УДК 004.657

© 2012 г. **Ю.А. Григорьев,** д-р техн. наук,
**Е.Ю. Ермаков**
(Московский государственный технический университет им. Н.Э. Баумана)

# МОДЕЛЬ ОБРАБОТКИ ЗАПРОСОВ В ПАРАЛЛЕЛЬНОЙ КОЛОНОЧНОЙ СИСТЕМЕ БАЗ ДАННЫХ

В статье анализируются существующие методы выполнения запросов в параллельной колоночной СУБД. Приводится преобразование Лапласа-Стилтьеса (ПЛС) времени выполнения запроса в параллельной колоночной СУБД и рассматриваются варианты этого преобразования для различных архитектур параллельных систем баз данных. Выполнено сравнение времени обработки запроса с планом $\pi_A(\sigma_F(R))$ в параллельной построчной и колоночной СУБД.

***Ключевые слова***. Параллельные колоночные базы данных, преобразование Лапласа-Стилтьеса, математическое ожидание времени выполнения запроса.

## Введение

Российский бизнес все острее осознает необходимость построения хранилищ данных. Являясь одними из наиболее значимых элементов ИТ-инфраструктуры предприятия, хранилища консолидируют информацию, необходимую для создания достоверных аналитических отчетов. Хранилища данных являются одними из крупнейших источников информации для современных аналитиков. И, по оценке Gartner, в ближайшей перспективе они останутся одними из ключевых компонентов автоматизированных информационных систем предприятий [1].

Большой потенциал колоночных систем при построении хранилищ данных подтверждают аналитические исследования и прогнозы аналитиков [1 – 3], которые видят колоночные СУБД одним из основных и перспективных направлений развития. Например, в работе [3] показано 200-кратное сокращение объема ввода-вывода по сравнению с аналогичной реляционной СУБД (РСУБД). Это достигается за счет того, что из базы данных читается не вся запись, а только те атрибуты, которые участвуют в запросе. Здесь также применяются эффективные методы сжатия столбцов.

Таким образом, перед проектировщиком системы обработки данных возникает непростая задача выбора между традиционными (строчными – Oracle, MS SQL Server и др.) и специализированными (колоночными – Vertica, ParAccel и др.) СУБД. Для принятия обоснованного технического решения по выбору типа СУБД необходимо использовать средства моделирования. Для традиционных

РСУБД такие методы уже существуют [4]. Для параллельных СУБД подобные исследования ведутся [5 – 8], но находятся на начальной стадии развития.

В работе предлагаются математические методы оценки времени выполнения запроса в параллельных колоночных системах баз данных, учитывающие особенности выполнения запросов к базе данных проектируемой системы, а также особенности реализации колоночных хранилищ.

## Организация работы колоночного хранилища

Под строчным хранением данных обычно понимается физическое хранение кортежа любого отношения в виде одной записи, в котором значения атрибута идут последовательно одно за другим, а за последним атрибутом кортежа в общем случае следует новый кортеж отношения.

Таким образом, на физическом носителе отношение R представлено в следующем виде

$$[\dot{a}_{11}, \dot{a}_{21}, \dots, \dot{a}_{n1}]_1 [\dot{a}_{12}, \dot{a}_{22}, \dots, \dot{a}_{n2}]_2 [\dot{a}_{13}, \dot{a}_{23}, \dots, \dot{a}_{n3}]_3 \dots [\dot{a}_{1m}, \dot{a}_{2m}, \dots, \dot{a}_{nm}]_m ,$$

где $\dot{a}_{ij}$ – значение атрибута $a_i$ в $j$-м кортеже отношения $R$; $[\dot{a}_{1j}, \dot{a}_{2j}, \dots, \dot{a}_{nj}]_j$ – $j$-й кортеж отношения $R$; $n$ – количество атрибутов отношения $R$; $m = T(R)$ – количество кортежей отношения $R$.

В колоночных хранилищах значения одного атрибута хранятся последовательно друг за другом [9], т.е. на физическом носителе отношение R примет следующий вид:

$$\langle \dot{a}_{11}, \dot{a}_{12}, \dot{a}_{13}, \dots, \dot{a}_{1m} \rangle_1 \langle \dot{a}_{21}, \dot{a}_{22}, \dot{a}_{23}, \dots, \dot{a}_{2m} \rangle_2 \dots \langle \dot{a}_{n1}, \dot{a}_{n2}, \dot{a}_{n3}, \dots, \dot{a}_{nm} \rangle_n ,$$

где $\dot{a}_{ij}$ – значение атрибута $a_i$ в $j$-м кортеже отношения $R$; $\langle \dot{a}_{i1}, \dot{a}_{i2}, \dot{a}_{i3}, \dots, \dot{a}_{im} \rangle_i$ – $i$-й столбец (атрибут) отношения $R$.

Каждая колонка, хранимая на диске, разделена на блоки определенного размера ($S_b$). Блок состоит из заголовка, размер которого пренебрежительно мал по сравнению с размером блока и непосредственно данных. При одном запросе к диску происходит чтение нескольких блоков, количество которых определяется параметром.

Каждой записи в столбце соответствует ее позиция (номер строки). В большинстве современных колоночных БД [10] значения столбца упорядочиваются по их позициям.

На логическом уровне колоночное хранилище идентично строчному: оно, по существу, представляет собой только модификацию физической (дисковой) структуры базы данных. В общем случае колоночные БД могут реализовывать совместимый со стандартами интерфейс реляционной базы данных (например, ODBC, JDBC и т.д.). Основные на данном уровне различия заключаются в информационных процессах, протекающих при формировании плана запроса и в процессе его выполнения.

