УДК 004.657 DOI: 10.14529/cmse150401

КОЛОНОЧНЫЙ СОПРОЦЕССОР БАЗ ДАННЫХ ДЛЯ КЛАСТЕРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Е.В. Иванова, Л.Б. Соколинский

Статья посвящена вопросам проектирования и реализации колоночного сопроцессора баз данных для реляционных СУБД. Колоночный сопроцессор (КСОП) разработан на базе колоночной модели хранения данных и ориентирован на большие кластерные вычислительные системы. КСОП может работать как на обычных центральных процессорах, так и на сопроцессорах с архитектурой MIC. КСОП поддерживает колоночные индексы с суррогатными ключами, которые во фрагментированном виде хранятся в оперативной памяти кластерной вычислительной системы. Фрагментация осуществляется на основе доменно-интервального принципа. На запросах класса OLAP колоночный сопроцессор КСОП демонстрирует производительность, многократно превышающую производительность строчных хранилищ.

Ключевые слова: колоночный сопроцессор, КСОП, распределенные колоночные индексы, доменно-интервальная фрагментация, кластерные вычислительные системы, многоядерные сопроцессоры, архитектура MIC.

Введение

В последнее время большую популярность приобрели колоночные СУБД (column-store DBMS) [1, 2]. В колоночной СУБД каждое отношение базы данных физически разбивается на столбцы (колонки), которые хранятся отдельно друг от друга. Раздельное хранение колонок позволяет колоночной СУБД считывать с диска только те колонки, которые соответствуют атрибутам, задействованным в запросе, в то время как строчная СУБД всегда считывает кортежи целиком. Это позволяет значительно экономить время на операциях ввода-вывода. Аналогичные преимущества колоночные СУБД получают при копировании данных из основной памяти в регистры. Помимо этого, колоночные СУБД позволяют эффективно использовать ряд технических приемов, недоступных или неэффективных для строчных СУБД. Первоначально колоночные хранилища были реализованы в ряде «академических» СУБД, среди которых следует упомянуть MonetDB [3], VectorWise (первоначальное название MonetDB/X100) [4] и C-Store [5]. Одной из первых коммерческих колоночных СУБД стала SybaseIQ [6]. Академические колоночные СУБД VectorWise и C-Store позднее эволюционировали в коммерческие системы Ingres VectorWise [7] и Vertica [8] соответственно. Начиная с 2013 года все основные производители СУБД включили в линейку продуктов колоночные версии своих систем: IBM [9], Microsoft [10–12], SAP [13] и Oracle [14]. В качестве современных колоночных СУБД также можно отметить EXASOL [15–17], Actian Vector [18], InfoBright [19] и SAND [20].

На рис. 1 схематично изображено основное отличие в физической организации колоночных и строчных хранилищ [2]: показаны три способа представления отношения Sales (Продажи), содержащего пять атрибутов. При колоночно-ориентированном подходе (рис. 1а и 1б) 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.

Рис. 1. Физическая организация колоночных и строчных хранилищ

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. 1в) 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.

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 [21]. Анализ предложенных решений показывает, 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.

Анализ современных тенденций в развитии аппаратного обеспечения и технологий баз данных говорит в пользу оптимальности следующего выбора среди возможных решений при разработке СУБД для обработки больших данных, схематично изображенного на рис. 2. Перспективным решением является создание колоночного сопроцессора КСОП (*CCOP — Columnar COProcessor*), совместимого с реляционной СУБД. Колоночный сопроцессор должен поддерживать распределенные *колоночные индексы*, постоянно хранимые в оперативной памяти кластерной вычислительной системы с многоядерными процессорными устройствами. Для взаимодействия с КСОП СУБД должна эмулировать материализационную модель обработки запроса. Суть материализационной модели состоит в том, что промежуточные отношения вычисляются полностью, за исключением атрибутов, не входящих в предикаты вышестоящих реляционных операций.

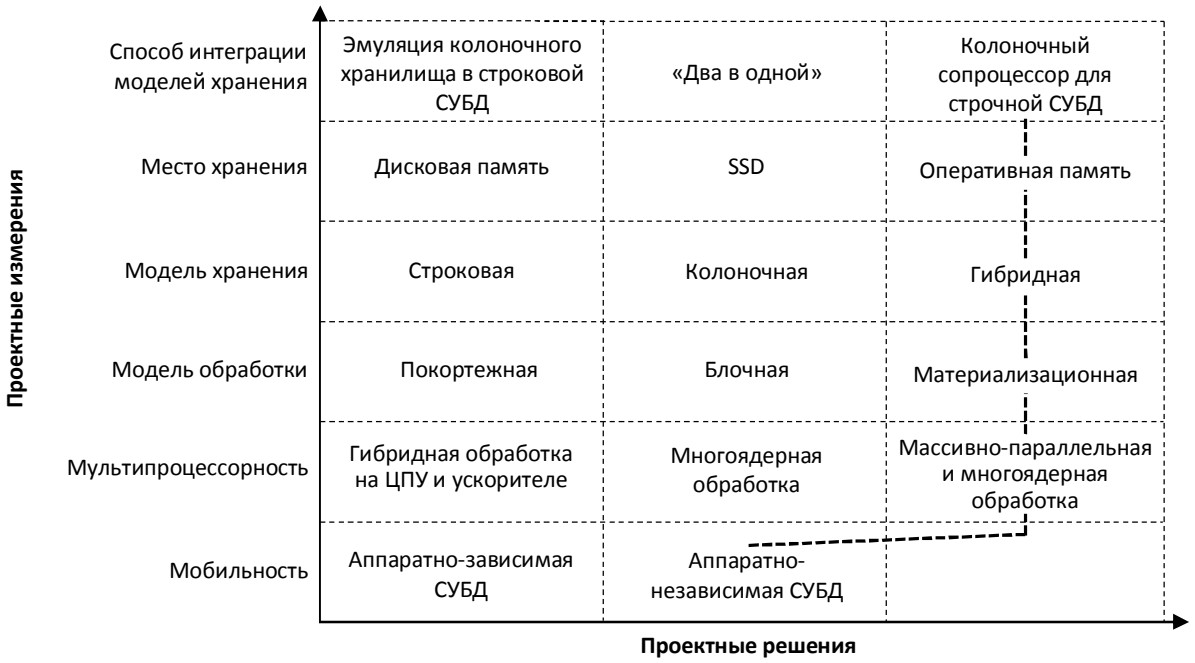


Рис. 2. Пространство проектных решений при разработке СУБД для больших данных

Это достигается переписыванием исходного SQL-запроса в последовательность запросов, использующих материализуемые представления. При таком подходе любая операция в плане выполнения запроса может быть заменена на вызов КСОП при условии, что в его памяти существуют необходимые колоночные индексы. Для сокращения расходов на разработку и сопровождение КСОП, он должен использовать аппаратно-независимые алгоритмы. Временные потери, связанные с отсутствием тонкого тюнинга под конкретную аппаратную платформу, должны компенсироваться хорошей масштабируемостью всех основных алгоритмов КСОП, используемых для выполнения запроса и для модификации колоночных индексов.

В данной, статье описываются архитектура, методы проектирования и реализации колоночного сопроцессора КСОП, удовлетворяющего вышеперечисленным требованиям. Теоретической основой КСОП является доменно-интервальная модель представления распределенных колоночных индексов, предложенная авторами в работах [22–25]. Статья имеет следующую структуру. В разделе 1 рассматривается архитектура колоночного сопроцессора КСОП, приводится общая схема взаимодействия СУБД и КСОП, даны интерфейсы основных операций, поддерживаемых КСОП. В разделе 2 описывается метод фрагментации и сегментации колоночных индексов, а также методы кодирования и сжатия данных, используемые в КСОП. В разделе 3 приведен пример выполнения запроса, поясняющий общую логику работы КСОП. В раздел 4 приводится структура КСОП и описываются схемы реализации основных операторов. В разделе 5 приводятся результаты вычислительных экспериментов по исследованию эффективности КСОП. В заключении суммируются полученные результаты, делаются итоговые выводы и обозначаются направления дальнейших исследований.

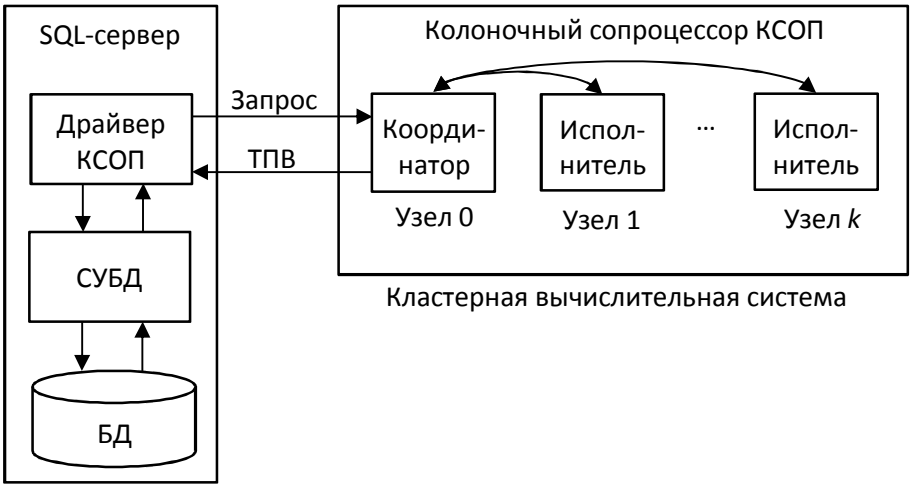


Рис. 3. Взаимодействие SQL-сервера с колоночным сопроцессором КСОП

1. Системная архитектура

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 для ресурсоемких реляционных операций по запросу СУБД. Общая схема взаимодействия СУБД и КСОП изображена на рис. 3. 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. В последнем случае длина вектора является фиксированной и называется *размерностью колоночного индекса*.

КСОП поддерживает следующие основные операции, доступные СУБД через интерфейс драйвера КСОП.

