SQL Server 11. Индексы колоночной памяти представляют собой колоночное хранилище в чистом виде, так как данные различных колонок хранятся в отдельных дисковых страницах, что позволяет значительно увеличить производительность операций ввода-вывода. Для повышения производительности при выполнении запросов класса OLTP пользователь должен создать индексы колоночной памяти для соответствующих таблиц. Решение об использовании индексов колоночной памяти в том или ином случае принимает СУБД, как это имеет место и для обычных индексов в виде B-деревьев. Авторы иллюстрируют преимущества новых индексов на тесте TPC DS [43]. Для таблицы catalog_sales (каталог продаж) создается индекс колоночной памяти, содержащий все 34 колонки этой таблицы. Таким образом, наряду со строчным хранилищем в Microsoft SQL Server 11 создается колоночное хранилище, в котором полностью дублируется информация для определенных колонок определенных таблиц. Построение индекса колоночной памяти происходит следующим образом. Исходная таблица делится на последовательные группы строк одинаковой длины. Столбцы значений атрибутов каждой группы кодируются и сжимаются независимо друг от друга. В результате получаются сжатые колоночные сегменты, каждый из которых сохраняется в виде самостоятельного BLOB (Binary Large Object) [44]. На этапе кодирования данные преобразуются в целые числа. Поддерживается две стратегии: 1) кодирование значения и 2) кодирование путем перечисления (подобно перечислимому типу в универсальных языках программирования). Затем осуществляется сжатие колонки методом RLE. Для достижения максимального эффекта от сжатия, исходные группы строк сортируются с помощью специального алгоритма Vertipaq. Для обработки запросов к колоночному хранилищу в Microsoft SQL Server 11 реализованы специальные операторы, поддерживающие блочную обработку данных [45] (в исполнителе для строчного хранилища используется традиционная покортежная обработка). Следует отметить, что блочные операторы не применимы к данным, хранящимся в сточной форме. Таким образом, в Microsoft SQL Server 11 фактически реализованы два независимых исполнителя запросов: один для строчного хранилища, другой для колоночного. СУБД при анализе SQL-запроса решает какому из двух исполнителей адресовать его выполнение. СУБД также обеспечивает механизмы синхронизации данных в обоих хранилищах.

Анализ рассмотренных решений показывает [25], что нельзя получить выгоду от хранения данных по столбцам, воспользовавшись системой баз данных со строковым хранением с вертикально разделенной схемой, либо проиндексировав все столбцы, чтобы обеспечить к ним независимый доступ. Системы баз данных с построчным хранением обладают существенно меньшей производительностью по сравнению с системами баз данных с поколоночным хранением на эталонном тестовом наборе для хранилищ данных Star Schema Benchmark (SSBM), [46, 47, 48]. Разница в производительности демонстрирует, что между системами имеются существенные различия на уровне выполнения запросов (кроме очевидных различий на уровне хранения).

3. ДОМЕННО-КОЛОНОЧНАЯ МОДЕЛЬ

В данном разделе строится новая доменно-колоночная модель представления данных и вводится понятие колоночного индекса.

Для обозначения реляционных операций мы будем использовать нотацию из известного учебника [49]. Под $\pi_{*\setminus A}(R)$ будем понимать проекцию на все атрибуты отношения R, за исключением атрибута A. С помощью символа “$\circ$” будем обозначать операцию конкатенации двух кортежей:

$$(x_1, \ldots, x_u) \circ (y_1, \ldots, y_v) = (x_1, \ldots, x_u, y_1, \ldots, y_v).$$

Под $R(A, B_1, \ldots, B_u)$ будем понимать отношение R с *суррогатным ключом* A (идентификатором целочисленного типа, однозначно определяющим кортеж) и атрибутами $B_1, \ldots, B_u$, представляющее собой множество кортежей длины $u+1$ вида $(a, b_1, \ldots, b_u)$, где $a \in \mathbb{Z}_{\geq 0}$ и $\forall j \in \{1, \ldots, u\} \left(b_j \in \mathfrak{D}_{B_j}\right)$. Здесь $\mathfrak{D}_{B_j}$ — домен атрибута B_j. Через $r.B_j$ будем обозначать значение атрибута B_j, через $r.A$ — значение суррогатного ключа кортежа r: $r = (r.A, r.B_1, \ldots, r.B_u)$. *Суррогатный ключ* отношения R обладает свойством $\forall r', r'' \in R\,(r' \neq r'' \Leftrightarrow r'.A \neq r''.A)$. Под *адресом кортежа* r будем понимать значение суррогатного ключа этого кортежа. Для получения кортежа отношения R по его адресу будем использовать *функцию разыменования* $\&_R$: $\forall r \in R\,(\&_R(r.A) = r)$.

Везде далее будем рассматривать отношения как множества, а не как мультимножества [49]. Это означает, что, если при выполнении некоторой операции получилось отношение с дубликатами, то к этому отношению по умолчанию применяется операция удаления дубликатов.

Пусть задано отношение $R(A, B, \ldots)$, $T(R) = n$. Пусть на множестве $\mathfrak{D}_B$ задано отношение линейного порядка. *Колоночным индексом* $I_{R.B}$ атрибута B отношения R будем называть упорядоченное отношение $I_{R.B}(A, B)$, удовлетворяющее следующим свойствам:

$$T(I_{R.B}) = n, \pi_A(I_{R.B}) = \pi_A(R); \tag{1}$$

$$\forall x_1, x_2 \in I_{R.B}\,(x_1 \leq x_2 \Leftrightarrow x_1.B \leq x_2.B); \tag{2}$$

$$\forall r \in R(\forall x \in I_{R.B} \quad (r.A = x.A \Rightarrow r.B = x.B)). \tag{3}$$

Условие (1) означает, что множества значений суррогатных ключей (адресов) индекса и индексируемого отношения совпадают. Условие (2) означает, что элементы индекса упорядочены в порядке возрастания значений атрибута B. Условие (3) означает, что атрибут A элемента индекса содержит адрес кортежа отношения R, имеющего такое же значение атрибута B, как и у данного элемента колоночного индекса.

С содержательной точки зрения колоночный индекс $I_{R.B}$ представляет собой таблицу из двух

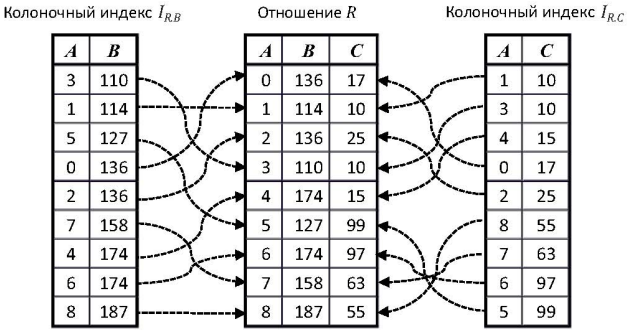

Рис. 2.
Колоночный индекс.

колонок с именами A и B. Количество строк в колоночном индексе совпадает с количеством строк в индексируемой таблице. Колонка B индекса $I_{R.B}$ включает в себя все значения колонки B таблицы R (с учетом повторяющихся значений), отсортированных в порядке возрастания. Каждая строка x индекса $I_{R.B}$ содержит в колонке A суррогатный ключ (адрес) строки r в таблице R имеющей такое же значение в колонке B что и x. На рис. 2 представлены примеры двух различных колоночных индексов для одного и того же отношения.