В строчных хранилищах план запроса представляет собой дерево, у каждого узла которого имеются один родитель и один (или два в случае пересечения) дочерних узла [11]. Реализация исполнителя планов базируется на следующих трех базовых парадигмах [11]: синхронный конвейер; итераторная модель; ско-

бочный шаблон.

Рассмотрим, какие изменения вносит колоночные базы данных в каждый из этих принципов.

*Синхронный конвейер*. Согласно [9], стандартным приемом, применяемым в хранилищах, является организация между операциями в дереве плана запроса так называемого синхронного конвейера для передачи кортежей промежуточных отношений. Суть данного метода состоит в том, что как только операция получает очередной кортеж своего результирующего отношения она немедленно передает его по конвейеру стоящей выше операции для обработки.

В колоночных хранилищах при реализации конвейера учитываются следующие особенности:

в связи с фундаментальными отличиями в типе хранения информации на физическом носителе операции выполняются не над кортежами отношения, а над блоками атрибутов отношения;

существует возможность проводить операции не над данными (блоками), а над позициями значений в этих блоках;

между узлами конвейера могут передаваться как позиции, так и указатели на блоки данных.

Пример конвейера колоночного хранилища для простого запроса по двум атрибутам таблицы, использующего вышеперечисленные возможности, представлен на рис. 1. Результатами операций 5 и 6 являются позиции значений в атрибутах. Списки полученных позиций передаются и обрабатываются (соединяются, пересекаются и т.п.) в операторе 4, результат действия которого попадает в операторы 2 и 3, считывающие указанные в полученном наборе позиций значения.

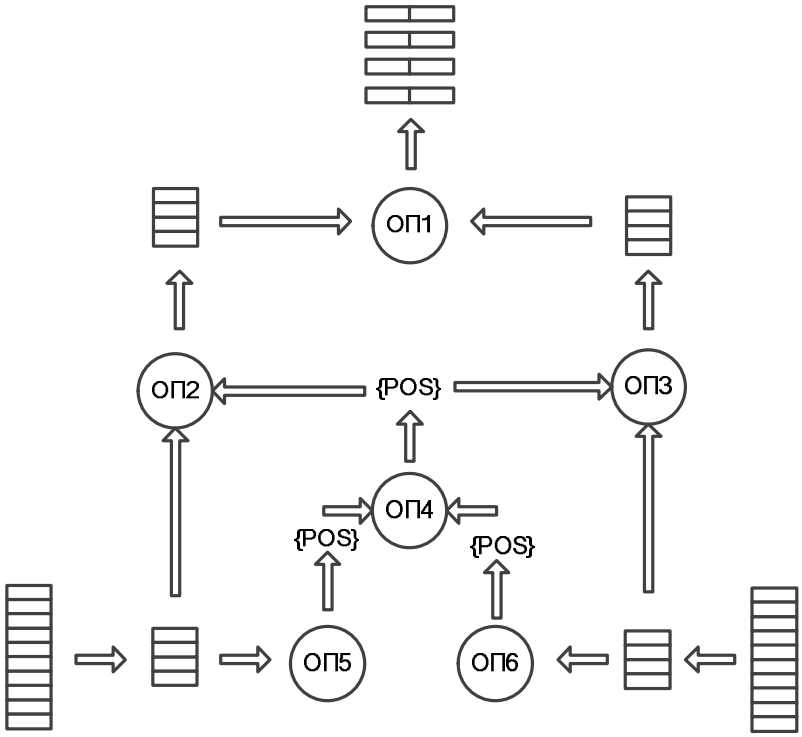


*Рис. 1.* Пример синхронного конвейера колоночного хранилища.

*Итераторная модель*. Она является общепринятым методом, используемым в СУБД для эффективной реализации синхронного конвейера. В соответствии с итераторной моделью с каждым узлом дерева плана запроса связывается специальная структура управления, называемая итератором [11]. Интерфейс итератора представлен двумя стандартными операциями с предопределенной семантикой: reset – установка итератора в состояние «перед первым кортежем», next – выдать очередной кортеж результирующего отношения. Методы reset и next родителя прямо или косвенно могут вызывать соответствующие методы дочерних узлов.

Для колоночных хранилищ характерны следующие изменения:

возможно наличие нескольких родительских узлов, т.е. результаты операции передаются не единственному следующему оператору;

используются как итераторы по кортежам, так и итераторы по блокам;

вводится операция материализации кортежа: получение исходного или необходимого на данном этапе плана запроса кортежа на основе передаваемых блоков значений атрибута.

Наличие нескольких родительских узлов ставит следующую проблему: если один родительский элемент обрабатывает входные данные значительно быстрее чем другой, данные могут быть потеряны и не переданы более медленному родительскому узлу. В [10] данную проблему решают следующим образом – вершиной у графа всегда является единственный элемент, который гарантирует одинаковую частоту запросов к нижестоящим узлам для всего графа.

*Скобочный шаблон*. Он используется для унифицированного представления узлов дерева запроса. В качестве основных методов здесь фигурируют `reset` и `next`, реализующие итератор. Основными атрибутами скобочного шаблона являются:

выходной буфер, в который помещается очередной кортеж результата;

КОП – код реляционной операции, реализуемой данным узлом;

указатель на скобочный шаблон левого сына;

указатель на скобочный шаблон правого сына («пусто» для унарных операций).