- CreateColumnIndex(TableID, ColumnID, SurrogateID, Width, Bottom, Top, Dimension) — создание распределенного колоночного индекса для атрибута ColumnID отношения TableID с параметрами: SurrogateID — идентификатор суррогатного ключа, Width — разрядность (32 или 64 бита); Bottom, Top — нижняя и верхняя границы доменного интервала; Dimension — размерность колоночного индекса. Возвращаемое значение: CIndexID — идентификатор созданного колоночного индекса.
- Insert(CIndexID, SurrogateKey, Value[*]) — добавление в колоночный индекс CIndexID нового кортежа (SurrogateKey, Value[*]).
- TransitiveInsert(CIndexID, SurrogateKey, Value[*], TValue[*]) — добавление в колоночный индекс CIndexID нового кортежа (SurrogateKey, Value[*]) с фрагментацией и сегментацией, определяемыми значением TValue[*].

- Delete(CIndexID, SurrogateKey, Value[*]) — удаление из колоночного индекса кортежа (SurrogateKey, Value[*]).
- TransitiveDelete(TCIndexID, SurrogateKey, TransitiveValue[*]) — удаление из колоночного индекса кортежа (SurrogateKey, Value[*]) с фрагментацией и сегментацией, определяемыми значением TValue[*].
- Execute(Query) — выполнение запроса на вычисление ТПВ с параметрами: Query — символьная строка, содержащая запрос в формате JSON. Примеры запросов приведены в разделе 3.2. Возвращаемое значение: PCTID — идентификатор Таблицы предварительных вычислений.

2. Управление данными

Все данные (колоночные индексы и метаданные), с которыми работает КСОП, хранятся в распределенной памяти кластерной вычислительной системы. Для колоночных индексов поддерживается двухуровневая система их разбиения на непересекающиеся части. Поясним ее работу на примере, изображенном на рис. 4, где показано разбиение колоночного индекса, построенного для атрибута B. В основе разбиения лежит домен $\mathfrak{D}_B = [0;99]$, на котором определен атрибут B. Домен разбивается на сегментные интервалы равной длины: [0; 11), [11; 22), ..., [77; 88), [88; 99]. Количество сегментных интервалов должно превышать суммарное количество процессорных ядер на узлах-исполнителях. Сегментные интервалы разбиваются на последовательные группы, которые называются *фрагментными интервалами*. В примере на рис. 4 это следующие интервалы: [0; 22), [22; 55), [55; 99]. Количество фрагментных интервалов должно совпадать с количеством узлов-исполнителей. Длины фрагментных интервалов могут не совпадать. Это необходимо для балансировки загрузки процессорных узлов в условиях перекоса данных.

На первом этапе распределения данных исходный колоночный индекс разбивается на фрагменты. Каждому фрагменту соответствует определенный фрагментный интервал. Кортеж (a, b) попадает в данный фрагмент тогда, и только тогда, когда значение b принадлежит соответствующему фрагментному интервалу. Все кортежи, принадлежащие одному фрагменту, хранятся на одном и том же процессорном узле. На втором этапе каждый фрагмент разбивается на сегменты. Каждому сегменту соответствует определенный сегментный интервал. Кортеж (a, b) попадает в данный сегмент тогда, и только тогда, когда значение b принадлежит соответствующему сегментному интервалу. Внутри каждого сегментного интервала записи сортируются в порядке возрастания значения атрибута.

В случае неравномерного распределения значений атрибута, по которому создан колоночный индекс, можно добиться примерно одинакового размера фрагментов, сдвигая границы фрагментных интервалов, как это сделано в примере на рис. 4. При этом могут получиться сегменты разной длины. В КСОП сегмент является наименьшей единицей распределения работ между процессорными ядрами. Если количество сегментов не велико по сравнению с количеством процессорных ядер на одном узле, то при выполнении запроса мы получим дисбаланс в загрузке процессорных ядер. Проблему балансировки

загрузки можно эффективно решить, уменьшив длину сегментных интервалов, и тем самым увеличив количество сегментов.

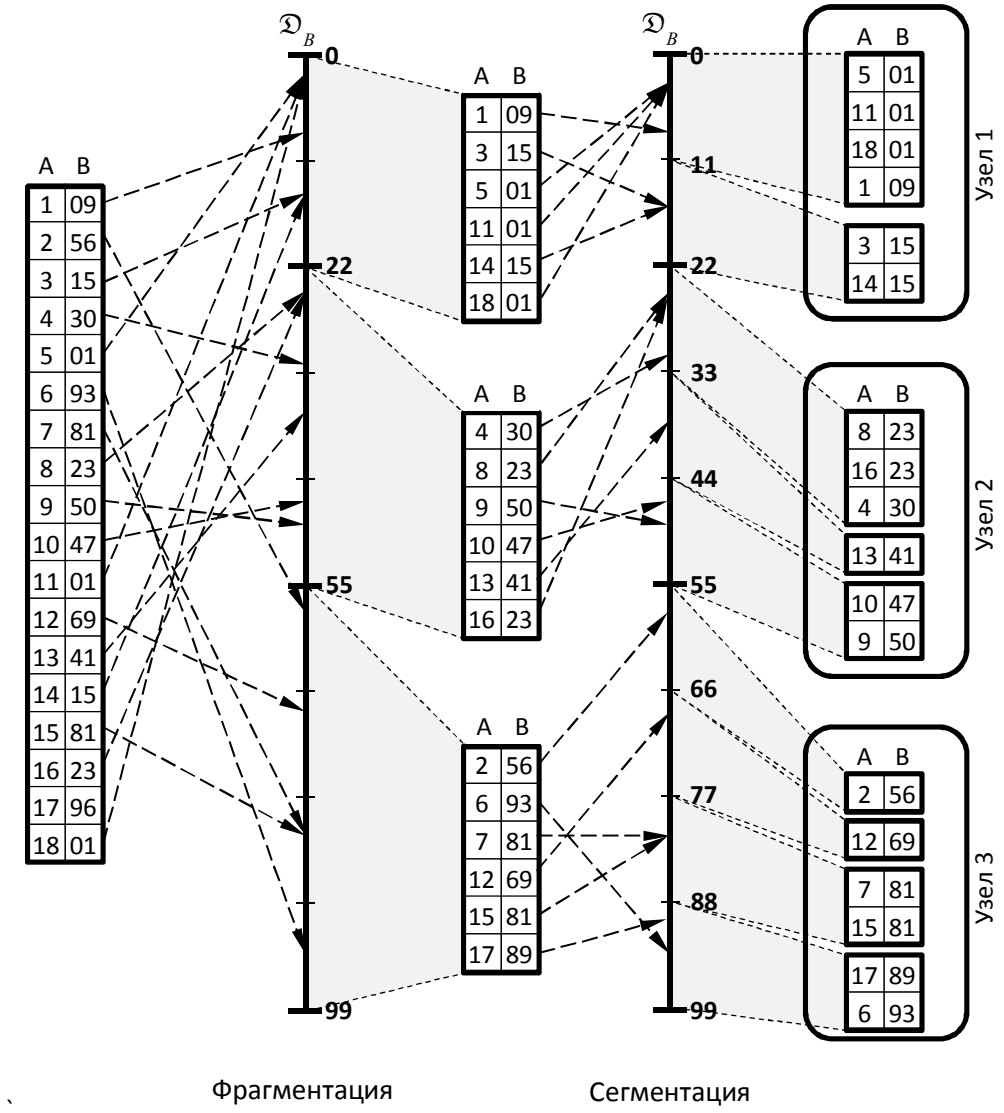


Рис. 4. Пример двухуровневого разбиения колоночного индекса на фрагменты и сегменты

КСОП работает только с данными целых типов (32 или 64 бита). Данные других типов должны быть предварительно закодированы драйвером КСОП в виде последовательности целых чисел. При этом могут использоваться методы, описанные в работе [11]. В случае, когда значение атрибута кодируется в виде целочисленного вектора размерности n (например, это может быть применено для длинных символьных строк), домен превращается в n-мерный куб. В этой ситуации n-мерный куб разбивается на n-мерные параллелепипеды по числу процессорных узлов (аналог фрагментного интервала), каждый из которых разбивается на еще меньшие n-мерные параллелепипеды по числу процессорных ядер в одном узле (аналог сегментного интервала).

Для сжатия закодированных сегментов могут использоваться как «тяжеловесные» методы (например, Хаффмана [26] или Лемпеля—Зива [27]), так и «легковесные»

(например, Run-Length Encoding [28, 29] или Null Suppression [30]), либо их комбинация [31]. В версии КСОП, описываемой в этой статье, для сжатия сегментов использовалась библиотека Zlib [32, 33], реализующая метод сжатия DEFLATE [34], являющийся комбинацией методов Хаффмана и Лемпеля—Зива. В работе [28] было показано, что легковесные методы типа Run-Length Encoding в случае колоночного представления информации могут оказаться более эффективными, чем тяжеловесные, поскольку допускают выполнение операций над данными без их распаковки.