Пусть на множестве значений домена $\mathfrak{D}_B$ задано отношение линейного порядка. Разобьем множество $\mathfrak{D}_B$ на $k > 0$ непересекающихся интервалов:

$$\left.\begin{aligned} &V_0 = [v_0; v_1], V_1 = (v_1; v_2], \ldots, \\ &V_{k-1} = (v_{k-1}; v_k]; \\ &v_0 < v_1 < \ldots < v_k; \\ &\mathfrak{D}_B = \bigcup_{i=0}^{k-1} V_i. \end{aligned}\right\} \tag{4}$$

Отметим, что в случае $\mathfrak{D}_B = \mathbb{R}$ будем иметь $v_0 = -\infty$ и $v_0 = +\infty$. Функция $\varphi_{\mathfrak{D}_B} : \mathfrak{D}_B \to \{0, \ldots, k-1\}$ называется *доменной функцией фрагментации* для $\mathfrak{D}_B$, если она удовлетворяет следующему условию:

$$\forall i \in \{0, \ldots, k-1\} \quad (\forall b \in \mathfrak{D}_B\,(\varphi_{\mathfrak{D}_B}(b) = i \Leftrightarrow b \in V_i)). \tag{5}$$

Другими словами, доменная функция фрагментации сопоставляет значению b — номер интервала, которому это значение принадлежит.

Пусть задан колоночный индекс $I_{R.B}$ для отношения $R(A, B, \ldots)$ с атрибутом B над доменом $\mathfrak{D}_B$ и доменная функция фрагментации $\varphi_{\mathfrak{D}_B}$. Функция

$$\varphi_{I_{R.B}} : I_{R.B} \to \{0, \ldots, k-1\}, \qquad (6)$$

определенная по правилу

$$\forall x \in I_{R.B} \left(\varphi_{I_{R.B}}(x) = \varphi_{\mathfrak{D}_B}(x.B)\right), \qquad (7)$$

называется *доменно-интервальной функцией фрагментации* для индекса $I_{R.B}$. Другими словами, функция фрагментации $\varphi_{I_{R.B}} : I_{R.B}$ сопоставляет каждому кортежу x из $I_{R.B}$ номер доменного интервала, которому принадлежит значение $x.B$.

Определим i-ый фрагмент $(i = 0, \ldots, k-1)$ индекса $I_{R.B}$ следующим образом:

$$I^i_{R.B} = \{x | x \in I_{R.B};\ \varphi_{I_{R.B}}(x) = i\}. \qquad (8)$$

Это означает, что в i-тый фрагмент попадают кортежи, у которых значение атрибута B принадлежит i-тому доменному интервалу. Будем называть фрагментацию, построенную таким образом, *доменно-интервальной*. Количество фрагментов k будем называть *степенью фрагментации*.

Доменно-интервальная фрагментация обладает следующими фундаментальными свойствами, вытекающими непосредственно из ее определения:

$$I_{R.B} = \bigcup_{i=0}^{k-1} I^i_{R.B}; \qquad (9)$$

$$\forall i, j \in \{0, \ldots, k-1\} \left(i \neq j \Rightarrow I^i_{R.B} \cap I^j_{R.B} = \emptyset\right). \qquad (10)$$

На рис. 3 схематично изображена фрагментация колоночного индекса, имеющая степень $k = 3$.

Пусть для отношения $R(A, B, C, \ldots)$ заданы колоночные индексы $I_{R.B}$ и $I_{R.C}$. *Транзитивной фрагментацией* индекса $I_{R.C}$ относительно индекса $I_{R.B}$ называется фрагментация, задаваемая функцией $\ddot{\varphi}_{I_{R.C}} : I_{R.C} \to \{0, \ldots, k-1\}$, удовлетворяющей условию: $\forall x \in I_{R.C}$

$$\ddot{\varphi}_{I_{R.C}}(x) = \varphi_{I_{R.B}}\left(\sigma_{A=x.A}(I_{R.B})\right). \qquad (11)$$

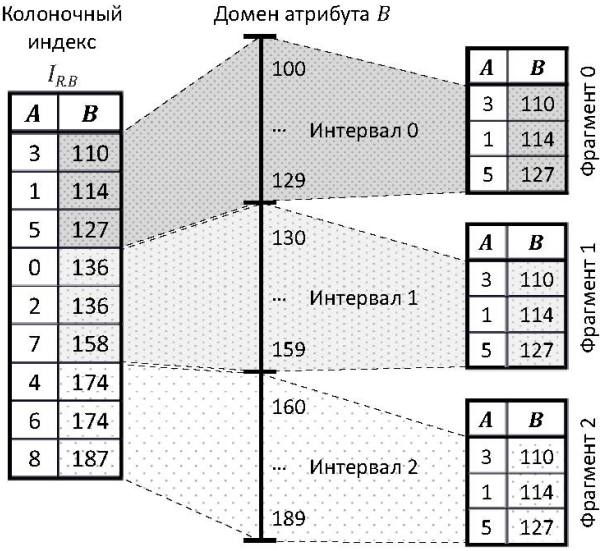

Рис. 3.
Фрагментация колоночного индекса.

Транзитивная фрагментация позволяет разместить на одном и том же узле элементы колоночных индексов, соответствующие одному кортежу индексируемого отношения.

4. ДЕКОМПОЗИЦИЯ РЕЛЯЦИОННЫХ ОПЕРАЦИЙ

В этом разделе описывается декомпозиция основных реляционных операций для распределенных колоночных индексов. Декомпозиция заключается в разбиении алгоритма выполнения операции на отдельные подзадачи, не требующих обменов данными.

Декомпозиция естественного соединения. Рассмотрим декомпозицию операции естественного соединения вида $\pi_{*\setminus A}(R) \bowtie \pi_{*\setminus A}(S)$. Пусть заданы два отношения $R(A, B_1, \ldots, B_u, C_1, \ldots, C_v)$ и $S(A, B_1, \ldots, B_u, D_1, \ldots, D_w)$. Пусть имеется два набора колоночных индексов по атрибутам $B_1, \ldots, B_u$: $I_{R.B_1}, \ldots, I_{R.B_u}$; $I_{S.B_1}, \ldots, I_{S.B_u}$. Пусть для всех этих индексов задана доменно-интервальная фрагментация степени k:

$$I_{R.B_j} = \bigcup_{i=0}^{k-1} I^i_{R.B_j};\ I_{S.B_j} = \bigcup_{i=0}^{k-1} I^i_{S.B_j}.$$

Положим

$$P_j^i = \pi_{I^i_{R.B_j}.A\to A_R,\ I^i_{S.B_j}.A\to A_S}(I^i_{R.B_j} \underset{I^i_{R.B_j}.B_j=I^i_{S.B_j}.B_j}{\bowtie} I^i_{S.B_j}) \quad (12)$$

для всех $i = 0, \ldots, k-1$. Определим

$$P = \bigcup_{i=0}^{k-1} P^i. \quad (13)$$

Построим отношение

$$Q(B_1, \ldots, B_u, C_1, \ldots, C_v, D_1, \ldots, D_w)$$

следующим образом:

$$Q = \{(\&_R(p.A_R).B_1, \ldots, \&_R(p.A_R).B_u, \&_B(p.A_B).C_1, \ldots, \&_B(p.A_B).C_v, \&_S(p.A_S).D_1, \ldots, \&_S(p.A_S).D_w) | p \in P\}. \quad (14)$$

Тогда $Q = \pi_{*\setminus A}(R) \bowtie \pi_{*\setminus A}(S)$. Доказательство этого факта приводится в работе авторов [50]. Пример вычисления операции естественного соединения двух отношений с использованием распределенных колоночных индексов приведен в работе авторов [51].