Сам по себе скобочный шаблон не содержит конкретной реализации реляционной операции. Однако после оптимизации запроса СУБД «вставляет» в каждый скобочный шаблон ту или иную реализацию соответствующей реляционной операции. Например, для операции соединения мы можем выбирать «соединение вложенными циклами», «соединение слиянием», «соединение хешированием» и др. Связь скобочного шаблона с конкретной реализацией операции осуществляется путем добавления еще одного специального атрибута в скобочный шаблон – «указатель на функцию реализации операции». В качестве параметра данной функции должен передаваться указатель на объемлющий скобочный шаблон.

Для колоночных систем в скобочный шаблон не вносится значительных изменений. Меняется формат выходных данных: это могут быть как кортежи, так и позиции элементов и указатели на блоки данных.

*Материализация кортежей*. Как было показано выше, одним из процессов

при формировании ответа на запрос в колоночных базах данных является материализация кортежей – процесс воссоздания кортежа на основе столбцов-атрибутов. В зависимости от момента применения данной операции в плане запроса в [12] предлагается следующие варианты материализации.

*Ранняя материализация*. Данный вариант аналогичен «естественной» материализации, применяемой в построчных хранилищах: каждый раз, когда осуществляется доступ к новому атрибуту, он добавляется к кортежу.

*Поздняя материализация*. Специфика колоночных хранилищ позволяет отложить процесс материализации до определенного момента, используя в процессе выполнения запроса позиции значений в колонках вместо самих значений атрибутов. К преимуществам данного метода можно отнести более высокую скорость работы с позициями значений по сравнению со всем кортежем. Слабым местом такого подхода является необходимость двойного чтения данных из столбца – в первом случае для получения позиций, во втором, уже после анализа и преобразования номеров, – для чтения непосредственно значений.

Таким образом, операцию материализации можно рассматривать в качестве момента, после которого исполнитель запросов начинает применять классические покортежные операции.

*Сжатие данных*. В современных СУБД широко используется сжатие данных, что позволяет повысить производительность за счет уменьшения числа дисковых операций ввода-вывода и объема передаваемых по сети данных. Колоночное хранение отношений дает возможность улучшить данный показатель по сравнению со строчными хранилищами. Это достигается за счет использования коэффициентов повторяемости значений атрибутов и возможности оперировать сжатыми данными (т.е. отсутствия затрат на декомпрессию). Ниже приведены описания некоторых алгоритмов сжатия, применяемых в колоночных базах данных [13].

**RLE (кодирование длин серий)** – последовательная серия одинаковых элементов данных заменяется на два символа: элемент и число его повторений.

**Словарный метод** – используется словарь, состоящий из последовательностей данных или слов. При сжатии эти слова заменяются на их коды из словаря. В наиболее распространенном варианте реализации в качестве словаря выступает сам исходный блок данных.

**Векторное кодирование –** с каждым значением сопоставляется битовая строка, где значение 1 обозначает позицию с данной величиной.

**Алгоритм Лемпеля-Зива-Велча** – при сжатии (кодировании) динамически создает таблицу преобразования строк: определенным последовательностям символов (словам) ставятся в соответствие группы бит фиксированной длины (обычно 12-битные).

В работе [13] предлагается полученный эмпирическим путем алгоритм выбора типа компрессии данных в столбцах.

*Параллельная обработка запросов*. Основной формой параллельной обработки запросов является фрагментный параллелизм. Подробно данный процесс рассмотрен в работах [5 – 8, 11]. В соответствии с этой схемой запрос на языке

SQL преобразуется в некоторый последовательный план. Данный последовательный план преобразуется в параллельный план, представляющий собой совокупность $n$ идентичных параллельных агентов, которые реализуют те же операции, что и последовательный план. Здесь $n$ обозначает количество процессорных узлов. Это достигается путем вставки оператора обмена exchange в соответствующие места дерева плана запроса. На завершающем этапе агенты рассылаются на соответствующие процессорные узлы, где интерпретируются исполнителем запросов. Результаты выполнения агентов объединяются корневым оператором exchange на нулевом процессорном модуле.

*Архитектуры параллельных систем баз данных*. Наиболее распространенной системой классификации параллельных систем баз данных является система, предложенная Майклом Стоунбрейкером (Michael Stonebraker) [11]: SE (Shared-Everything) – архитектура с разделяемыми памятью и дисками; SD (Shared-Disks) – архитектура с разделяемыми дисками; SN (Shared-Nothing) – архитектура без совместного использования ресурсов.

**Преобразование Лапласа-Стилтьеса времени выполнения простого запроса**

Используя подход, предложенный в [4], ниже выводим преобразование Лапласа-Стилтьеса (ПЛС) времени выполнения запроса к базе данных с планом $\pi_A(\sigma_F(R))$ и условием $F = f_0 \cap \mathop{\mathbf{I}}_{i=1}^{K_F} f_i$. При этом учитываются следующие условия:

колоночное хранение данных; наличие компрессии данных (метод RLE); каждая колонка хранится на диске в своих блоках; отдельная колонка представляет собой таблицу с кортежем (значение атрибута, позиция); время пересечения битовых масок не учитывается (для двух масок – это одна ассемблерная команда).