3. Пример выполнения запроса

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},..., 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 $R.B$ и $S.B$ зафиксируем: [0; 59] и [60; 119]. Пусть атрибут $S.C$ 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 $S.C$ 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, приведенного выше, он преобразуется драйвером КСОП в план, определяемый следующим выражением реляционной алгебры:

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

При выполнении драйвером операции Exec 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 от-

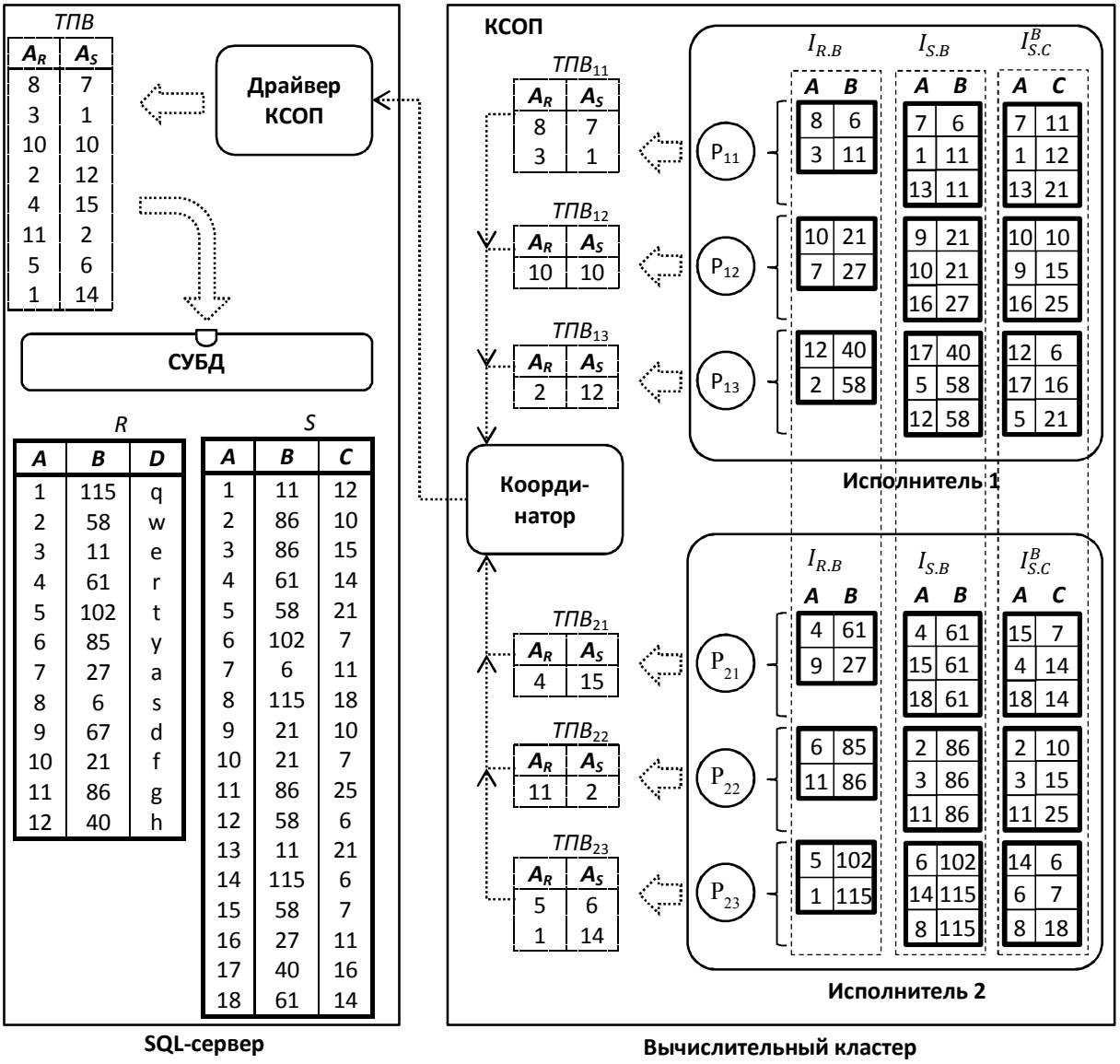


Рис. 5. Вычисление ТПВ с использованием КСОП

ношения 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.A_S)
   ) ON (R.A = ТПВ.A_R).
```

При этом используются обычные кластеризованные индексы в виде B-деревьев, заранее построенные для атрибутов $R.A$ и $S.A$. После выполнения запроса ТПВ удаляется, а распределенные колоночные индексы $I_{R.B}$, $I_{S.B}$ и $I^B_{S.C}$ сохраняются в оперативной памяти узлов-исполнителей для последующего использования.

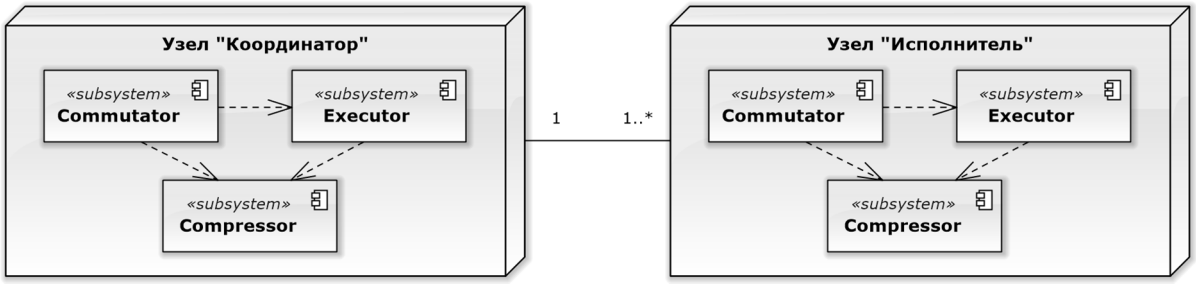


Рис. 6. Структура КСОП

4. Проектирование и реализация

На рис. 6 представлена диаграмма развертывания КСОП, основанная на нотации стандарта UML 2.0. КСОП является распределенной системой и включает в себя два типа узлов: координаторы и исполнители. В простейшем случае в системе имеется один узел-координатор и несколько узлов-исполнителей, как это показано на рис. 6. Однако, если при большом количестве исполнителей координатор становится узким местом, в системе может быть несколько координаторов. В этом случае они образуют иерархию в виде сбалансированного дерева, листьями которого являются узлы-исполнители.

И координатор, и исполнитель имеют унифицированную структуру, состоящую из трех компонент: *Commutator* (*Коммутатор*), *Executor* (*Исполнитель*), *Compressor* (*Модуль сжатия*). Однако их реализации в координаторе и исполнителе различается. Компонент Commutator в координаторе выполняет две функции: 1) обмен сообщениями в формате JSON с драйвером КСОП по протоколу TCP/IP (см. рис. 3); 2) обмен сообщениями с исполнителями по технологии MPI. Компонент Commutator в исполнителе осуществляет обмен сообщениями с координатором по технологии MPI. Компонент Executor на исполнителе организует параллельную обработку сегментов колоночных индексов, используя технологию OpenMP, и формирует фрагмент ТПВ. Компонент Executor на координаторе объединяет фрагменты ТПВ, вычисленные исполнителями в единую таблицу. Компоненты Commutator и Executor используют компонент Compressor для сжатия/распаковки данных. Рассмотрим, как КСОП выполняет основные операции над распределенными колоночными индексами, перечисленные в разделе 1.

При выполнении оператора *CreateColumnIndex* координатор добавляет информацию о структуре колоночного индекса в свой локальный словарь и отправляет исполнителям указание создать в своих локальных словарях дескриптор с информацией о новом колоночном индексе. При этом координатор определяет длину сегментного интервала и границы фрагментных интервалов. Эта информация сохраняется в дескрипторе колоночного индекса. Кроме этого, дескриптор включает в себя битовую шкалу сегментов, в которой значение 1 соответствует непустым сегментам, значение 0 — пустым. При начальном создании колоночного индекса все сегменты на всех узлах-исполнителях являются пустыми.

При выполнении оператора *Insert* координатору передается кортеж (SurrogateKey, Value) для вставки в указанный колоночный индекс. Координатор определяет, в границы какого фрагментного интервала попадает значение Value, и пересылает этот кортеж на соответствующий узел-исполнитель. Исполнитель определяет номер сегментного интервала, к которому принадлежит значение Value. Если соответствующий сегмент не

пуст, то кортеж вставляется в сегмент с сохранением упорядочения по полю Value. Если соответствующий сегмент пуст, то создается новый сегмент из одного кортежа.

При выполнении оператора *TransitiveInsert* координатору кроме нового кортежа (SurrogateKey, Value) передается дополнительное значение TValue, которое им используется для определения номера фрагментного интервала. Кортеж (SurrogateKey, Value) вместе со значением TValue пересылается на соответствующий узел-исполнитель. Исполнитель по значение TValue определяет номер сегментного интервала и вставляет новый кортеж в соответствующий сегмент.

При выполнении оператора *Delete* координатору передается суррогатный ключ SurrogateKey и значение Value, которые надо удалить. Координатор определяет, в границы какого фрагментного интервала попадает значение Value, и пересылает этот кортеж на соответствующий узел-исполнитель. Исполнитель определяет номер сегментного интервала, к которому принадлежит значение Value, производит в соответствующем сегменте поиск кортежа с ключом SurrogateKey и выполняет его удаление.

Оператор *TransitiveDelete* выполняется аналогично оператору *Delete* с той лишь разницей, что номера фрагментного и сегментного интервалов вычисляются по транзитивному значению TValue.

Выполнение оператора *Execute* включает в себя две фазы: 1) вычисление фрагментов ТПВ на узлах-исполнителях; 2) слияние фрагментов ТПВ в единую таблицу на узле-координаторе и пересылка ее на SQL-сервер. На первой фазе процессорные ядра (легковесные процессы OpenMP) выбирают необработанные сегментные интервалы и выполняют вычисление *сегментных* ТПВ для соответствующих сегментов колоночных индексов, задействованных в запросе. После того, как все сегментные интервалы обработаны, получившийся фрагмент ТПВ пересылается на узел-координатор в сжатом виде. На второй фазе координатор распаковывает все полученные сегментные ТПВ и объединяет их в единую таблицу. Для распараллеливания этого процесса используется технология OpenMP. Получившаяся ТПВ пересылается на SQL-сервер. При этом также может использоваться сжатие данных.

Колоночный сопроцессор КСОП был реализован на языке Си с использованием аппаратно независимых параллельных технологий MPI и OpenMP. Он может работать как на ЦПУ Intel Xeon X5680, так и на сопроцессоре Intel Xeon Phi без модификации кода. 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/.