Декомпозиция операции группировки. Рассмотрим декомпозицию операции группировки вида $\gamma_{B,C_1,\ldots,C_u,\mathrm{agrf}(D_1,\ldots,D_w)\to F}(R)$. Пусть задано отношение

$$R(A, B, C_1, \ldots, C_u, D_1, \ldots, D_w, \ldots)$$

с суррогатным ключом A. Пусть для атрибутов $D_1, \ldots, D_w$ задана агрегирующая функция **agrf**. Пусть имеется колоночный индекс $I_{R.B}$. Пусть также имеются колоночные индексы: $I_{R.C_1}, \ldots, I_{R.C_u}$; $I_{R.D_1}, \ldots, I_{R.D_w}$. Пусть для индекса $I_{R.B}$ задана доменно-интервальная фрагментация степени k: $I_{R.B} = \bigcup_{i=0}^{k-1} I^i_{R.B}$. Пусть для индексов $I_{R.C_1}, \ldots, I_{R.C_u}$ и $I_{R.D_1}, \ldots, I_{R.D_w}$ задана транзитивная относительно $I_{R.B}$ фрагментация

$$\forall j \in \{1, \ldots, u\} \left(I_{R.C_j} = \bigcup_{i=0}^{k-1} I^i_{R.C_j} \right);$$

$$\forall j \in \{1, \ldots, w\} \left(I_{R.D_j} = \bigcup_{i=0}^{k-1} I^i_{R.D_j} \right).$$

Положим

$$P_i = \pi_{A,\ F}(\gamma_{\min(A)\to A,B,C_1,\ldots,C_u,\mathrm{agrf}(D_1,\ldots,D_w)\to F}(I^i_{R.B} \bowtie I^i_{R.C_1} \bowtie \ldots \bowtie I^i_{R.C_u} \bowtie \bowtie I^i_{R.D_1} \bowtie \ldots \bowtie I^i_{R.D_w})) \quad (15)$$

для всех $i = 0, \ldots, k-1$. Определим $P = \bigcup_{i=0}^{k-1} P_i$. Построим отношение $Q(B, C_1, \ldots, C_u, F)$ следующим образом:

$$Q = \{(\&_R(p.A).B, \&_R(p.A).C_1, \ldots, \&_R(p.A).C_u, p.F) | p \in P\}. \quad (16)$$

Тогда $Q = \gamma_{B,C_1,\ldots,C_u,\mathrm{agrf}(D_1,\ldots,D_w)\to F}(R)$. Доказательство корректности описанной декомпозиции операции группировки приводится в работе авторов [52].

Декомпозиция операции пересечения. Рассмотрим декомпозицию операции пересечения вида $\pi_{B_1,\ldots,B_u}(R) \cap \pi_{B_1,\ldots,B_u}(S)$. Пусть заданы два отношения $R(A, B_1, \ldots, B_u)$ и $S(A, B_1, \ldots, B_u)$, имеющие одинаковый набор атрибутов. Пусть имеется два набора колоночных индексов по атрибутам $B_1, \ldots, B_u$: $I_{R.B_1}, \ldots, I_{R.B_u}$; $I_{S.B_1}, \ldots, I_{S.B_u}$. Пусть для всех этих индексов задана доменно-интервальная фрагментация степени k:

$$I_{R.B_j} = \bigcup_{i=0}^{k-1} I^i_{R.B_j}; I_{S.B_j} = \bigcup_{i=0}^{k-1} I^i_{S.B_j}.$$

Положим

$$P_j^i = \pi_{I^i_{R.B_j}.A\to A_R,\ I^i_{S.B_j}.A\to A_S}(I^i_{R.B_j} \underset{(I^i_{R.B_j}.B_j=I^i_{S.B_j}.B_j)}{\bowtie} I^i_{S.B_j}) \quad (17)$$

для всех $i = 0, \ldots, k-1$ и $j = 1, \ldots, u$. Определим $P_j = \bigcup_{i=0}^{k-1} P_j^i$. Положим $P = \bigcap_{j=1}^{u} P_j$. Построим отношение $Q(A, B_1, \ldots, B_u)$ следующим образом:

$$Q = \{r | r \in R \wedge r.A \in \pi_{A_R}(P)\}. \quad (18)$$

Тогда $\pi_{B_1,\ldots,B_u}(Q) = \pi_{B_1,\ldots,B_u}(R) \cap \pi_{B_1,\ldots,B_u}(S)$. Доказательство корректности описанной декомпозиции операции пересечения приводится в нашей работе [53].

С использованием описанной методики авторами также была выполнена декомпозиция операций проекции, выбора, удаления дубликатов и объединения [54].

5. КОЛОНОЧНЫЙ СОПРОЦЕССОР КСОП

На базе описанной в разделе 3 доменно-колоночной модели представления данных и методов декомпозиции реляционных операций разработана программная система “Колоночный СОПроцессор (КСОП)” (Columnar COProcessor CCOP) для кластерных вычислительных систем. В данном разделе описывается архитектура и реализации колоночного сопроцессора КСОП.

Колоночный сопроцессор КСОП — это программная система, предназначенная для управления распределенными колоночными индексами, размещенными в оперативной памяти кластерной вычислительной системы. Назначение КСОП — вычислять таблицы предварительных вычислений (ТПВ) для ресурсоемких реляционных операций по запросу СУБД. Общая схема взаимодействия СУБД и КСОП изображена на рис. 4. КСОП включает в себя программу “Координатор”, запускаемую на узле вычислительного кластера с номером 0, и программу “Исполнитель”, запускаемую на всех остальных узлах, выделенных для работы КСОП. На SQL-сервере устанавливается специальная программа “Драйвер КСОП”, обеспечивающая взаимодействие с координатором КСОП по протоколу TCP/IP. КСОП работает только с данными целых типов 32 или 64 байта. При создании колоночных индексов для атрибутов других типов, их значение кодируется в виде целого числа или вектора целых чисел. КСОП поддерживает следующие основные операции, доступные СУБД через интерфейс драйвера КСОП: CreateColumnIndex (создание распределенного колоночного индекса), Execute (выполнение запроса на вычисление ТПВ), Insert (добавление в колоночный индекс нового кортежа), TransitiveInsert (добавление в колоночный индекс нового кортежа по транзитивному значению), Delete (удаление из колоночного индекса кортежа), TransitiveDelete (удаление из колоночного индекса кортежа по транзитивному значению).

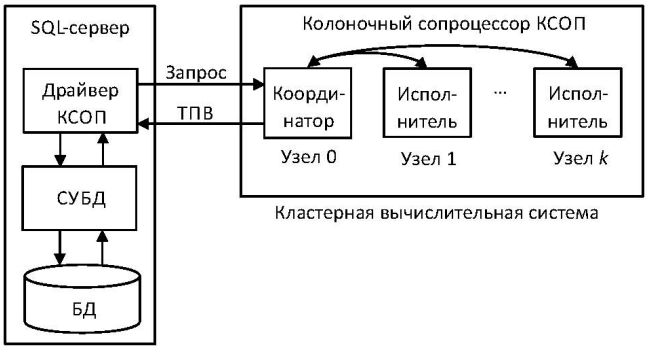

Рис. 4.
Взаимодействие SQL-сервера с КСОП.