ПЛС времени выполнения запроса на одном процессоре кластера (или машины):

$$f(s) = G\left(\prod_{i=1}^{K_F} c_i(s, r_i) \cdot \prod_{j=K_F+1}^{K_F+K_A} c_j(s,1) \cdot (1 - P_1(1 - f_P^u(s)z))\right), \quad (1)$$

$$z = (1 - P_2(1 - f_M^2(s) f_N(s))), \quad (2)$$

где $G = z^{V/n}$ – производящая функция (ПФ) числа позиций (записей) проекции $R$, обрабатываемых на одном процессоре; $V$ – общее число записей в проекции $R$; $n$ – число процессоров в кластере (или в машине).

*Примечание*. При хорошей хеш-функции таблица равномерно фрагментируется по ключевому (уникальному) атрибуту с точностью до 0.5%.

$K_F$ – число атрибутов в условии F, для которых заданы элементарные условия поиска (т.е. фильтрация по атрибуту)

$$c_i(s, r) = f_{Di}(s) f_M^2(s) f_P^r(s), \quad (3)$$

где $f_{Di}(s)$ – ПЛС времени чтения кортежа $i$-го столбца с диска; $f_M^2(s)$ – ПЛС времени сохранения кортежа в ОП ($f_M(s)$) и его чтения в кэш процессора ($f_M(s)$).

*Примечание*. Будем считать, что кэш процессора – это «черная дыра», в ко-

торой сохраняются данные процессора, необходимые для вычислений. Из кэша в ОП перемещаются только результирующие материализованные записи, передаваемые по шине (см. далее $f_N(s)$) процессору, где выполняется сборка.

$f_P^r(s)$ – ПЛС времени обработки кортежа столбца в процессоре, r – число логических операций, необходимых для проверки условия по соответствующему атрибуту (для i-го столбца вводится аргумент $r_i$ – см. (1)).

$K_A$ – число атрибутов в проекции $\pi_A$ запроса и число атрибутов, входящих в предикат $f_0$.

$c_j(s,1)$ – ПЛС времени чтения кортежа $j$-го столбца проекции $p_A$ с диска в кэш процессора (см. (3)), где 1 означает, что в процессоре проверяется только значение битовой маски в позиции, указанной в кортеже, а

$$G(...(1-P_1(1-z))) - \quad (4)$$

это ПФ числа записей таблицы $R$, удовлетворяющих элементарным условиям по атрибутам $K_F$ ( $\mathop{\mathbf{I}}_{i=1}^{K_F} f_i$ ), $P_1 = \prod_{i=1}^{K_F} P_{f_i}$ .

$f_P^u(s)$ – учитывает, что для проверки условия $f_0$ для материализованных записей (4) потребуется '$u$' логических операций процессора.

$$z = (1-P_2(1-z_1))). \quad (5)$$

После подстановки этого выражения в (4) получаем ПФ числа записей, удовлетворяющих и условию $\mathop{\mathbf{I}}_{i=1}^{K_F} f_i$ (см. $P_1$), и условию $f_0$ (см. $P_2$).

$f_M^2(s)$ – учитывает перемещение записей таблицы $R$, удовлетворяющих условию поиска F, из кэша процессора в ОП, а затем из ОП в буфер межпроцессорной шины.

$f_N(s)$ – учитывает передачу записей таблицы R, удовлетворяющих условию поиска $F$, по шине процессору, выполняющему сборку.

### Оценка среднего времени выполнения запроса

Для получения формул воспользуемся подходом, предложенным в работах [5 – 8]. Хотя поведение параллельной системы баз данных при выполнении запроса описывается в виде замкнутой системы массового обслуживания (СМО), в качестве модели разделяемого ресурса предлагается использовать разомкнутую СМО M/M/1 [7].

В табл. 1 приведены результаты сравнения замкнутой «модели ремонтника» и соответствующей разомкнутой СМО M/M/1.

Как видно, на уровне загрузки разделяемого ресурса $r$ = 0.6 имеет место приемлемая погрешность.

Особая точность не требуется, поскольку цель проводимого исследования – это сопоставление и выбор варианта (архитектуры) системы, построенной на основе параллельной колоночной базы данных (ППБД). В то же время расчеты существенно упрощаются.

*Таблица 1*

| Количество процессоров в модели ремонтника $n$ | Загрузка разделяемого ресурса $r = (n\text{-}1)b/a$ | Погрешность оценки среднего времени пребывания в разделяемом ресурсе (%) |
|---|---|---|
| 6 | 0,4 | 12 |
| | 0,5 | 22 |
| | 0,6 | 39 |
| | 0,7 | 71 |
| 12 | 0,4 | 6 |
| | 0,5 | 13 |
| | 0,6 | 24 |
| | 0,7 | 46 |
| 18 | 0,4 | 4 |
| | 0,5 | 9 |
| | 0,6 | 18 |
| | 0,7 | 35 |

Формулы для $f_{Di}(s)$, $f_M(s)$, $f_N(s)$, $f_P(s)$ (см. формулы (1) – (3)) в зависимости от архитектуры ППБД представлены в табл. 2.