5. Результаты экспериментов

В данном разделе приводятся результаты вычислительных экспериментов по исследованию эффективности колоночного сопроцессора КСОП.

5.1. Вычислительная среда

Эксперименты проводились на двух вычислительных комплексах с кластерной архитектурой: «Торнадо ЮУрГУ» Южно-Уральского государственного университета и «RSC PetaStream» Межведомственного суперкомпьютерного центра Российской академии наук. Основные параметры этих систем приведены в табл. 1.

Таблица 1

Параметры вычислительных комплексов

Параметры	**Вычислительный комплекс**	
	«Торнадо ЮУрГУ»	**«RSC PetaStream»**
Количество узлов	384	64
Тип процессоров	2 × Intel Xeon X5680 (12 ядер по 3.33 ГГц; 2 потока на ядро)	
Оперативная память узла	24 Гб	
Тип сопроцессора	Intel Xeon Phi SE10X (61 ядро по 1.1 ГГц; 4 потока на ядро)	Intel Xeon Phi 7120 (61 ядро по 1.24 ГГц)
Память сопроцессора	8 Гб	16 Гб
Тип системной сети	InfiniBand QDR (40 Гбит/с)	InfiniBand FDR (56 Гбит/с)
Тип управляющей сети	Gigabit Ethernet	Gigabit Ethernet
Операционная система	Linux CentOS 6.2	Linux CentOS 7.0

Для тестирования колоночного сопроцессора КСОП использовалась синтетическая база данных, построенная на основе эталонного теста TPC-H [35]. Тестовая база данных состояла из двух таблиц: **ORDERS** (ЗАКАЗЫ) и **CUSTOMER** (КЛИЕНТЫ), схема которых приведена на рис. 7. 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×63 000 000 кортежей, размер отношения **CUSTOMER** — SF×630 000 кортежей.

Для генерации тестовой базы данных была написана программа на языке Си, использующая методику, изложенную в работе [36]. Атрибуты **CUSTOMER.A** и **ORDERS.A**, игравшие роль суррогатных ключей, заполнялись целыми числами с шагом 1, начиная со значения 0. Атрибут **CUSTOMER.ID_CUSTOMER**, являющийся первичным ключом, заполнялся целыми числами с шагом 1, начиная со значения 1. Атрибут **ORDERS.ID_ORDER**, являющийся первичным ключом, также заполнялся целыми числами с шагом 1, начиная со значения 1. Атрибут **ORDERS.ID_CUSTOMER**, являющийся внешним ключом, заполнялся значениями из интервала [1, SF*630000]. При этом, для имитирования перекоса данных использовались следующие распределения:

- равномерное (uniform);
- «45-20»;
- «65-20»;
- «80-20».

Для генерации неравномерного распределения была использована вероятностная модель. В соответствии с этой моделью коэффициент перекоса θ, $(0 \le \theta \le 1)$ задает распределение, при котором каждому целому значению из интервала [1, SF*630000] назначается некоторый весовой коэффициент $p_i (i = 1, \ldots, N)$, определяемый формулой

$$p_i = \frac{1}{i^{\theta} \cdot H_N^{(\theta)}}, \sum_{i=1}^{N} p_i = 1,$$

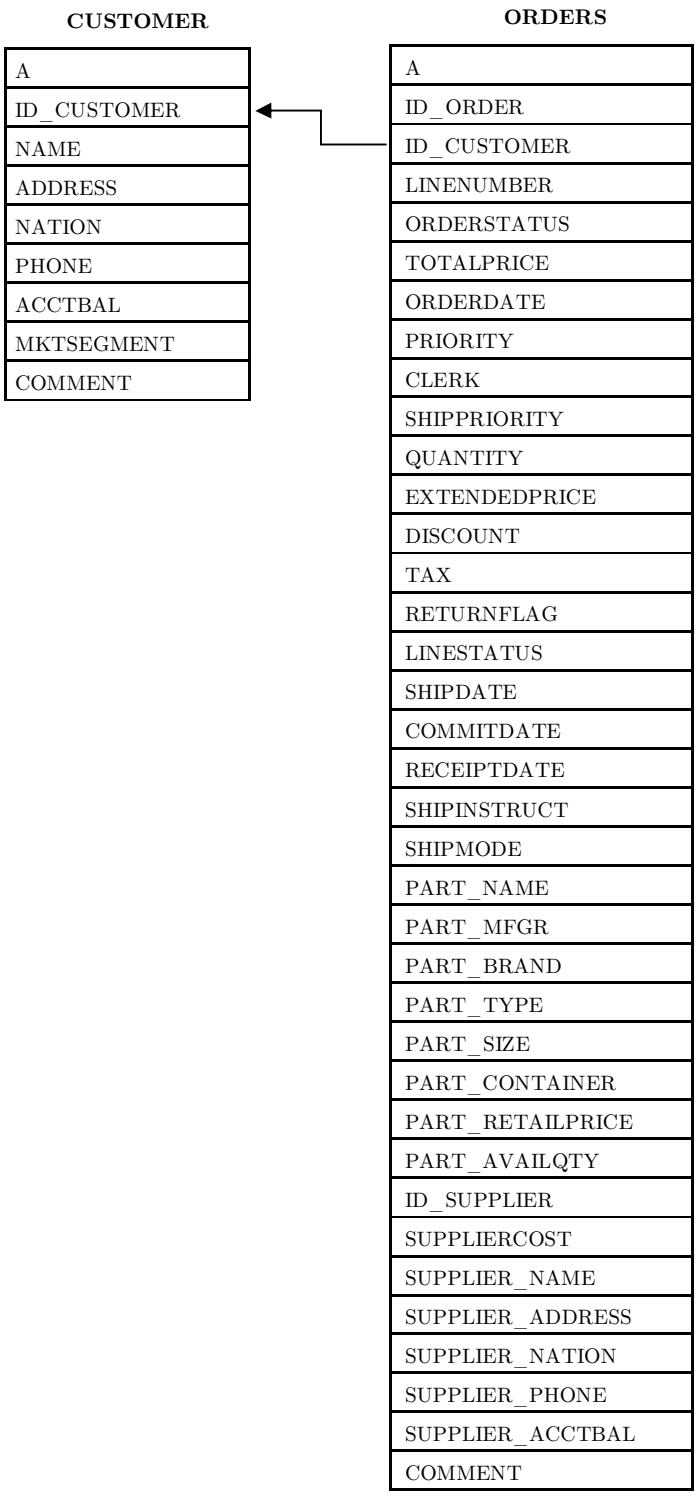


Рис. 7. Схема тестовой базы данных

где $N = \text{SF*}630000$ — количество различных значений атрибута **ORDERS.ID_CUSTOMER** и $H_N^{(s)} = 1^{-s} + 2^{-s} + \cdots + N^{-s}$, N-е гармоническое число порядка s. В случае $\theta = 0$ распределение весовых коэффициентов соответствует равномерному распределению. При $\theta = 0{,}86$ распределение соответствует правилу «80-20», в соответствии с которым 20 % самых популярных значений занимают 80 % позиций в столбце **ID_CUSTOMER** в таблице **ORDERS**. При $\theta = 0{,}73$ распределение соответ-

ствует правилу «65-20» (20 % самых популярных значений занимают 65 % позиций в столбце **ID_CUSTOMER** в таблице **ORDERS**). При $\theta = 0{,}5$ распределение соответствует правилу «45-20» (20 % самых популярных значений занимают 45 % позиций в столбце **ID_CUSTOMER** в таблице **ORDERS**).

Все остальные атрибуты в отношениях **CUSTOMER** и **ORDERS**, заполнялись случайными значениями соответствующих типов с равномерным распределением.

Тестовая база данных была развернута в СУБД PostgreSQL 9.4.0 на выделенном узле вычислительного кластера «Торнадо ЮУрГУ». В качестве тестового запроса фигурировал следующий SQL-запрос Q1:

```
# Запрос Q1
SELECT * FROM CUSTOMER, ORDERS
WHERE (CUSTOMER.ID_CUSTOMER=ORDERS.ID_CUSTOMER)
AND (ORDERS.TOTALPRICE <= Sel*100 000).
```

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**. Например, при Sel = 0,5 размер результирующего отношения составляет 50 % от размера отношения **ORDERS**, при Sel = 0,05 — 5 %, а при Sel = 1 — 100 %. Таким образом, большая селективность запроса соответствует меньшему значению коэффициента селективности.

С помощью колоночного сопроцессора КСОП в оперативной памяти кластерной вычислительной системы были созданы следующие распределенные колоночные индексы:

$I_{CUSTOMER.ID\_CUSTOMER}(A,\ ID\_CUSTOMER)$,

$I_{ORDERS.ID\_CUSTOMER}(A,\ ID\_CUSTOMER)$,

$I_{ORDER.TOTALPRICE}(A,\ TOTALPRICE)$.

Все индексы сортировались по значениям соответствующих атрибутов. Индексы $I_{CUSTOMER.ID\_CUSTOMER}$, $I_{ORDERS.ID\_CUSTOMER}$ фрагментировались и сегментировались на основе доменно-интервального принципа по домену [1, SF*630000], индекс $I_{ORDER.TOTALPRICE}$ фрагментировался и сегментировался транзитивно относительно индекса $I_{ORDERS.ID\_CUSTOMER}$. Все фрагментные и сегментные интервалы, на которые делился домен, имели одинаковую длину. Сегменты индексов сжимались с помощью библиотеки Zlib.

При выполнении запроса колоночный сопроцессор КСОП вычислял таблицу предварительных вычислений $P(A\_ORDERS,\ A\_CUSTOMER)$ следующим образом:

$$P = \pi_{I_{CUSTOMER.ID\_CUSTOMER}.A \to A\_CUSTOMER,\ I_{ORDERS.ID\_CUSTOMER}.A \to A\_ORDERS}\Big(I_{CUSTOMER.ID\_CUSTOMER} \underset{\boxed{\substack{I_{CUSTOMER.ID\_CUSTOMER}.ID\_CUSTOMER = \\ I_{ORDERS.ID\_CUSTOMER}.ID\_CUSTOMER}}}{\bowtie} \left(I_{ORDERS.ID\_CUSTOMER} \bowtie \sigma_{TOTALPRICE \le Sel \cdot 100\,000}\left(I_{ORDERS.TOTALPRICE}\right)\right)\Big).$$

Вычисления фрагментов таблицы P производились параллельно на доступных процессорных узлах без обменов данными. Затем полученные фрагменты пересылались на узел с PostgreSQL, где *координатор* осуществлял их слияние в общую таблицу P. Вместо выполнения запроса Q1 в PostgreSQL выполнялся следующий запрос Q2 с использованием таблицы предварительных вычислений P:

```
# Запрос Q2
SELECT * FROM
   CUSTOMER INNER JOIN (
      P INNER JOIN ORDERS ON (ORDERS.A = P.A_ORDERS)
   ) ON (CUSTOMER.A=P.A_CUSTOMER);
```

При этом в PostgreSQL для атрибутов CUSTOMER.A и ORDERS.A предварительно были созданы кластеризованные индексы в виде B-деревьев.

5.2. Балансировка загрузки процессорных ядер Xeon Phi

В разделе 2 было сказано, что балансировку загрузки процессорных узлов, на которых располагаются распределенные колоночные индексы, можно осуществлять путем сдвигов границ фрагментных интервалов соответствующих доменов. В отличие от этого все сегментные интервалы для каждого конкретного домена имеют одинаковую длину. В случае, если значения атрибута в столбце распределены неравномерно, сегменты соответствующего колоночного индекса внутри одного процессорного узла могут значительно отличаться по своим размерам. Потенциально это может привести к дисбалансу в загрузке процессорных ядер, так как каждый сегмент обрабатывается целиком на одном ядре. Однако, если число обрабатываемых сегментов значительно превышает количество процессорных ядер, дисбаланса загрузки не возникнет. Целью первого эксперимента было определить оптимальное соотношение между количеством сегментов и процессорных ядер в условиях перекоса в распределении данных. Использовались следующие параметры эксперимента:

- Вычислительная система: «Торнадо ЮУрГУ»;
- Количество задействованных узлов: 1;
- Используемый процессор: Xeon Phi;
- Количество задействованных ядер: 60;
- Количество нитей на ядро: 1;
- Коэффициент масштабирования базы данных: SF = 1;
- Коэффициент селективности: Sel=0,0005;
- Распределение значений в колонке ORDERS.ID_CUSTOMER: uniform, «45-20», «65-20», 80-20»;
- Операция: построение таблицы предварительных вычислений P.

Результаты эксперимента представлены на рис. 8. 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.

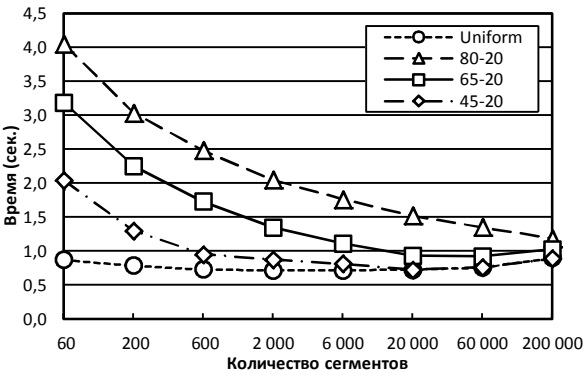


Рис. 8. Балансировка загрузки процессорных ядер сопроцессора Xeon Phi

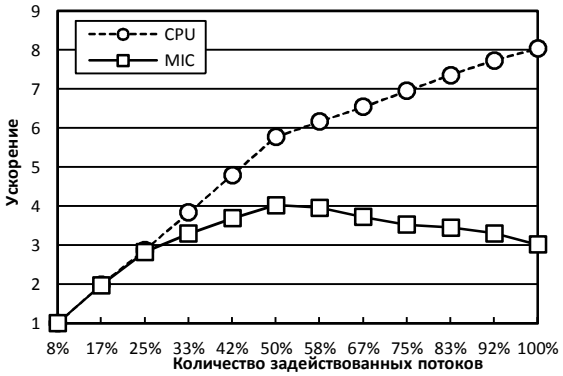


Рис. 9. Влияние гиперпоточности на ускорение

5.3. Влияние гиперпоточности

В большинстве случаев современные кластерные вычислительные системы оснащаются процессорами, поддерживающими технологию гиперпоточности (hyper-threading) [37], при включении которой каждое физическое ядро процессора определяется операционной системой как два или более логических ядра. У каждого логического ядра имеется свой набор регистров и контроллер прерываний. Остальные элементы физического ядра являются общими для всех логических ядер. При определенных рабочих нагрузках использование гиперпоточности позволяет увеличить производительность процессора.

ЦПУ Intel Xeon X5680 имеет аппаратную поддержку двух потоков на ядро, сопроцессор Intel Xeon Phi поддерживает четыре потока на ядро. Целью второго эксперимента было определить насколько эффективно гиперпоточность может быть применена при работе колоночного сопроцессора КСОП. Использовались следующие параметры эксперимента:

- Вычислительная система: «Торнадо ЮУрГУ»;
- Количество задействованных узлов: 1;
- Используемые процессорные устройства (ПУ):
  2 × Intel Xeon X5680 (CPU) / Intel Xeon Phi (MIC);
- Количество задействованных ядер: 12 / 60;
- Коэффициент масштабирования базы данных: SF = 1;
- Коэффициент селективности: Sel=0,0005;
- Распределение значений в колонке ORDERS.ID_CUSTOMER: uniform;
- Операция: построение таблицы предварительных вычислений P.

Эксперимент сначала проводился на 2 × Intel Xeon X5680 (CPU), а затем на Intel Xeon Phi (MIC). Варьировался процент задействованных потоков. Для 2 × Intel Xeon X5680 величина 100 % соответствует 24 потокам. Для Intel Xeon Phi величина 100 % соответствует 240 потокам. Ускорение вычислялось по формуле $t_{x\%}/t_{20\%}$, где $t_{x\%}$ — время выполнения операции по вычислению ТПВ на x % потоков, $t_{20\%}$ — на 20 % потоков. Результаты эксперимента представлены на рис. 9. 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 уско-

рения становиться 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.

5.4. Масштабируемость КСОП

В данном разделе исследуется масштабируемость колоночного сопроцессора КСОП на двух различных вычислительных системах и на базах данных двух разных масштабов. Масштабируемость является одной из важнейших характеристик при разработке параллельных СУБД для вычислительных систем с массовым параллелизмом. Основной численной характеристикой масштабируемости является ускорение, которое вычисляется по формуле t_x/t_k, где t_k — время выполнения запроса на некоторой базовой конфигурации, включающей k процессорных узлов, t_x — время выполнения запроса на конфигурации, включающей x процессорных узлов при $x \geq k$. При использовании колоночного сопроцессора КСОП общее время выполнения запроса вычисляется по следующей формуле:

$$t = t_{openMP} + t_{MPI} + t_{merge} + t_{SQL},$$

где t_{openMP} — максимальное время вычисления фрагментов ТПВ и их сжатие на отдельных процессорных узлах; t_{MPI} — время пересылки сжатых фрагментов ТПВ на узел-координатор; t_{merge} — время их распаковки и слияния в единую таблицу; t_{SQL} — время выполнения запроса с использованием ТПВ на SQL-сервере. В описываемой реализации роль SQL-сервера играла СУБД PostgeSQL, установленная на узле-координаторе. Поэтому время t_{SQL} не зависело от количества используемых процессорных узлов. Это время будет исследовано в разделе 5.5. В данном разделе мы исследуем время, вычисляемое по формуле:

$$t_{total} = t_{openMP} + t_{MPI} + t_{merge}.$$

Первый эксперимент на масштабируемость проводился на кластере «Торнадо ЮУрГУ» и имел следующие параметры:

- Вычислительная система: «Торнадо ЮУрГУ»;
- Количество задействованных узлов: 60–210;
- Используемые процессорные устройства (ПУ): 2 × Intel Xeon X5680;
- Использование гиперпоточности: не используется;
- Коэффициент масштабирования базы данных: SF = 1;
- Коэффициент селективности: 0,05–0,0005;
- Распределение значений в колонке ORDERS.ID_CUSTOMER: uniform;
- Операция: построение таблицы предварительных вычислений P.

Результаты эксперимента приведены на рис. 10. Графики ускорения вычисления ТПВ на рис. 10 показывают, что селективность запроса Sel оказывается фактором, ограничивающим масштабируемость. Так, при $Sel = 0{,}05$ масштабируемость ограничена 150 вычислительными узлами, при $Sel = 0{,}005$ — 180, а при $Sel = 0{,}0005$ превышает 210.

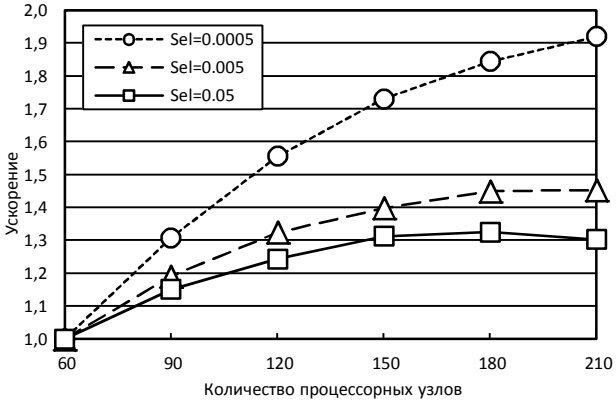


Рис. 10. Ускорение вычисления ТПВ на кластере «Торнадо ЮУрГУ» при SF = 1

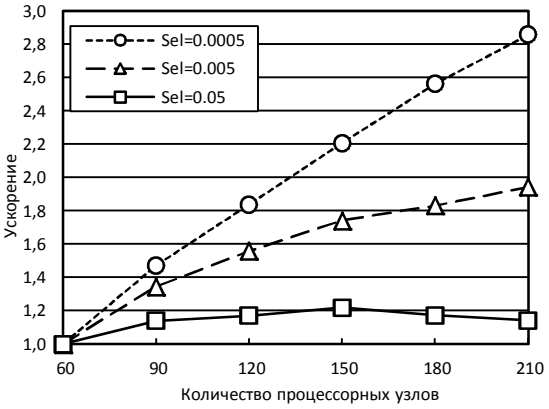
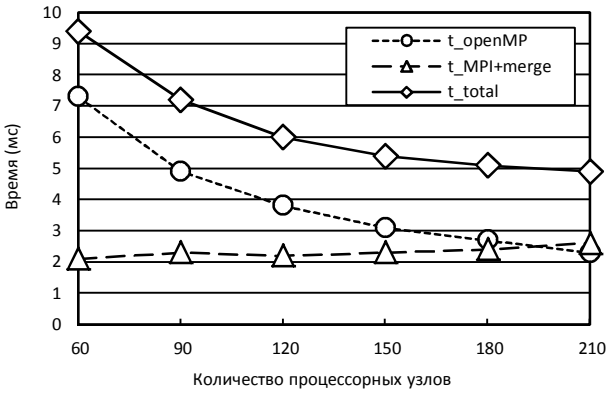
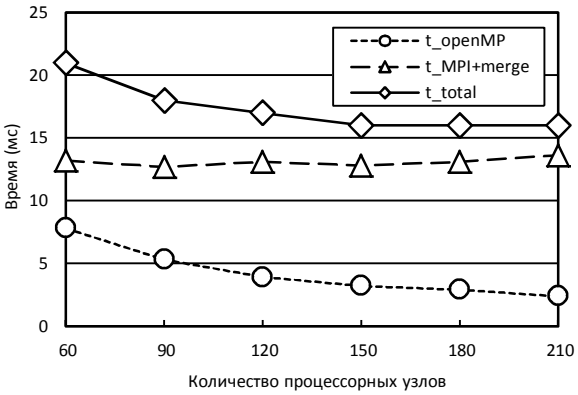


Рис. 11. Ускорение вычисления ТПВ на кластере «Торнадо ЮУрГУ» при SF = 10

Это связано с тем, что при увеличении селективности увеличивается размер ТПВ. При $Sel = 0.0005$ он составляет порядка 31 500 кортежей, при $Sel = 0.005$ — 315 000, а при $Sel = 0{,}05$ — 3 150 000 (значения даны для масштабного коэффициента SF = 1). На рис. 12 изображены зависимости выполнения времени подопераций от количества процессорных узлов. При малой селективности (рис. 12а) сумма времени пересылки фрагментов ТПВ, их сжатия, распаковки и слияния (t_MPI+merge) остается практически неизменной при увеличении процессорных узлов. Время вычисления фрагментов ТПВ на процессорных узлах (t_openMP) с ростом количества узлов уменьшается, и вплоть то 210 узлов превышает t_MPI+merge. Поэтому общее время (t_total) также уменьшается, что обеспечивает положительное ускорение. Однако при большой селективности (рис. 12б) t_MPI+merge, оставаясь практически неизменным, значительно превышает t_openMP, что препятствует существенному уменьшению общего времени при увеличении количества процессорных узлов.



а) Sel = 0,0005



б) Sel = 0,05

Рис. 12. Время выполнения подопераций при вычислении ТПВ на «Торнадо ЮУрГУ» при SF = 1