Для организации взаимодействия между драйвером и КСОП был разработан язык CCOPQL (CCOP Query Language), базирующийся на формате данных JSON. Каждый колоночный индекс делится на фрагменты, которые в свою очередь делятся на сегменты. Все сегменты одного фрагмента располагаются в сжатом виде в оперативной памяти одного процессорного узла. Для сжатия сегментов использовалась библиотека Zlib [55, 56], реализующая метод сжатия DEFLATE [57], являющийся комбинацией методов Хаффмана и Лемпеля-Зива.

Поясним общую логику работы КСОП на простом примере. Пусть имеется база данных из двух отношений $R(A, B, D)$ и $S(A, B, C)$, хранящихся на SQL сервере (см. рис. 5). Пусть нам необходимо выполнить запрос:

```
SELECT D, C
FROM R, S
WHERE R.B = S.B AND C<13.
```

Предположим, что КСОП имеет только два узла-исполнителя и на каждом узле имеется три процессорных ядра (процессорные ядра на рис. 5 промаркированы обозначениями $P_{11}, \ldots, P_{23}$). Положим, что атрибуты $R.B$ и $S.B$ определены на домене целых чисел из интервала [0; 120). Сегментные интервалы для $R.B$ и $S.B$ определим следующим образом: $[0; 20)$, $[20; 40)$, $[40; 60)$, $[60; 80)$, $[80; 100)$, $[100; 120)$. В качестве фрагментных интервалов для $R.B$

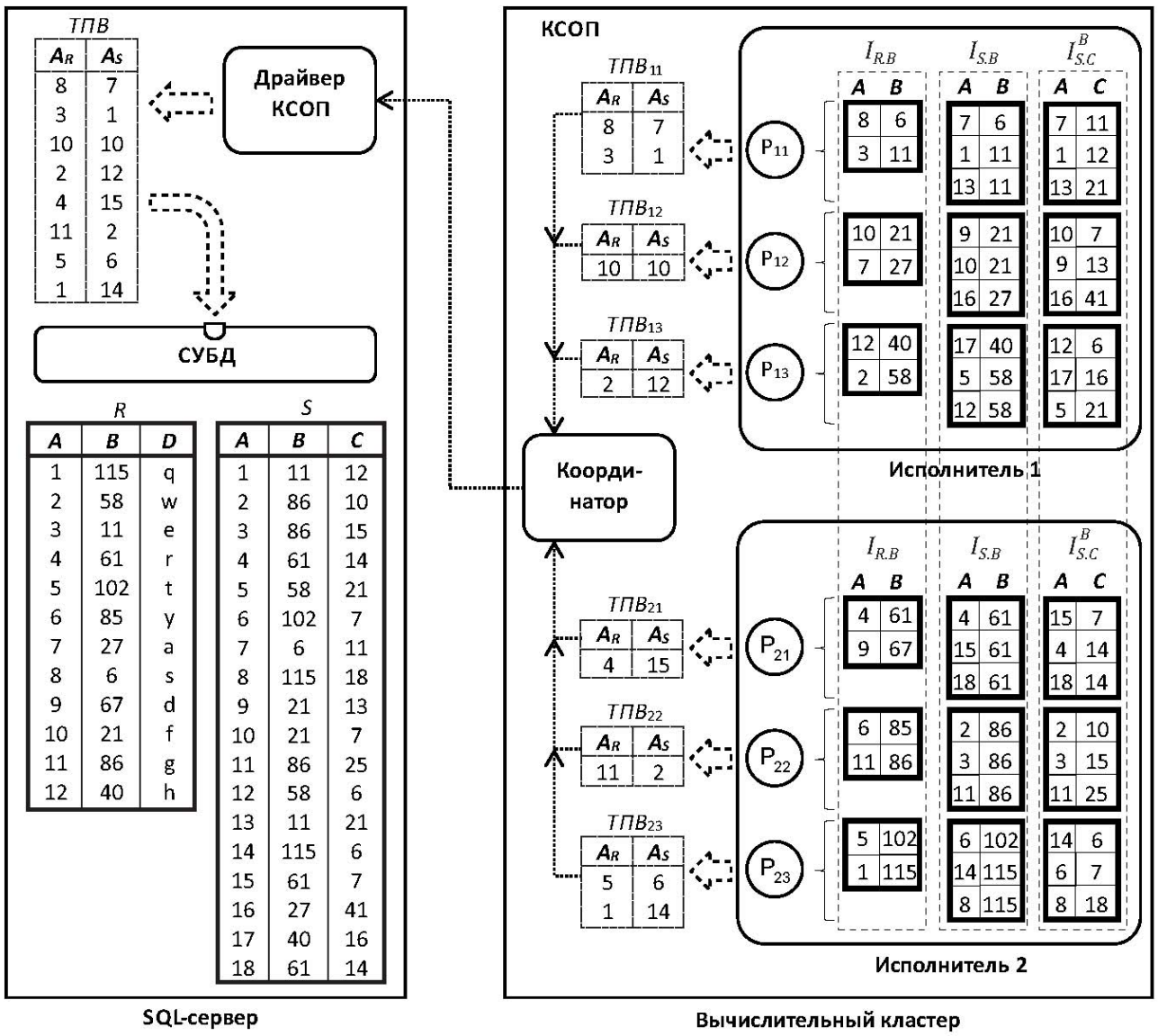

Рис. 5.
Вычисление ТПВ с использованием КСОП.

и $S.B$ зафиксируем: [0; 59] и [60,119]. Пусть атрибут $S.$ определен на домене целых чисел из интервала $[0; 25]$. Изначально администратор базы данных с помощью драйвера КСОП создает для атрибутов $R.B$ и $S.B$ распределенные колоночные индексы $I_{R.B}$ и $I_{S.B}$. Затем для атрибута $S.$ создается распределенный колоночный индекс $I^B_{S.C}$, который фрагментируется и сегментируется транзитивно относительно индекса $I_{S.B}$. Распределенные колоночные индексы $I_{R.B}$, $I_{S.B}$ и $I^B_{S.C}$ сохраняются в оперативной памяти узлов-исполнителей. Таким образом мы получаем распределение данных внутри КСОП, приведенное на рис. 5. При поступлении SQL запроса он преобразуется драйвером КСОП в план, определяемый следующим выражением реляционной алгебры:

$$\pi_{I_{R.B}.A \to A_R, I_{S.B}.A \to A_S}(I_{R.B} \underset{I_{R.B}.B = I_{S.B}.B}{\bowtie} (I_{S.B} \bowtie \sigma_{C<13}(I^B_{S.C}))).$$

При выполнении драйвером операции Execute указанный запрос передается координатору КСОП в виде оператора CCOPQL в формате JSON. Запрос выполняется независимо

процессорными ядрами узлов-исполнителей над соответствующими группами сегментов. При этом за счет доменной фрагментации и сегментации не требуются обмены данными как между узлами-исполнителями, так и между процессорными ядрами одного узла. Каждое процессорное ядро вычисляет свою часть ТПВ, которая пересылается на узел-координатор. Координатор объединяет фрагменты ТПВ в единую таблицу и пересылает ее драйверу, который выполняет материализацию этой таблицы в виде отношения в базе данных, хранящейся на SQL сервере. После этого SQL сервер вместо исходного SQL оператора, выполняет следующий оператор:

```
SELECT D, C
FROM
    R INNER JOIN (
      ТПВ INNER JOIN S ON (S.A = ТПВ.AS)
    ) ON (R.A = ТПВ.AR).
```

При этом используются обычные кластеризованные индексы в виде B-деревьев, заранее построенные для атрибутов R.A и S.A.