*Таблица 2*

<table>
<tr><th></th><th>SE</th><th>SD</th><th>SN</th></tr>
<tr><td>$f_{Di}(s)$</td><td colspan="2">$(1-\frac{1}{L_i})+\frac{1}{L_i}((1-p_D)+p_D\frac{m_{DB}-(n-1)\cdot(p_D\sum_{i=1}^{K_F+K_A}\frac{l_{Di}}{L_i})/N}{m_{DB}-(n-1)\cdot(p_D\sum_{i=1}^{K_F+K_A}\frac{l_{Di}}{L_i})/N+s}\}$</td><td>$(1-\frac{1}{L_i})+\frac{1}{L_i}((1-p_D)+p_D\frac{m_{DB}}{m_{DB}+s})$</td></tr>
<tr><td>$f_M(s)$</td><td rowspan="2">$\frac{m_M-(n-1)l_M}{m_M-(n-1)l_M+s}$<br>(обмен между процессорами осуществляется через ОП)</td><td colspan="2">$\frac{m_M}{m_M+s}$</td></tr>
<tr><td>$f_N(s)$</td><td colspan="2">$\frac{m_N-(n-1)l_N}{m_N-(n-1)l_N+s}$</td></tr>
<tr><td>$f_P(s)$</td><td colspan="3">$\frac{m_P}{m_P+s}$</td></tr>
</table>

Поясним формулу $f_{Di}(s)$ для архитектур SE и SD. Это ПЛС времени чтения кортежа $i$-го столбца с диска. Здесь введены следующие обозначения: $L_i$ – число позиций в блоке $i$-го столбца:

$$L_i = \frac{Q_{\text{Б}}}{\text{Q}_{\text{Ki}} / k_{Ci}}; \qquad (6)$$

$Q_Б$ – размер блока диска; $Q_{Ki}$ – размер кортежа $i$-го столбца; $k_{Ci}$ – среднее число позиций, покрываемых одним кортежем (учитывает сжатие столбца); $(1 - p_D)$ – вероятность, что блок находится в буфере; $1/\mu_{DB}$ – среднее время чтения блока с диска;

$$(n-1)\cdot(p_D \sum_{i=1}^{K_F+K_A} \frac{l_{Di}}{L_i}) \qquad (7)$$

– суммарная интенсивность заявок со всех $(n-1)$ процессоров, на которых выполняется запрос, на чтение блоков со всех разделяемых дисков (–1 учитывает, что конкретный процессор не ждет сам себя, что следует из теории экспоненциальных сетей СМО); $(p_D / L_i)$ – учитывает, что каждая $L_i$-я позиция столбца читается из другого блока, а $p_D$ – это вероятность того, что этот блок надо читать с диска; поэтому и получается, что $(p_D / L_i)\cdot l_{Di}$ – интенсивность заявок на чтение блоков $i$-го столбца с диска; $l_{Di}$ – интенсивность заявок с одного процессора, на котором выполняется запрос, на чтение кортежей $i$-го столбца (эта интенсивность определяется по аналогии с [1, 2], только там $Q_D$ определяется для каждого i); N – число разделяемых дисков.

Формула для $f_{Di}(s)$ (см. табл. 2, колонки SE и SD) читается так:

с вероятностью $(1 - 1/L_i)$ позиция $i$-го столбца читается из текущего блока, который находится в буфере (ПЛС = 1, т.е. время чтения с диска равно 0);

с вероятностью $(1/L_i)$ позиция должна читаться из другого блока, но с вероятностью $(1 - p_D)$ этот блок уже находится в буфере (ПЛС = 1, т.е. время чтения с диска равно 0), и уже с вероятностью $p_D$ блок читается с диска (ПЛС для времени пребывания в СМО M/M/1).

Дифференцируя выражение (1) как сложную функцию по s в нуле, можно получить моменты случайного времени $(\xi)$ обработки простого запроса в параллельной колоночной базе данных (ППБД):

$$M_x = -f'(0),\ M_{x^2} = f''(0),\ s_x^2 = M_{x^2} - M_x^2. \qquad (8)$$

После дифференцирования (1) получим

$$M = \sum_{i=1}^{K_F+K_A} Q_{Di}\cdot \bar{f}_{Di} + Q_M \cdot \bar{f}_M + Q_N \cdot \bar{f}_N + Q_P \cdot \bar{f}_P\ , \qquad (9)$$

где

$$Q_{Di} = \frac{V}{n}, \qquad (10)$$

$$Q_M = 2\frac{V}{n}((K_a+1) + K_f + P_1P_2), \qquad (11)$$

$$Q_N = \frac{V}{n}P_1P_2, \qquad (12)$$

$$Q_P = \frac{V}{n}((K_a+1) + 2\sum_{i=1}^{K_f} r_i + u). \qquad (13)$$

Исследуем поведение математического ожидания времени выполнения запроса к БД для архитектуры SE. Предположим, что в параллельной системе баз данных "узким местом" является подсистема ввода/вывода (диск). Тогда, используя ПЛС из табл. 2 для расчета средних величин $\bar{f}_{Di}$, $\bar{f}_M$, $\bar{f}_N$, $\bar{f}_P$, а также выражение (9), получим

$$M = \frac{1}{m_{DB} - (n-1)\cdot(p_D \sum_{i=1}^{K_F+K_A} \frac{l_{Di}}{L_i})/N} \sum_{i=1}^{K_F+K_A} \frac{p_D \cdot Q_{Di}}{L_i} + \frac{Q_M + Q_N}{m_M} + \frac{Q_P}{m_P}, \qquad (14)$$