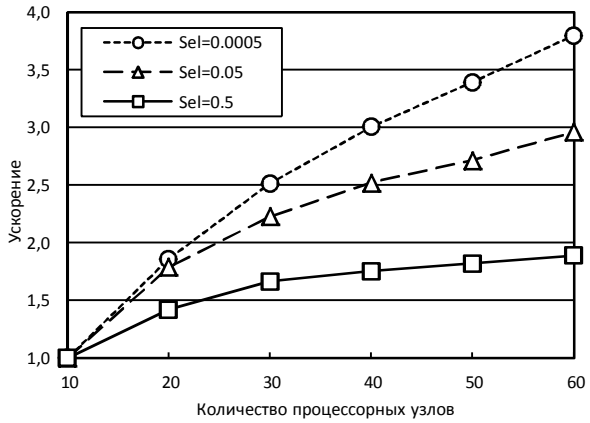


Рис. 13. Ускорение вычисления ТПВ на кластере «RSC PetaStream» при SF = 1

При увеличение коэффициента масштабируемости базы данных SF до значения 10 картина существенно меняется (см. рис. 11). На рис. 11 видно, что в этом случае кривая ускорения для $Sel = 0{,}0005$ становится практически линейной, а для $Sel = 0{,}005$ приближается к линейной. Однако, и в этом случае при большом значении коэффициента селективности $Sel = 0{,}05$ масштабируемость ограничивается 150 процессорными узлами. Причина та же самая: при $Sel = 0{,}05$ время передачи, распаковки и слияния ТПВ ($t_{MPI} + t_{merge}$) меняется мало и существенно превышает время ее вычисления t_{openMP}, в то время как при $Sel = 0{,}0005$ значение ($t_{MPI} + t_{merge}$) почти на порядок меньше t_{openMP}. В соответствие с этим можно сделать вывод, что при малой селективности запроса КСОП демонстрирует на больших базах данных ускорение, близкое к линейному.

При исследовании масштабируемости колоночного сопроцессора КСОП важным вопросом является ее зависимость от аппаратной архитектуры вычислительной системы. Как было сказано разделе 4, КСОП реализован с использованием аппаратно независимых параллельных технологий MPI и OpenMP. Поэтому он может работать как на ЦПУ Intel Xeon X5680, так и на сопроцессоре Intel Xeon Phi. Масштабируемость КСОП на вычислительном кластере «RSC PetaStream» была исследована в эксперименте, который имел следующие параметры:

- Вычислительная система: «RSC PetaStream»;
- Количество задействованных узлов: 10–60;
- Используемые процессорные устройства (ПУ): Intel Xeon Phi 7120;
- Использование гиперпоточности: не используется;
- Коэффициент масштабирования базы данных: SF = 1;
- Коэффициент селективности: 0,5–0,0005;
- Распределение значений в колонке ORDERS.ID_CUSTOMER: uniform;
- Операция: построение таблицы предварительных вычислений P.

Результаты эксперимента представлены на рис. 13. Они показывают, что масштабируемость КСОП и в этом случае приближается к линейной при уменьшении селективности запроса.

5.5. Использование КСОП при выполнении SQL-запросов

В заключительном эксперименте было исследовано в какой мере использование колоночного сопроцессора КСОП может ускорить выполнение запроса класса OLAP в реляционной СУБД. Эксперимент проводился при следующих параметрах:

- Вычислительная система: «Торнадо ЮУрГУ»;
- Конфигурация PostgreSQL: 1 процессорный узел с SSD Intel;
- Конфигурация КСОП: 210 процессорных узлов;
- Используемые процессорные устройства (ПУ): 2 × Intel Xeon X5680;
- Использование гиперпоточности: не используется;
- Коэффициент масштабирования базы данных: SF = 1;
- Коэффициент селективности: 0,05–0,0005;
- Распределение значений в колонке ORDERS.ID_CUSTOMER: uniform;
- Операция: выполнение SQL-запроса.

В ходе выполнения эксперимента исследовались следующие три конфигурации (см. табл. 2):

- PostgreSQL: выполнение запроса Q1 (см. раздел 5.1) без создания индексных файлов в виде B-деревьев;
- PostgreSQL & B-Trees: выполнение запроса Q1 (см. раздел 5.1) с предварительным созданием индексных файлов в виде B-деревьев для атрибутов CUSTOMER.ID_CUSTOMER и ORDERS.ID_CUSTOMER;
- PostgreSQL & CCOP: выполнение запроса Q2 (см. раздел 5.1) с использованием ТПВ P и предварительным созданием индексных файлов в виде B-деревьев для атрибутов CUSTOMER.A и ORDERS.A.

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.

Таблица 2

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

Конфигурация	Время (мин)					
	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-Trees	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

Эксперименты показали, 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).

Заключение

В статье описана общая архитектура организации системы баз данных с использованием колоночного сопроцессора КСОП. В состав системы входит SQL-сервер и вычислительный кластер. На SQL-сервере устанавливается реляционная СУБД с внедренным коннектором и драйвер КСОП. На вычислительном кластере устанавливается колоночный сопроцессор КСОП. Все колоночные индексы создаются и поддерживаются в распределенной оперативной памяти вычислительного кластера. При создании информационных систем с такой конфигурацией необходимо будет еще подключить подсистему восстановления колоночных индексов после сбоя. Для этого копии колоночных индексов и метаданных могут создаваться на твердотельных дисках, установленных на каждом вычислительном узле. Приведены результаты экспериментов над синтетической базой данных большого размера, исследующие эффективность колоночного сопроцессора КСОП. Эксперименты показали, что подходы и методы параллельного выполнения запросов класса OLAP на базе доменно-колоночной модели, демонстрируют хорошую масштабируемость (до нескольких сотен процессорных узлов и десятков тысяч процессорных ядер) для запросов с большой селективностью, которые являются типичными для хранилищ данных. Использование колоночного сопроцессора КСОП во взаимодействии с СУБД PostgreSQL позволило повысить эффективность выполнения таких запросов более чем на два порядка. Однако, эффективность использования КСОП снижается при уменьшении размеров базы данных и при увеличении размеров результирующего отношения.

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 ними.
- Обобщение описанных подходов и методов на многомерные данные.

Работа выполнена при финансовой поддержке Минобрнауки РФ в рамках ФЦП «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2014—2020 годы» (Госконтракт № 14.574.21.0035).

Литература

1. Чернышев, Г.А. Организация физического уровня колоночных СУБД / Г.А Чернышев // Труды СПИИРАН. — 2013. — № 7. Вып. 30. — С. 204–222.
2. Abadi, D.J. The Design and Implementation of Modern Column-Oriented Database Systems / D.J. Abadi, P.A. Boncz, S. Harizopoulos, S. Idreos, S. Madden // Foundations

and Trends in Databases. — 2013. — Vol. 5, No. 3. — P. 197–280. DOI: 10.1561/1900000024.

3. Idreos, S. MonetDB: Two Decades of Research in Column-oriented Database Architectures / S. Idreos, F. Groffen, N. Nes, S. Manegold, S. Mullender, M.L. Kersten // IEEE Data Engineering Bulletin. — 2012. — Vol. 35, No. 1. — P. 40–45.
4. Boncz, P.A. MonetDB/X100: Hyper-pipelining query execution / P.A. Boncz, M. Zukowski, N. Nes // Proceedings of the Second Biennial Conference on Innovative Data Systems Research (CIDR), January 4–7, Asilomar, CA, USA. — 2005. — P. 225–237.
5. Stonebraker, M. C-Store: A Column-Oriented DBMS / M. Stonebraker, D.J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S.R. Madden, E.J. O'Neil, P.E. O'Neil, A. Rasin, N. Tran, S.B. Zdonik // Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05), August 30 – September 2, 2005, Trondheim, Norway. — ACM, 2005. — P. 553–564.
6. MacNicol, R. Sybase IQ multiplex — designed for analytics / R. MacNicol, B. French // Proceedings of the Thirtieth International Conference on Very Large Data Bases, August 31–September 3, 2004, Toronto, Canada. — Morgan Kaufmann, 2004. — P. 1227–1230. DOI: 10.1016/b978-012088469-8/50111-x.
7. Zukowski, M. Vectorwise: Beyond column stores / M. Zukowski, P.A. Boncz // IEEE Data Engineering Bulletin. — 2012. — Vol. 35, No. 1. — P. 21–27.
8. Lamb, A. The Vertica analytic database: C-store 7 years later / A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandier, L. Doshi, C. Bear // Proceedings of the VLDB Endowment. — 2012. — Vol. 5, No. 12. — P. 1790–1801. DOI: 10.14778/2367502.2367518.
9. Barber, R. Business Analytics in (a) Blink / R. Barber, P. Bendel, M. Czech, O. Draese, F. Ho, N. Hrle, S. Idreos, M.-S. Kim, O. Koeth, J.-G. Lee, T.T. Li, G.M. Lohman, K. Morfonios, R. Müller, K. Murthy, I. Pandis, L. Qiao, V. Raman, R. Sidle, K. Stolze, S. Szabo // IEEE Data Engineering Bulletin. — 2012. — Vol. 35, No. 1. — P. 9–14.