Колоночный сопроцессор КСОП был реализован на языке Си с использованием аппаратнонезависимых параллельных технологий MPI и OpenMP. Объем исходного кода составил около двух с половиной тысяч строк. Исходные тексты КСОП свободно доступны в сети Интернет по адресу: https://github.com/elena-ivanova/colomnindices/.

6. ВЫЧИСЛИТЕЛЬНЫЕ ЭКСПЕРИМЕНТЫ

В данном разделе приводятся результаты вычислительных экспериментов по исследованию эффективности разработанных моделей, методов и алгоритмов обработки сверхбольших баз данных с использованием распределенных колоночных индексов.

Эксперименты проводились на двух вычислительных комплексах с кластерной архитектурой: “Торнадо ЮУрГУ” и “RSC PetaStream” МСЦ РАН. Система “Торнадо ЮУрГУ” [58] включает в себя 384 процессорных узлов, соединенных сетями InfiniBand QDR и Gigabit Ethernet. В состав процессорного узла входит два шестиядерных ЦПУ Intel Xeon X5680, ОЗУ 24 Гб и сопроцессор Intel Xeon Phi SE10X (61 ядро по 1.1 ГГц), соединенные шиной PCI Express. Система “RSC PetaStream” [59] состоит из 8 модулей, включающих в себя 8 сопроцессоров Intel Xeon Phi 7120, каждый из которых имеет 61 процессорное ядро и 16 Гб встроенной памяти GDDR5. Модули соединены между собой сетями InfiniBand FDR и Gigabit Ethernet. Непосредственно на каждом сопроцессоре (на одном ядре) загружается операционная система Linux CentOS 7.0.

Для тестирования колоночного сопроцессора КСОП использовалась синтетическая база данных, построенная на основе эталонного теста TPC-H [60].

Тестовая база данных состояла из двух таблиц: ORDERS (ЗАКАЗЫ) и CUSTOMER (КЛИЕНТЫ). Структура этих таблиц приведена в работе [61]. Для имитирования перекоса данных использовались следующие распределения значения атрибута ORDERS.ID_CUSTOMER (внешний ключ, определяющий идентификатор клиента, сделавшего заказ): равномерное (uniform), “45-20”, “65-20”, “80-20” [62]. Для варьирования размера результирующего отношения использовался коэффициент селективности Sel, принимающий значение из интервала $]0;1]$. Коэффициент Sel определяет размер (в кортежах) результирующего отношения относительно размера отношения ORDERS. Для масштабирования базы данных использовался масштабный коэффициент SF (Scale Factor), значение которого изменялось от 1 до 10. При проведении экспериментов размер отношения ORDERS составлял $SF \times 63\,000\,000$ кортежей, размер отношения CUSTOMER — $SF \times 630\,000$ кортежей.

В первом эксперименте исследовалась балансировка загрузки процессорных ядер Xeon Phi при различных перекосах в распределении значений внешнего ключа ORDERS.ID_CUSTOMER. Результаты эксперимента представлены на рис. 6. Из графиков видно, что при малом количестве сегментов сильный перекос по данным приводит к существенному дисбалансу в загрузке процессорных ядер. В случае, когда количество сегментов равно 60 и совпадает с количеством ядер, время выполнения операции при распределении “80-20” более чем в четыре раза превышает время выполнения той же операции при равномерном (uniform) распределении. Однако при увеличении количества сегментов влияние перекоса

по данным нивелируется. Для распределения “45-20” оптимальным оказывается число сегментов, равное 10 000, для распределения “65-20” — 20 000, и для распределения “80-20” — 200 000.

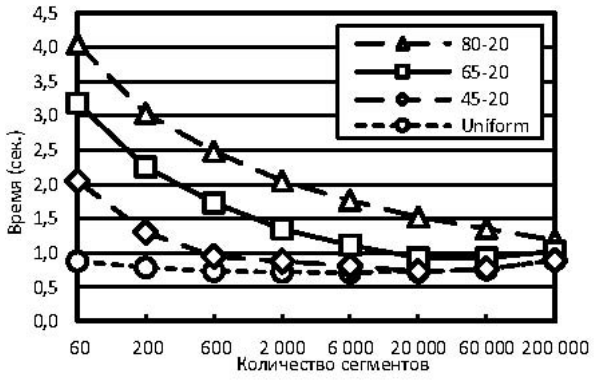

Рис. 6.
Балансировка загрузки процессорных ядер сопроцессора Xeon Phi.

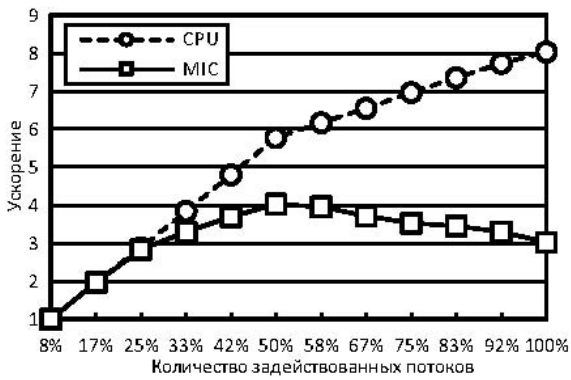

Рис. 7.
Влияние гиперпоточности на ускорение.

Целью второго эксперимента было определить, насколько эффективно гиперпоточность может быть применена при работе колоночного сопроцессора КСОП. ЦПУ Intel Xeon X5680 имеет аппаратную поддержку двух потоков на ядро, сопроцессор Intel Xeon Phi поддерживает четыре потока на ядро. Результаты эксперимента представлены на рис. 7. Эксперимент показал, что для CPU производительность растет вплоть до максимального числа аппаратно поддерживаемых потоков. Однако при использовании одного потока на ядро на CPU наблюдается ускорение, близкое к идеальному, а при использовании двух потоков на ядро прирост ускорения становится более медленным. Применительно к MIC картина меняется. При использовании одного потока на ядро на MIC наблюдается ускорение, близкое к идеальному. При использовании двух потоков на ядро прирост ускорения становится более медленным. Использование же большего количества потоков на одно ядро ведет к деградации производительности.

В третьем и четвертом экспериментах была исследована масштабируемость КСОП при работе на вычислительных системах с массовым параллелизмом. На рис. 8 приведены кривые ускорения при вычислении колоночным сопроцессором ТПВ на кластере “Торнадо ЮУрГУ”, на рис. 9 — на кластере “RSC PetaStream”. Графики на рис. 8 показывают, что селективность запроса Sel оказывается фактором, ограничивающим масштабируемость. Так, для $Sel = 0.0005$ кривая ускорения становится практически линейной, а для $Sel = 0.005$ приближается к линейной. Однако при большом значении коэффициента селективности $Sel = 0.05$ масштабируемость ограничивается 150 процессорными узлами. Причина в том, что при $Sel = 0.05$ время передачи, распаковки и слияния ТПВ меняется мало и существенно превышает время ее вычисления, в то время как при $Sel = 0.0005$ значение времени передачи, распаковки и слияния ТПВ почти на порядок меньше времени ее вычисления. В соответствии с этим можно сделать вывод, что при малой селективности запроса КСОП демонстрирует на больших базах данных ускорение, близкое к линейному.