где $Q_{Di}, Q_M, Q_N, Q_P$ определяются формулами (10) – (13),

$$l_{Di} = \frac{Q_{Di}}{\frac{Q_M + Q_N}{m_M} + \frac{Q_P}{m_P}}. \qquad (15)$$

## Пример расчета

Был выполнен расчет отношения среднего времени обработки простого запроса $\pi_A(\sigma_F(R))$ в строчной СУБД к среднему времени выполнения этого запроса в колоночной СУБД в зависимости от отношения количества атрибутов, участвующих в запросе, к общему количеству атрибутов в таблице. Среднее время для колоночной СУБД рассчитывалось по формуле (14), среднее время для строчной СУБД получено в [5, 6]. Для упрощения расчетов будем считать, что $K_A = K_F = K$ т.е. количество атрибутов, участвующих в операции фильтрации, равно количеству атрибутов, использующихся в операции проекции. Также примем, что таблица состоит из H = 100 одинаковых по размеру и типу атрибутов. Расчеты были выполнены при следующих значениях характеристик ресурсов:

процессор Intel Xeon 5160. Для выбранного процессора измеренное значение числа процессорных циклов, выполняемых в секунду $\mu_P = 1.5\cdot10^9$(1/c);

внешняя память $N$ = 50 дисков 3.5" Seagate Cheetah 15K.6 ST3146356FC; размер блока чередования (stripe size) $Q_{БЧ}$ = 64 Кб;

среднее время поиска и чтения блока чередования с диска

$t_{БЧ} = t_{подвода} + t_{вращения}/2 + Q_{БЧ}/v_{чтения} = 4 + 4/2 + 64/200 = 6.3$ мс,

при этом интенсивность чтения блоков с диска равна $\mu_{DB} = 1000/6.3 = 160$ (1/c);

оперативная память DDR3-1600 PC3-12800. Расчеты показывают, что интенсивность чтения записей базы данных из ОП равна $\mu_M = 10.4\cdot10^6$ (1/c);

Остальные параметры для расчетов приведены в табл. 3, где $L$ – среднее число записей таблицы $R$ в блоке чередования для строчной СУБД; $k_C$ – среднее число позиций, покрываемых одним кортежем столбца колоночной базы данных (см. (6), $k_C = 1$ – нет сжатия).

*Таблица 3*

| | | |
|---|---|---|
| $V = 10^6$ | $r_i = 20$ | $L = 100$ |
| $P_1 = 0.01$ | $U = 50$ | $H = 100$ |
| $P_2 = 0.007$ | $p_D = 0.9$ | $L_i = L\cdot H\cdot k_C$ |

Графики отношения времени выполнения запроса в строчной СУБД к времени выполнения запроса в колоночной СУБД (Y) в зависимости от отношения количества атрибутов, участвующих в запросе, к общему количеству атрибутов в таблице (X = 100*2*K/H) для различного среднего числа позиций, покрываемых одним кортежем ($k_C$ учитывает сжатие), представлены на рис. 2. Графики построены для числа процессоров $n$ = 2.

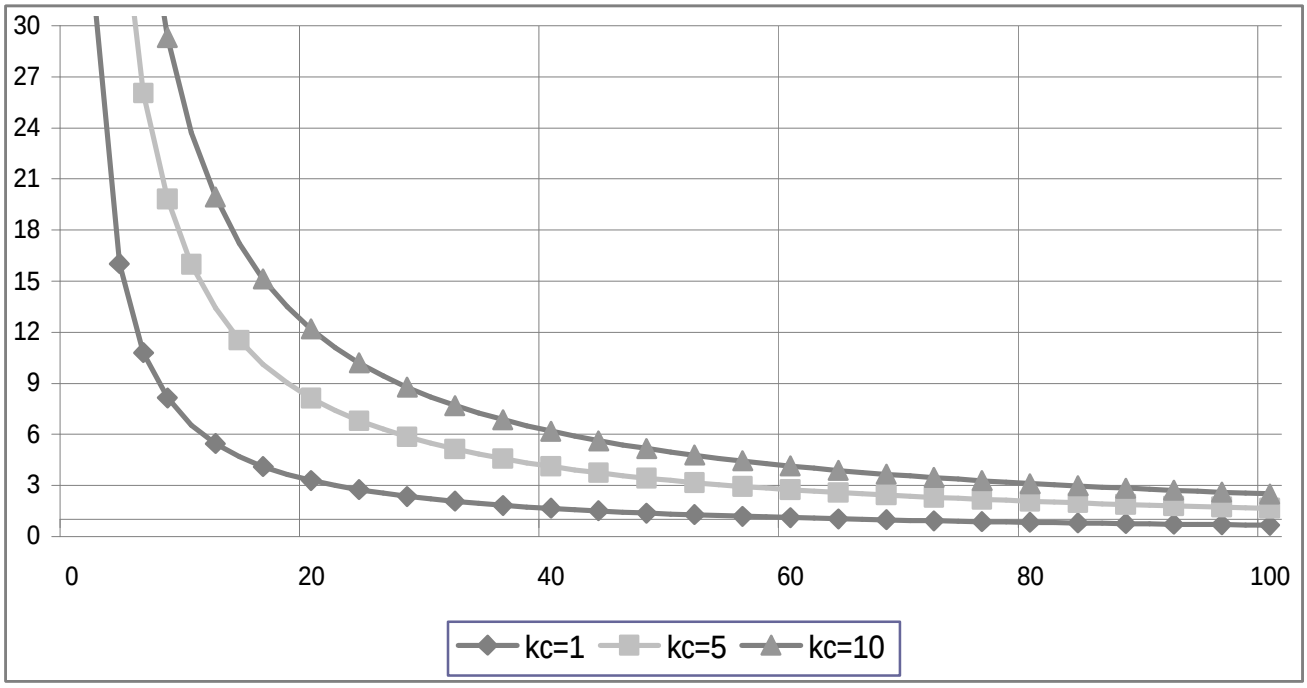


*Рис. 2.* Отношение времени выполнения запроса в строчной СУБД к времени выполнения запроса в колоночной СУБД (Y, %,) в зависимости от отношения количества атрибутов, участвующих в запросе, к общему количеству атрибутов в таблице (X, %) при разных '$k_C$'.