10. Larson, P.-A. Enhancements to SQL server column stores / P.-A. Larson, C. Clinciu, C. Fraser, E.N. Hanson, M. Mokhtar, M. Nowakiewicz, V. Papadimos, S.L. Price, S. Rangarajan, R. Rusanu, M. Saubhasik // Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD'13), June 22–27, 2013, New York, NY, USA. — ACM, 2013. — P. 1159–1168. DOI: 10.1145/2463676.2463708.
11. Larson, P.-A. SQL server column store indexes / P.-A. Larson, C. Clinciu, E.N. Hanson, A. Oks, S.L. Price, S. Rangarajan, A. Surna, Q. Zhou // Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (SIGMOD'11), June 12–16, 2011, Athens, Greece. — ACM, 2011. — P. 1177–1184. DOI: 10.1145/1989323.1989448.
12. Larson, P.-A. Columnar Storage in SQL Server 2012 / P.-A. Larson, E.N. Hanson, S.L. Price // IEEE Data Engineering Bulletin. — 2012. — Vol. 35, No. 1. — P. 15–20.
13. Färber, F. The SAP HANA Database — An Architecture Overview / F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, J. Dees // IEEE Data Engineering Bulletin. — 2012. — Vol. 35, No. 1. — P. 28–33.
14. Weiss, R. A Technical Overview of the Oracle Exadata Database Machine and Exadata Storage Server — White Paper. — Oracle Corporation. — 2012. — 35 p. / R.A. Weiss. URL: http://www.oracle.com/technetwork/database/exadata/exadata-technical-whitepaper-134575.pdf (дата обращения: 29.10.2015).

15. A Drill-Down into EXASolution. — Technical Whitepaper. — EXASOL AG. — 2014. — 15 p. URL: http://info.exasol.com/whitepaper-exasolution-2-en.html (дата обращения: 22.10.2015).
16. A Peek under the Hood. — Technical Whitepaper. — EXASOL AG. — 2014. — 16 p. URL: http://www.breos.com/sites/default/files/pdf/downloads/exasol_whitepaper.pdf (дата обращения: 22.10.2015).
17. EXASolution. — Business Whitepaper. — EXASOL AG. — 2015. — 11 p. URL: http://info.exasol.com/business-whitepaper-exasolution-en.html (дата обращения: 27.10.2015).
18. Actian SQL Analytics in Hadoop. — A Technical Overview. — Actian Corporation. — 2015. — 16 p. URL: http://bigdata.actian.com/SQLAnalyticsinHadoop (дата обращения: 27.10.2015).
19. Ślęzak, D. Towards approximate SQL: infobright's approach / D. Ślęzak, M. Kowalski // Proceedings of the 7th international conference on Rough sets and current trends in computing (RSCTC'10). — Springer-Verlag, 2010. — P. 630–639. DOI: 10.1007/978-3-642-13529-3_67.
20. SAND CDBMS: A Technological Overview. — White Paper. — SAND Technology. — 2010. — 16 p. URL: http://www.sand.com/downloads/side2239eadd/wp_sand_cdbms_technological_overview_en.pdf (дата обращения: 29.10.2015).
21. Abadi, D.J. Column-Stores vs. Row-Stores: How Different Are They Really? / D.J. Abadi, S.R. Madden, N. Hachem // Proceedings of the 2008 ACM SIGMOD international conference on Management of data, June 9–12, 2008, Vancouver, BC, Canada. — ACM, 2008. — P. 967–980. DOI: 10.1145/1376616.1376712.
22. Ivanova, E. Decomposition of Natural Join Based on Domain-Interval Fragmented Column Indices / E. Ivanova, L. Sokolinsky // Proceedings of the 38th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO, May 25–29, 2015, Opatija, Croatia. — IEEE, 2015. — P. 223–226. DOI: 10.1109/mipro.2015.7160266.
23. Иванова, Е.В. Декомпозиция операции группировки на базе распределенных колоночных индексов / Е.В. Иванова, Л.Б. Соколинский // Наука ЮУрГУ. — Челябинск: Издательский центр ЮУрГУ, 2015. — С. 15–23.
24. Иванова, Е.В. Декомпозиция операций пересечения и соединения на основе доменно-интервальной фрагментации колоночных индексов / Е.В. Иванова, Л.Б. Соколинский // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. — 2015. — Т. 4, № 1. — С. 44–56. DOI: 10.14529/cmse150104.
25. Иванова, Е.В. Использование распределенных колоночных хеш-индексов для обработки запросов к сверхбольшим базам данных / Е.В. Иванова // Научный сервис в сети Интернет: многообразие суперкомпьютерных миров: Труды Международной суперкомпьютерной конференции (22–27 сентября 2014 г., Новороссийск). — М.: Изд-во МГУ, 2014. — С. 102–104.
26. Huffman, D. A method for the construction of minimum-redundancy codes / D. Huffman // Proceedings of the I.R.E. — 1952. — Vol. 40, No. 9. — P. 1098–1101. DOI: 10.1109/jrproc.1952.273898.
27. Ziv, J. A universal algorithm for sequential data compression / J. Ziv, A. Lempel // IEEE Transactions on Information Theory. — 1977. — Vol. 23, No. 3. — P. 337–343. DOI: 10.1109/tit.1977.1055714.

28. Abadi, D.J. Integrating compression and execution in column-oriented database systems / D.J. Abadi, S.R. Madden, M. Ferreira // Proceedings of the 2006 ACM SIGMOD international conference on Management of data, June 26–29, 2006, Chicago, Illinois. — ACM, 2006. — P. 671–682. DOI: 10.1145/1142473.1142548.
29. Bassiouni, M.A. Data Compression in Scientific and Statistical Databases / M.A. Bassiouni // IEEE Transactions on Software Engineering. — 1985. — Vol. 11, No. 10. — P. 1047–1058. DOI: 10.1109/tse.1985.231852.
30. Ruth, S.S. Data Compression for Large Business Files / S.S. Ruth, P.J. Kreutzer // Datamation. — 1972. — Vol. 19, No. 9. — P. 62–66.
31. Roth, M.A. Database compression / M.A. Roth, S.J. Van Horn // ACM SIGMOD Record. — 1993. — Vol. 22, No. 3. — P. 31–39. DOI: 10.1145/163090.163096.
32. Deutsch, P. ZLIB Compressed Data Format Specification version 3.3 / P. Deutsch, J.-L. Gailly. — United States: RFC Editor. — 1996. DOI: 10.17487/rfc1950.
33. Roelofs, G. Zlib: A Massively Spiffy Yet Delicately Unobtrusive Compression Library / G. Roelofs, J. Gailly, M. Adler. URL: http://www.zlib.net/ (дата обращения: 20.09.2015).
34. Deutsch, P. DEFLATE Compressed Data Format Specification version 1.3 / P. Deutsch. — United States: RFC Editor. — 1996. DOI: 10.17487/rfc1951.
35. TPC Benchmark H — Standard Specification, Version 2.17.1. — Transaction Processing Performance Council (http://www.tpc.org). — 2014. — 136 p. URL: http://www.tpc.org/
tpc_documents_current_versions/pdf/tpch2.17.1.pdf (дата обращения: 29.10.2015).
36. Gray, J. Quickly generating billion-record synthetic databases / J. Gray, P. Sundaresan, S. Englert, K. Baclawski, P.J. Weinberger // Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, May 24–27, 1994, Minneapolis, Minnesota. — ACM Press, 1994. — P. 243–252. DOI: 10.1145/191843.191886.
37. Ungerer, T. A survey of processors with explicit multithreading / T. Ungerer, B. Robič, J. Šilc // ACM Computing Surveys. — 2003. — Vol. 35, No. 1. — P. 29–63. DOI: 10.1145/641865.641867.

Иванова Елена Владимировна, программист отдела поддержки и обучения пользователей Лаборатории суперкомпьютерного моделирования, Южно-Уральский государственный университет (Челябинск, Российская Федерация), Elena.Ivanova@susu.ru.

Соколинский Леонид Борисович, д. ф.-м. н., профессор, проректор по информатизации, Южно-Уральский государственный университет (Челябинск, Российская Федерация), Leonid.Sokolinsky@susu.ru.

Поступила в редакцию 10 сентября 2015 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2015, vol. 4, no. 4, pp. 5–31

DOI: 10.14529/cmse150401

COLUMNAR DATABASE COPROCESSOR FOR COMPUTING CLUSTER SYSTEM

E.V. Ivanova, South Ural State University (Chelyabinsk, Russian Federation) Elena.Ivanova@susu.ru,

L.B. Sokolinsky, South Ural State University (Chelyabinsk, Russian Federation) Leonid.Sokolinsky@susu.ru

The paper is devoted to the design and implementation issues of columnar coprocessor for RDBMS. Columnar coprocessor (CCOP) is developed on the base of columnar data storage model. It is designed for large computing cluster systems. CCOP can utilize CPUs as well as manycore coprocessors MIC. CCOP maintains the columnar indices with surrogate keys stored in distributed main memory. The partitioning is performed on the base of domain-interval model. For the data warehouse workload, CCOP demonstrates performance much higher than row-stores do.