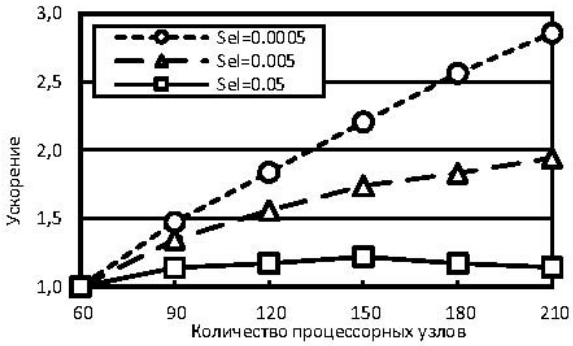

Рис. 8.
Ускорение вычисления ТПВ на кластере “Торнадо ЮУрГУ” при $SF = 10$.

Аналогичная картина наблюдалась при тестировании КСОП на вычислительном кластере “RSC PetaStream” (рис. 9). В следующем экспе-

Таблица 1. Время вычисления SQL-запроса и ускорение в сравнении с PostgreSQL при $SF = 1$.

3*Конфигурация	Время в минутах					
	$Sel = 0.0005$		$Sel = 0.005$		$Sel = 0.05$	
	1-й запуск	2-й запуск	1-й запуск	2-й запуск	1-й запуск	2-й запуск
PostgreSQL	7.3	1.21	7.6	1.29	7.6	1.57
PostgreSQL & B-Tree	2.62	2.34	2.83	2.51	2.83	2.63
PostgreSQL & CCOP	0.073	0.008	0.65	0.05	2.03	1.72
Ускорение						
$\frac{t_{PostgreSQL}}{t_{PostgreSQL\ \&\ CCOP}}$	100	151	12	27	4	0.9
$\frac{t_{PostgreSQL\&B-Trees}}{t_{PostgreSQL\ \&\ CCOP}}$	36	293	4	50	1.4	1.53

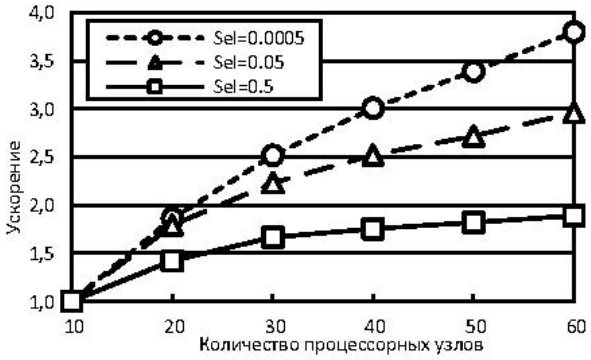

Рис. 9.
Ускорение вычисления ТПВ на кластере "RSC PetaStream" при $SF = 1$.

рименте было исследовано в какой мере использование колоночного сопроцессора КСОП может ускорить выполнение запроса класса OLAP в реляционной СУБД. В ходе выполнения эксперимента исследовались следующие три конфигурации:

1. PostgreSQL: выполнение запроса без создания индексных файлов в виде B-деревьев.
2. PostgreSQL & B-Trees: выполнение запроса с предварительным созданием индексных файлов в виде B-деревьев для атрибутов соединения.
3. PostgreSQL & CCOP: выполнение запроса с использованием ТПВ и предварительным созданием индексных файлов в виде B-деревьев для суррогатных ключей.

В последнем случае ко времени выполнения запроса добавлялось время создания ТПВ колоночным сопроцессором КСОП (CCOP). В каждом случае замерялось время первого и повторного запуска запроса. Это связано с тем, что после первого выполнения запроса PostgreSQL собирает статистическую информацию, сохраняемую в словаре базы данных, которая затем используется для оптимизации плана выполнения запроса. Эксперименты показали (см. табл. 1), что при отсутствии индексов в виде B-деревьев использование колоночного сопроцессора позволяет увеличить производительность выполнения запроса в 100–150 раз для коэффициента селективности $Sel = 0.0005$. Однако при больших значениях коэффициента селективности эффективность использования КСОП может снижаться вплоть до отрицательных значений (ускорение меньше единицы).

7. ЗАКЛЮЧЕНИЕ

В статье были рассмотрены вопросы разработки и исследования эффективных методов параллельной обработки сверхбольших баз данных с использованием колоночного представления информации, ориентированных на кластерные вычислительные системы, оснащенные многоядерными ускорителями, и допускающих интеграцию с реляционными СУБД. Введены колоночные индексы с суррогатными ключами. Предложена доменно-колоночная модель распределения данных по узлам многопроцессорной системы, на основе которой разработаны формальные методы декомпозиции реляционных операций на части, которые могут выполняться независимо (без обменов данными) на различных вычислительных узлах и процессорных ядрах. Корректность методов декомпозиции во всех случаях подтверждена математическими доказательствами. Разработанные методы реализованы в колоночном

сопроцессоре КСОП, который работает на кластерных вычислительных системах, в том числе оснащенных многоядерными ускорителями, и может применяться в сочетании с реляционной СУБД для выполнения ресурсоемких операций. Взаимодействие СУБД с КСОП осуществляется через специальный драйвер, который устанавливается на SQL сервере, где работает СУБД. Для организации этого взаимодействия в код СУБД встраивается специальный коннектор.

Основные результаты являются новыми и не покрываются ранее опубликованными научными работами других авторов, обзор которых был дан в разделе 2 . Отметим основные отличия.

В работе [42] описываются индексы колоночной памяти (column store indexes), внедренные в Microsoft SQL Server 11. Этот подход фактически предполагает создание двух независимых исполнителя запросов (один для строчного хранилища, другой для колоночного) в рамках одной СУБД. Подход, описанный в настоящей статье, предполагает реализацию колоночных индексов в рамках отдельной программной системы — колоночного сопроцессора КСОП. Кроме этого, в работе [42] отсутствует описание методов фрагментации индексов колоночной памяти и ничего не говорится о методах распараллеливания операций над индексами колоночной памяти. Также следует отметить, что индексы колоночной памяти представляются в Microsoft SQL Server 11 в виде BLOB-объектов, хранимых на дисках. В КСОП колоночные индексы в распределенном виде хранятся в оперативной памяти кластерной вычислительной системы.

Методы, предложенные в работе [41] для включения в строчную СУБД колоночно-ориентированной обработки запросов, требуют глубокой модернизации СУБД и не приспособлены для работы на кластерных вычислительных системах с распределенной памятью. В отличие от этого КСОП требует минимальной модификации СУБД путем добавления в нее специального коннектора и показывает масштабируемость, близкую к линейной, на кластерных вычислительных системах с сотнями процессорных узлов и десятками тысяч процессорных ядер. В работе [40] в строчной СУБД вводятся дополнительные структуры данных, называемые c-таблицами, напоминающие колоночные индексы КСОП, однако полностью отсутствуют аспекты, связанные распределением данных и параллельной обработкой.

Подход к эмуляции колоночного хранилища в строчной СУБД, основанный на материализованных представлениях [25], не позволяет соединять в одном представлении колонки разных таблиц, что ограничивает его применимость для параллельного выполнения соединений в многопроцессорных системах с распределенной памятью. В отличие от этого КСОП способен выполнять соединения без обменов данными на кластерных вычислительных системах с большим количеством процессорных узлов.

Схема зеркалирования данных “разбитое зеркало” (fractured mirror) из [38] позволяет организовывать эффективную параллельную обработку фрагментно-независимых запросов в колоночном стиле на кластерной вычислительной системе. Однако, если запрос зависит от способа фрагментации данных, происходит деградация производительности системы из-за большого количества пересылок данных между процессорными узлами. Доменно-интервальная фрагментация, используемая в КСОП, позволяет избежать таких пересылок. Другие подходы к эмуляции колоночного хранилища в строчной СУБД, описанные в [25], здесь не рассматриваются, так как они показываю худшую производительность по сравнению с обычными реляционными СУБД. В отличие от них КСОП позволяет ускорить выполнение запросов к хранилищу данных в сотни раз по сравнению с реляционной СУБД PostgreSQL.