В табл. 4 приведены значения Y для некоторых X.

*Таблица 4*

| | $k_C = 1$ | $k_C = 5$ | $k_C = 10$ |
|---|---|---|---|
| X = 2% ($K_A = K_F = 1$) | 31 | 70 | 99 |
| X = 65% | 1 | 2,5 | 3,8 |
| X = 100% | 0,66 | 1,65 | 2,5 |

Из графиков видно, что при использовании менее 20% атрибутов время выполнения запроса в колоночной СУБД меньше в разы по сравнению со строчной СУБД, при большем количестве атрибутов время выполнения запроса растет практически пропорционально числу используемых в запросе атрибутов. Для $k_C$ = 1 (нет сжатия данных) среднее время выполнения запроса в строчной и колоночной СУБД становится равным (Y=1) при использовании 65% атрибутов (табл. 4). Увеличение времени выполнения запроса в колоночной СУБД при большем количестве используемых атрибутов можно объяснить ростом числа читаемых с диска столбцов таблицы (для строчных СУБД время не изменяется, так как с диска записи читаются целиком). При X=100% и $k_C$=1 среднее время выполнения запроса в строчной СУБД в 1.5 (1/0.66) раза меньше, чем в колоночной СУБД. При достаточно хорошем сжатии столбцов таблицы картина меняется: колоночная СУБД лучше строчной даже при использовании в запросе 100% атрибутов (см. табл. 4).

На рис.3 представлены графики зависимости среднего времени выполнения запроса в колоночной СУБД от числа процессоров для различного соотношения используемых в запросе атрибутов (10%, 50%, 100%), а также время выполнения запроса в строчной СУБД. Из графиков видно, что для строчной СУБД десятисе-

кундная отметка среднего времени обработки запроса достигается при числе процессоров n=15. Для колоночной СУБД эта отметка достигается при соотношении используемых в запросе атрибутов 10% (10 атрибутов из 100) уже при n = 2 (и это при отсутствии сжатия столбцов), $k_C = 1$. Экономия средств налицо.

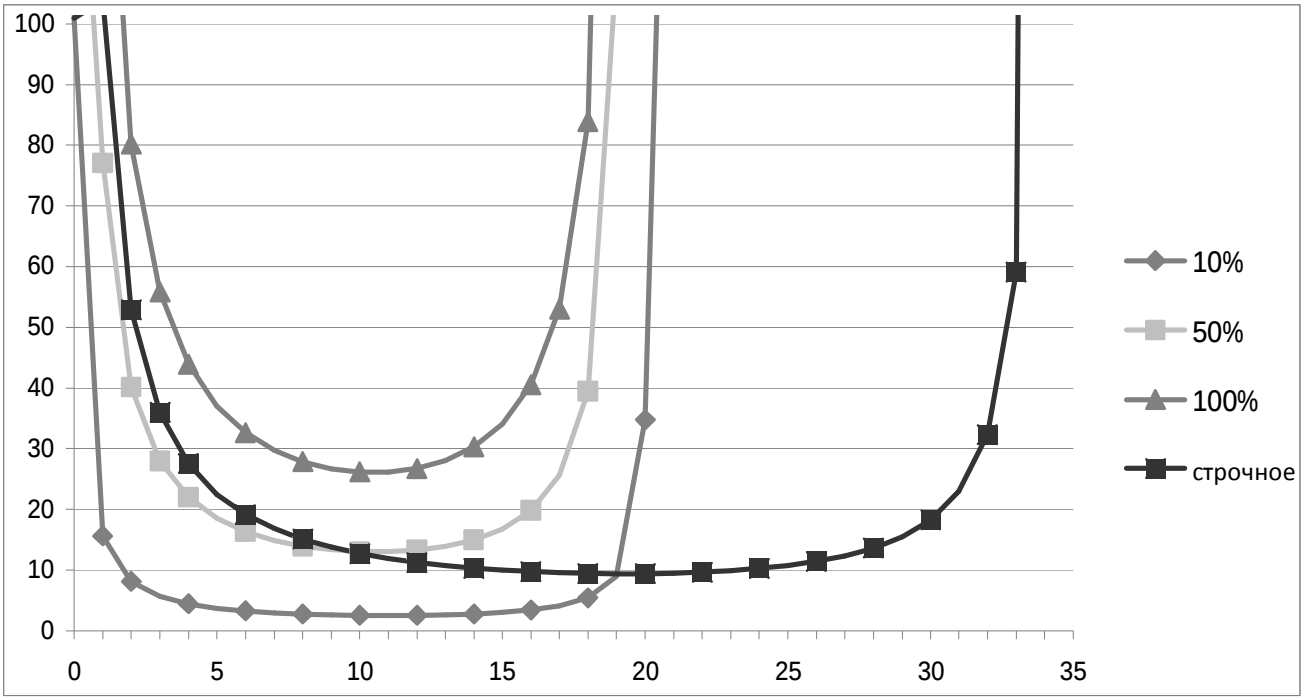


*Рис. 3.* Время выполнения запроса в колоночной СУБД (с) в зависимости от числа процессоров для различного отношения используемых в запросе атрибутов и время выполнения запроса в строчной СУБД (везде $k_C = 1$).