Keywords: columnar coprocessor, CCOP, distributed columnar indices, domain-interval fragmentation, computing cluster systems, manycore coprocessors, MIC architecture.

References

1. Chernyshev G.A Organizaciya fizicheskogo urovnya kolonochnyh SUBD [Physical Layer Organization of Columnar DBMS] // Trudy SPIIRAN [Proceedings of the SPIIRAS]. 2013. Vol. 7. No. 30. P. 204–222.
2. 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. DOI: 10.1561/1900000024.
3. 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.
4. 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), January 4–7, Asilomar, CA, USA. 2005. P. 225–237.
5. 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), August 30 – September 2, 2005, Trondheim, Norway. ACM, 2005. P. 553–564.
6. MacNicol R., French B. Sybase IQ multiplex — designed for analytics // Proceedings of the Thirtieth International Conference on Very Large Data Bases, August 31 – September 3, 2004, Toronto, Canada. Morgan Kaufmann, 2004. P. 1227–1230. DOI: 10.1016/b978-012088469-8/50111-x.

7. Zukowski M., Boncz P.A. Vectorwise: Beyond column stores // IEEE Data Engineering Bulletin. 2012. Vol. 35, No. 1. P. 21–27.
8. Lamb A., Fuller M., Varadarajan R., Tran N., Vandier B., Doshi L., Bear C. The Vertica analytic database: C-store 7 years later // Proceedings of the VLDB Endowment. 2012. Vol. 5, No. 12. P. 1790–1801. DOI: 10.14778/2367502.2367518.
9. Barber R., Bendel P., Czech M., Draese O., Ho F., Hrle N., Idreos S., Kim M.-S., Koeth O., Lee J.-G., Li T.T., Lohman G. M., Morfonios K., Müller R., Murthy K., Pandis I., Qiao L., Raman V., Sidle R., Stolze K., Szabo S. Business Analytics in (a) Blink // IEEE Data Engineering Bulletin. 2012. Vol. 35, No. 1. P. 9–14.
10. Larson P.-A., Clinciu C., Fraser C., Hanson E. N., Mokhtar M., Nowakiewicz M., Papadimos V., Price S. L., Rangarajan S., Rusanu R., Saubhasik M. Enhancements to SQL server column stores // Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD'13), June 22–27, 2013, New York, NY, USA. ACM, 2013. P. 1159–1168. DOI: 10.1145/2463676.2463708.
11. 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), June 12–16, 2011, Athens, Greece. ACM, 2011. P. 1177–1184. DOI: 10.1145/1989323.1989448.
12. Larson P.-A., Hanson E. N., Price S. L. Columnar Storage in SQL Server 2012 // IEEE Data Engineering Bulletin. 2012. Vol. 35, No. 1. P. 15–20.
13. Färber F., May N., Lehner W., Große P., Müller I., Rauhe H., Dees J. The SAP HANA Database – An Architecture Overview // IEEE Data Engineering Bulletin. 2012. Vol. 35, No. 1. P. 28–33.
14. Weiss R. A Technical Overview of the Oracle Exadata Database Machine and Exadata Storage Server. Oracle Corporation White Paper, 2012. 35 p. URL: http://www.oracle.com/technetwork/database/exadata/exadata-technical-whitepaper-134575.pdf (accessed: 29.10.2015).
15. A Drill-Down into EXASolution. Technical Whitepaper. EXASOL AG, 2014. 15 p. URL: http://info.exasol.com/whitepaper-exasolution-2-en.html (accessed: 22.10.2015).
16. A Peek under the Hood. Technical Whitepaper. EXASOL AG, 2014. 16 p. URL: http://www.breos.com/sites/default/files/pdf/downloads/exasol_whitepaper.pdf (accessed: 22.10.2015).
17. EXASolution. Business Whitepaper. EXASOL AG, 2015. 11 p. URL: http://info.exasol.com/business-whitepaper-exasolution-en.html (accessed: 27.10.2015).
18. Actian SQL Analytics in Hadoop. A Technical Overview. Actian Corporation, 2015. 16 p. URL: http://bigdata.actian.com/SQLAnalyticsinHadoop (accessed: 27.10.2015).
19. Ślęzak D., Kowalski M. Towards approximate SQL: infobright's approach // Proceedings of the 7th international conference on Rough sets and current trends in computing (RSCTC'10). Springer-Verlag, 2010. P. 630–639. DOI: 10.1007/978-3-642-13529-3_67.
20. SAND CDBMS: A Technological Overview. White Paper. SAND Technology, 2010. 16 p. URL: http://www.sand.com/downloads/side2239eadd/wp_sand_cdbms_technological_overview_en.pdf (accessed: 29.10.2015).
21. 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, June 9–12, 2008, Vancouver, BC, Canada. ACM, 2008. P. 967–980. DOI: 10.1145/1376616.1376712.

22. Ivanova E., Sokolinsky L. Decomposition of Natural Join Based on Domain-Interval Fragmented Column Indices // Proceedings of the 38th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO, May 25–29, 2015, Opatija, Croatia. IEEE, 2015. P. 223–226. DOI: 10.1109/mipro.2015.7160266.
23. Ivanova E.V., Sokolinsky L.B. Dekompoziciya operacii gruppirovki na baze raspredelennyh kolonochnyh indeksov [Decomposition of Grouping Operation Based on Fragmented Column Indices] // Nauka YUUrGU [Science of SUSU]. Chelyabinsk: SUSU publishing center, 2015. P. 15–23.
24. Ivanova E.V., Sokolinsky L.B. Dekompoziciya operacij peresecheniya i soedineniya na osnove domenno-interval'noj fragmentacii kolonochnyh indeksov [Decomposition of Intersection and Join Operations Based on Domain-Interval Fragmented Column Indices] // Vestnik Yuzhno-Ural'skogo gosudarstvennogo universiteta. Seriya «Vychislitel'naya matematika i informatika» [Bulletin of South Ural State University. Series: Computational Mathematics and Software Engineering]. 2015. Vol. 4, No. 1. P. 44–56. DOI: 10.14529/cmse150104.
25. Ivanova E.V. Ispol'zovanie raspredelennyh kolonochnyh hesh-indeksov dlya obrabotki zaprosov k sverhbol'shim bazam dannyh [Using Distributed Column Hash Indices for Query Execution for Very Large Databases] // Nauchnyj servis v seti Internet: mnogoobrazie superkomp'yuternyh mirov: Trudy Mezhdunarodnoj superkomp'yuternoj konferencii [Scientific service in Internet: The variety of supercomputing worlds: Proceedings of the International Supercomputer Conference], September 22–27, 2014, Novorossiysk, Russia. Moscow: MSU publishing center, 2014. P. 102–104.
26. Huffman D. A method for the construction of minimum-redundancy codes // Proceedings of the I.R.E. 1952.Vol. 40, No. 9. P. 1098–1101. DOI: 10.1109/jrproc.1952.273898.
27. Ziv J., Lempel A., A universal algorithm for sequential data compression // IEEE Transactions on Information Theory. 1977. Vol. 23, No. 3. P. 337–343. DOI: 10.1109/tit.1977.1055714.
28. 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, June 26–29, 2006, Chicago, Illinois. ACM, 2006. P. 671–682. DOI: 10.1145/1142473.1142548.
29. Bassiouni M. A. Data Compression in Scientific and Statistical Databases // IEEE Transactions on Software Engineering. 1985. Vol. 11, No. 10. P. 1047–1058. DOI: 10.1109/tse.1985.231852.
30. Ruth S.S., Kreutzer P.J. Data Compression for Large Business Files // Datamation. 1972. Vol. 19, No. 9. P. 62–66.
31. Roth M. A., Van Horn S. J. Database compression // ACM SIGMOD Record. 1993. Vol. 22, No. 3. P. 31–39. DOI: 10.1145/163090.163096.
32. Deutsch P., Gailly J.-L. ZLIB Compressed Data Format Specification version 3.3. United States: RFC Editor, 1996. DOI: 10.17487/rfc1950.
33. Roelofs G., Gailly J., Adler M. Zlib: A Massively Spiffy Yet Delicately Unobtrusive Compression Library. URL: http://www.zlib.net/ (accessed: 20.09.2015).
34. Deutsch P. DEFLATE Compressed Data Format Specification version 1.3. United States: RFC Editor, 1996. DOI: 10.17487/rfc1951.

35. TPC Benchmark H — Standard Specification, Version 2.17.1. Transaction Processing Performance Council (http://www.tpc.org), 2014. 136 p. URL: http://www.tpc.org/tpc_documents_current_versions/pdf/tpch2.17.1.pdf (accessed: 29.10.2015).
36. 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, May 24–27, 1994, Minneapolis, Minnesota. ACM Press, 1994. P. 243–252. DOI: 10.1145/191843.191886.
37. Ungerer T., Robič B., Šilc J. A survey of processors with explicit multithreading // ACM Computing Surveys. 2003. Vol. 35, No. 1. P. 29–63. DOI: 10.1145/641865.641867.

Received September 10, 2015.