Полученные результаты могут применяться при создании масштабируемых колоночных сопроцессоров для существующих коммерческих и свободно-распространяемых SQL-СУБД. Это позволит обрабатывать сверхбольшие хранилища данных на кластерных вычислительных системах, в том числе с узлами, включающими в себя многоядерные ускорители типа GPU или MIC.

Оценку объема оперативной памяти, необходимой для хранения колоночных индексов, можно выполнить на основе стандартного теста TPC-H [60], используемого для моделирования

обработки аналитических баз данных. Число записей во всех таблицах теста TPC-H составляет 8 661 245 при коэффициенте масштабирования $SF = 1$. Предположим, что для каждой таблицы создается три колоночных индекса. Как показали проведенные нами эксперименты колоночные индексы сжимаются в среднем в 3 раза. Учитывая, что запись колоночного индекса состоит из двух полей по 8 байт, получаем, что объем оперативной памяти, необходимой для хранения колоночных индексов, при $SF = 1$ составляет $8\,661\,245 \times 3 \times 16/3 \approx 0.13$ гигабайт. При $SF = 30\,000$, что соответствует среднему объему базы данных, объем необходимой оперативной памяти составит 4.2 терабайта. При максимальном значении $SF = 100\,000$, что соответствует большой базе данных, для хранения колоночных индексов понадобится 14 терабайт. Вычислительный кластер "Торнадо ЮУрГУ" [58] имеет суммарную память 13.8 терабайт (384 узла с памятью 24 Гб и 96 узлов с памятью 48 Гб). Эта система занимает 349 место в 46-ой редакции рейтинга TOP500 (ноябрь 2015) и, таким образом, относится к вычислительным кластерам средней мощности. Проведенные расчеты показывают, что оперативной памяти подобной системы вполне достаточно для хранения аналитических баз данных средних размеров. Для сверхбольших баз данных понадобятся более масштабные вычислительные системы. Отметим, однако, что оперативная память системы под номером 1 в 46-ой редакции списка TOP500 (ноябрь 2015) составляет один петабайт, что вполне достаточно для хранения колоночных индексов аналитической базы данных очень большого размера.

В качестве направлений дальнейших исследований можно выделить следующие.

- Разработка и исследование методов интеграции КСОП со свободно распространяемыми реляционными СУБД типа PostgreSQL.
- Интеграция в КСОП легковесных методов сжатия, не требующих распаковки для выполнения операций над данными.
- Обобщение предложенных подходов и методов на многомерные данные.

СПИСОК ЛИТЕРАТУРЫ

1. *Turner V., Gantz J.F., Reinsel D., et al.* The Digital Universe of Opportunities: Rich Data and the creasing Value of the Internet of Things. IDC white paper. 2014. http://www.idcdocserv.com/1678.
2. Big Data Insights. Microsoft. 2013. https://blogs.msdn.microsoft.com/ /microsoftenterpriseinsight/2013/04/12/big-data-insights/.
3. *Stonebraker M., Madden S., Dubey P.* Intel "big data" science and technology center vision and execution plan // ACM SIGMOD Record, 2013. Vol. 42, No. 1. P. 44–49.
4. *Harizopoulos S., Abadi D., Madden S., Stonebraker M.* OLTP Through the Looking Glass, And What We Found There // Proceedings of the ACM SIGMOD International Conference on Management of Data. 2008. P. 981–992.
5. *Williams M.H., Zhou S.* Data Placement in Parallel Database Systems // Parallel database techniques, 1998. P. 203–218.
6. TOP500: 500 most powerful computer systems in the world. http://www.top500.org.
7. *Kostenetskii P.S., Sokolinsky L.B.* Simulation of Hierarchical Multiprocessor Database Systems // Programming and Computer Software, 2013. Vol. 39, No. 1. P. 10–24.
8. *Lepikhov A.V., Sokolinsky L.B.* Query Processing in a DBMS for Cluster Systems // Programming and Computer Software. 2010. Vol. 36. No. 4. P. 205–215.
9. *Lima A.A., Furtado C., Valduriez P., Mattoso M.* Parallel OLAP Query Processing in Database Clusters with Data Replication // Distributed and Parallel Databases, 2009. Vol. 25, No. 1-2. P. 97–123.
10. *Pukdesree S., Lacharoj V., Sirisang P.* Performance Evaluation of Distributed Database on PC Cluster Computers // WSEAS Transactions on Computers, 2011. Vol. 10, No. 1. P. 21–30.
11. *Соколинский Л.Б.* Параллельные системы баз данных. М.: Издательство Московского государственного университета, 2013. 184 с.
12. *Taniar D., Leung C.H.C., Rahayu W., Goel S.* High Performance Parallel Database Processing and Grid Databases. John Wiley & Sons, 2008. 554 p.

13. *Sokolinsky L.B.* Survey of Architectures of Parallel Database Systems // Programming and Computer Software. 2004. Vol. 30, No. 6. P. 337–346.

14. *Deshmukh P.A.* Review on Main Memory Database // International Journal of Computer & Communication Technology, 2011. Vol. 2, No. 7. P. 54–58.

15. *Garcia-Molina H., Salem K.* Main Memory Database Systems: An Overview // IEEE Transactions On Knowledge and Data Engineering, 1992. Vol. 4, No. 6. P. 509–516.

16. *Plattner H., Zeier A.* In-Memory Data Management: An Inflection Point for Enterprise Applications. Springer, 2011. 254 p.

17. *LeHong H., Fenn J.* Hype Cycle for Emerging Technologies. Research Report. Gartner. 2013. 85 p.

18. *Chaudhuri S., Dayal U.* An Overview of Data Warehousing and OLAP Technology // SIGMOD Record, 1997. Vol. 26, No. 1. P. 65–74.

19. *Furtado P.* A Survey of Parallel and Distributed Data Warehouses // International Journal of Data Warehousing and Mining, 2009. Vol. 5, No. 5. P. 57–77.

20. *Golfarelli M., Rizzi S.* A Survey on Temporal Data Warehousing // International Journal of Data Warehousing and Mining, 2009. Vol. 5, No. 1. P. 1–17.

21. *Oueslati W., Akaichi J.* A Survey on Data Warehouse Evolution // International Journal of Database Management Systems, 2010. Vol. 2, No. 4. P. 11–24.

22. *Boncz P.A., Kersten M.L.* MIL primitives for querying a fragmented world // VLDB Journal, 1999. Vol. 8, No. 2. P. 101–119.

23. *Boncz P.A., Zukowski M., Nes N.* MonetDB/X100: Hyper-pipelining query execution // Proceedings of the Second Biennial Conference on Innovative Data Systems Research (CIDR), 2005. P. 225–237.

24. *Stonebraker M., Abadi D.J., Batkin A., Chen X., Cherniack M., Ferreira M., Lau E., Lin A., Madden S.R., O'Neil E.J., O'Neil P.E., Rasin A., Tran N., Zdonik S.B.* C-Store: A Column-Oriented DBMS // Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05), 2005. P. 553–564.

25. *Abadi D.J., Madden S.R., Hachem N.* Column-Stores vs. Row-Stores: How Different Are They Really? // Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008. P. 967–980.

26. *Abadi D. J., Madden S. R., Ferreira M.* Integrating compression and execution in column-oriented database systems // Proceedings of the 2006 ACM SIGMOD international conference on Management of data, 2006. P. 671–682.