## Заключение

Проанализирован способ выполнения запросов в параллельной колоночной системе баз данных. Рассмотрены изменения, вносимые в синхронный конвейер, итераторную модель и скобочный шаблон, а также операции материализации и компрессии данных.

Получено преобразование Лапласа-Стилтьеса (ПЛС) времени выполнения запроса, имеющего план $\pi_A(\sigma_F(R))$, в параллельной колоночной СУБД. Рассмотрены варианты этого преобразования для различных архитектур параллельных систем баз данных.

Приведен пример расчета отношения среднего времени выполнения запроса в строчной СУБД к среднему времени выполнения запроса в колоночной СУБД в зависимости от отношения количества атрибутов, участвующих в запросе, к общему количеству атрибутов в таблице. На его основании можно сделать вывод, что при хорошем сжатии столбцов (см. $k_C$) время выполнения запроса в колоночной СУБД меньше, чем в строчной СУБД, даже при использовании в запросе 100% атрибутов (см. табл. 4).

Для колоночной СУБД десятисекундная отметка среднего времени выполнения запроса при отношении используемых в запросе атрибутов 10% достигает-

ся при меньшем числе процессоров ($n$ = 2), чем для строчных СУБД ($n$ = 15). Это свидетельствует об экономии средств при использовании колоночных СУБД.

В дальнейшем плпнируется получить оценки времени выполнения запросов с более сложными планами реализации (например, для плана соединения таблиц).

ЛИТЕРАТУРА


1. *Арсентьев А*. Хранилища данных становятся инфраструктурным компонентом №1. CNews аналитика. 2010. [Электронный ресурс]. [http://retail.cnews.ru/reviews/free/BI2010/articles/ articles6.shtml]. Проверено 27.06.2011.
2. ***Stonebraker M.***. My Top 10 Assertions About Data Warehouses. / Перевод Сергея Кузнецова, 2010 г.: [Электронный ресурс]. [http://citforum.ru/gazeta/166/]. Проверено 27.06.2011.
3. ***Stonebraker M., Bear Ch., Çetintemel U., Cherniack M. and etc.*** One Size Fits All? – Part 2: Benchmarking Results. 3rd Biennial Conference on Innovative Data Systems Research (CIDR), January 7-10, 2007, Asilomar, California, USA. / перевод С. Кузнецова, 2007 г.: [Электронный ресурс]. [http://citforum.ru/database/articles/one_size_fits_all_2/]. Проверено 27.06.2011.
4. *Григорьев Ю.А., Плутенко А.Д.* Теоретические основы анализа процессов доступа к распределенным базам данных. – Новосибирск: Наука, 2002.
5. *Григорьев Ю.А., Плужников В.Л.*. Оценка времени выполнения запросов и выбор архитектуры параллельной системы баз данных. – М.: Изд-во МГТУ, 2009.
6. *Григорьев Ю.А., Плужников В.Л.* Модель обработки запросов в параллельной системе баз данных // Вестник МГТУ им. Н.Э. Баумана. – 2010. – № 4. – С.78-90.
7. *Григорьев Ю.А., Плужников В.Л.* Оценка времени соединения таблиц в параллельной системе баз данных// Информатика и системы управления. – 2011. – № 1. – С.3-16.
8. *Григорьев Ю.А., Плужников В.Л.* Анализ времени обработки запросов к хранилищу данных в параллельной системе баз данных // Информатика и системы управления. – 2011. – № 2. – С. 94-106.
9. ***Stonebraker M.l, Abadi D.J., Batkin A., Chen X. and etc.*** C-Store: A Column-Oriented DBMS [Электронный ресурс]. [http://www.cs.yale.edu/homes/dna/pubs/displaypubs.cgi/]. Проверено 22.10.2011.
10. ***Daniel J.A.*** Query Execution in Column-Oriented Database Systems. [Электронный ресурс]. [http://www.cs.yale.edu/homes/dna/papers/abadiphd.pdf]. Проверено 25.12.2011.
11. *Соколинский Л.Б., Цымблер М.Л.* Лекции по курсу "Параллельные системы баз данных": [Электронный ресурс]. [http://pdbs.susu.ru/CourseManual.html]. Проверено 22.10.2011.
12. ***Daniel J.A., Daniel S.M., David J. DeWitt, Samuel R.M.*** Materialization Strategies in a Column-Oriented DBMS ***In Proceedings of ICDE, 2007.*** [Электронный ресурс]. http://db.lcs.mit.edu/ projects/cstore/abadiicde2007.pdf]. Проверено 25.12.2011.
13. ***Daniel J. A., Samuel R.M., Miguel C.F.*** Integrating Compression and Execution in Column-Oriented Database Systems ***In Proceedings of ICDE, 2006.*** [Электронный ресурс]. [http://db.lcs. mit.edu/projects/cstore/abadisigmod06.pdf]. Проверено 25.12.2011.
14. *Жожикашвили В.А., Вишневский В.М.* Сети массового обслуживания. Теория и применение к сетям ЭВМ. – М.: Радио и связь, 1988.
15. *Клейнрок Л.* Теория массового обслуживания. – М.: Машиностроение, 1979.
16. *Бронштейн О.И., Духовный И.М.* Модели приоритетного обслуживания в информационно-вычислительных системах. – М.: Наука, 1976.



*E-mail:*
*Григорьев Юрий Александрович – grigorev@iu5.bmstu.ru;*
*Ермаков Евгений Юрьевич – JK.Ermakov@gmail.com*.