27. *Чернышев Г. А.* Организация физического уровня колоночных СУБД // Труды СПИИРАН. 2013. № 7. Вып. 30. С. 204–222.

28. *Abadi D.J., Boncz P.A., Harizopoulos S.* Column-oriented Database Systems // Proceedings of the VLDB Endowment. 2009. Vol. 2, No. 2. P. 1664–1665.

29. *Abadi D.J., Boncz P.A., Harizopoulos S., Idreos S., Madden S.* The Design and Implementation of Modern Column-Oriented Database Systems // Foundations and Trends in Databases. 2013. Vol. 5, No. 3. P. 197–280.

30. *Plattner H.* A common database approach for OLTP and OLAP using an in-memory column database // Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, 2009. P. 1–2.

31. *Copeland G. P., Khoshafian S. N.* A decomposition storage model // Proceedings of the 1985 ACM SIGMOD international conference on Management of data, 1985. P. 268–279.

32. *Idreos S., Groffen F., Nes N., Manegold S., Mullender S., Kersten M. L.* MonetDB: Two Decades of Research in Column-oriented Database Architectures // IEEE Data Engineering Bulletin. 2012. Vol. 35, No. 1. P. 40–45.

33. *Zukowski M., Heman S., Nes N., Boncz P.* Super-Scalar RAM-CPU Cache Compression // Proceedings of the 22nd International Conference on Data Engineering, 2006. P. 59–71.

34. *Chen Z., Gehrke J., Korn F.* Query Optimization in Compressed Database Systems // Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, 2001. P. 271–282.

35. *Westmann T., Kossmann D., Helmer S., Moerkotte G.* The implementation and performance of compressed databases // ACM SIGMOD Record, 2000. Vol. 29, No. 3. P. 55–67.

36. *Aghav S.* Database compression techniques for performance optimization // Proceedings of the 2010 2nd International Conference on Computer Engineering and Technology (ICCET), 2010. P. 714–717.

37. *Lemke C., Sattler K.-U., Faerber F., Zeier A.* Speeding up queries in column stores: a case for compression // Proceedings of the 12th international conference on Data warehousing and knowledge discovery (DaWaK'10), 2010. P. 117–129.

38. *Ramamurthy R., Dewitt D., Su Q.* A case for fractured mirrors // Proceedings of the VLDB Endowment. 2002. Vol. 12, No. 2. P. 89–101.

39. *Khoshafian S., Copeland G., Jagodis T., Boral H., Valduriez P.* A query processing strategy for the decomposed storage model // Proceedings of the Third International Conference on Data Engineering, 1987. P. 636–643.

40. *Bruno N.* Teaching an Old Elephant New Tricks // Online Proceedings of Fourth Biennial Conference on Innovative Data Systems Research (CIDR 2009), 2009. http://www-db.cs.wisc.edu/cidr/cidr2009/Paper_2.pdf.

41. *El-Helw A., Ross K.A., Bhattacharjee B., Lang C.A., Mihaila G.A.* Column-oriented query processing for row stores // Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP (DOLAP '11), 2011. P. 67–74.

42. *Larson P.-A., Clinciu C., Hanson E. N., Oks A., Price S. L., Rangarajan S., Surna A., Zhou Q.* SQL server column store indexes // Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (SIGMOD '11), 2011. P. 1177–1184.

43. TPC Benchmark DS – Standard Specification. Transaction Processing Performance Council. 2015. 135 p. http://www.tpc.org/TPC_Documents_Current_Versions/pdf/tpc-ds_v2.1.0.pdf.

44. *Shapiro M., Miller E.* Managing databases with binary large objects // 16th IEEE Symposium on Mass Storage Systems. 1999. P. 185–193.

45. *Padmanabhan S., Malkemus T., Agarwal R., Jhingran A.* Block oriented processing of relational database operations in modern computer architectures // Proceedings of the 17th International Conference on Data Engineering, 2001. P. 567–574.

46. *O'Neil P. E., Chen X., O'Neil E. J.* Adjoined Dimension Column Index to Improve Star Schema Query Performance // Proceedings of the 24th International Conference on Data Engineering (ICDE 2008), 2008. P. 1409–1411.

47. *O'Neil P. E., O'Neil E. J., Chen X.* The Star Schema Benchmark (SSB). Revision 3, June 5, 2009.http://www.cs.umb.edu/ poneil/ /StarSchemaB.PDF

48. *O'Neil P. E., O'Neil E. J., Chen X., Revilak S.* The Star Schema Benchmark and Augmented Fact Table Indexing // Performance Evaluation and Benchmarking, First TPC Technology Conference (TPCTC 2009), 2009. P. 237–252.

49. *Гарсиа-Молина Г., Ульман Дж., Уидом Дж.* Системы баз данных. Полный курс. М.: Издательский дом «Вильямс», 2004. 1088 с.

50. *Ivanova E., Sokolinsky L.B.* Join Decomposition Based on Fragmented Column Indices // Lobachevskii Journal of Mathematics. 2016. Vol. 37, No. 3. P. 255–260.

51. *Иванова Е.В., Соколинский Л.Б.* Использование сопроцессоров Intel Xeon Phi для выполнения естественного соединения над сжатыми данными // Вычислительные методы и программирование: Новые вычислительные технологии. 2015. Т. 16, Вып. 4. С. 534–542.

52. *Иванова Е.В., Соколинский Л.Б.* Декомпозиция операции группировки на базе распределенных колоночных индексов // Наука ЮУрГУ: Материалы 67-ой научной конференции профессорско-преподавательского состава, аспирантов и сотрудников ЮУрГУ. Секции естественных наук. 2015. С. 15–22.

53. *Иванова Е.В., Соколинский Л.Б.* Декомпозиция операций пересечения и соединения на основе доменно-интервальной фрагментации колоночных индексов // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2015. Т. 4, No 1. С. 44–56.

54. *Иванова Е.В. Соколинский Л.Б.* Параллельная декомпозиция реляционных операций на основе распределенных колоночных индексов // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2015. Т. 4, № 4. С. 80–100.

55. *Deutsch P., Gailly J.-L.* ZLIB Compressed Data Format Specification version 3.3. RFC Editor. 1996. https://www.ietf.org/rfc/rfc1950.txt.

56. *Roelofs G., Gailly J., Adler M.* Zlib: A Massively Spiffy Yet Delicately Unobtrusive Compression Library. http://www.zlib.net/

57. *Deutsch P.* DEFLATE Compressed Data Format Specification version 1.3. RFC Editor. 1996. https://www.ietf.org/rfc/rfc1951.txt.

58. *Kostenetskiy P.S., Safonov A.Y.* SUSU Supercomputer Resources // Proceedings of the 10th Annual International Scientific Conference on Parallel Computing Technologies (PCT 2016). CEUR Workshop Proceedings. Vol. 1576, CEUR-WS 2015. P. 561–573.

59. Массивно-параллельный суперкомпьютер RSC PetaStream. http://rscgroup.ru/ru/our-solutions/massivno-parallelnyy-superkompyuter-rsc-petastream.

60. TPC Benchmark H — Standard Specification. Transaction Processing Performance Council. 2014. 136 p. http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf.

61. *Иванова Е.В., Соколинский Л.Б.* Колоночный сопроцессор баз данных для кластерных вычислительных систем // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2015. Т. 4, № 4. С. 5–31.

62. *Gray J., Sundaresan P., Englert S., Baclawski K., Weinberger P. J.* Quickly generating billion-record synthetic databases // Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, 1994. P. 243–252.