УДК 004.657

© 2009 г. **Ю.А. Григорьев,** д-р техн. наук,
В.Л. Плужников
(Московский государственный технический университет им. Н.Э. Баумана)

ОЦЕНКА ВРЕМЕНИ ВЫПОЛНЕНИЯ ЗАПРОСОВ И ВЫБОР АРХИТЕКТУРЫ ПАРАЛЛЕЛЬНОЙ СИСТЕМЫ БАЗ ДАННЫХ

В статье анализируются существующие методы выполнения запросов в последовательной и параллельной СУБД. Приводится преобразование Лапласа-Стилтьеса (ПЛС) времени выполнения запроса в параллельной СУБД и рассматриваются варианты этого преобразования для различных архитектур параллельных систем баз данных. Анализируется способ выбора архитектуры по критерию стоимости с ограничением на верхнюю границу доверительного интервала времени выполнения запроса.
Ключевые слова. Параллельные базы данных, преобразование Лапласа-Стилтьеса, математическое ожидание времени выполнения запроса.

Введение

Параллельные системы баз данных начинают вытеснять традиционные компьютеры основного класса, так как они позволяют работать со значительно более крупными базами данных в режиме, поддерживающем транзакции [1]. Успех таких систем опровергает прогноз статьи 1983 г. [2], предрекавшей скорое исчезновение машин баз данных. За последние десять лет компании Teradata, Tandem и ряд новоявленных компаний успешно разрабатывали и продавали параллельные машины. Одно из объяснений – широкое распространение реляционных баз данных. В 1983 г. они только еще появлялись на рынке, сегодня же доминируют. Реляционные запросы как нельзя лучше подходят для параллельного выполнения. Они состоят из однородных операций над однородным потоком данных. Каждая операция образует новое отношение (таблицу), так что из операций могут быть составлены высокопараллельные графы потоков данных.

В настоящее время не существует научно обоснованного метода оценки и выбора архитектуры параллельной системы баз данных. Сравнительный анализ архитектурных решений выполняется или на основе экспертных оценок качественных критериев (масштабируемости, доступности данных, баланса загрузки, межпроцессорных коммуникаций, когерентности кэшей, организации блокировок и др.) [3], или на основе результатов тестов (TPC и др.) для конкретных платформ [4].

В статье предлагается математический метод оценки и выбора архитектуры

параллельных систем баз данных, учитывающий особенности выполнения запросов к базе данных проектируемой системы, а также топологию (структуру) различных архитектурных решений.

Выполнение запроса в последовательной СУБД

В следующих двух разделах в качестве иллюстрации основных положений параллельных систем баз данных приведены некоторые важные сведения, изложенные Л.Б. Соколинским и М.Л. Цымблером в работе [3].

Известно [3, 5, 6], что SQL-запрос, поступивший в СУБД, подвергается оптимизации. СУБД строит иерархический план выполнения запроса так, чтобы минимизировать затраты ресурсов (стоимость), связанные с реализацией запроса.

Исполнение последовательных планов основывается на следующих трех базовых парадигмах: синхронный конвейер; итераторная модель; скобочный шаблон. Следует отметить, что синхронный конвейер, итераторная модель и скобочный шаблон могут быть использованы с равным успехом и при реализации параллельных СУБД.

Синхронный конвейер. Суть данного метода состоит в том, что, как только операция получает очередной кортеж своего результирующего отношения, она передает его по конвейеру вышестоящей операции для обработки. Например, узел, читающий записи из исходной таблицы, передает их узлу, выполняющему соединение записей разных таблиц.

Итераторная модель. Эта модель является общепринятым методом, используемым в СУБД для эффективной реализации синхронного конвейера. В соответствие с итераторной моделью с каждым узлом дерева плана запроса связывается специальная структура управления, называемая итератором. Интерфейс итератора представлен двумя стандартными операциями с предопределенной семантикой:

reset – установиь итератор в состояние "перед первым кортежем";

next – выдать очередной кортеж результирующего отношения.

Алгоритм выполнения плана запроса на базе итераторной модели изображен на рис. 1. На первом шаге выполняется метод reset применительно к корневому узлу. Затем в цикле выполняется метод next для корневого узла. Он каждый раз возвращает указатель на очередной кортеж результирующего отношения. В приведенном примере эти кортежи просто выводятся на экран. Цикл завершается, когда метод next выдает указатель на специальный кортеж, обозначающий конец файла – EOF (End Of File). Методы reset и next родителя прямо или косвенно могут вызывать соответствующие методы дочерних узлов. Эти вызовы изображены на рисунке пунктирными стрелками. Реализация итератора базируется на скобочном шаблоне, который рассмотрен ниже.

Скобочный шаблон. Основными атрибутами скобочного шаблона являются выходной буфер, в который помещается очередной кортеж результата (он может быть прочитан путем вызова родителем метода next соответствующего дочернего узла);

КОП – код реляционной операции, реализуемой данным узлом;

указатель на скобочный шаблон левого дочернего узла;

указатель на скобочный шаблон правого дочернего узла ("пусто" для унарных операций).

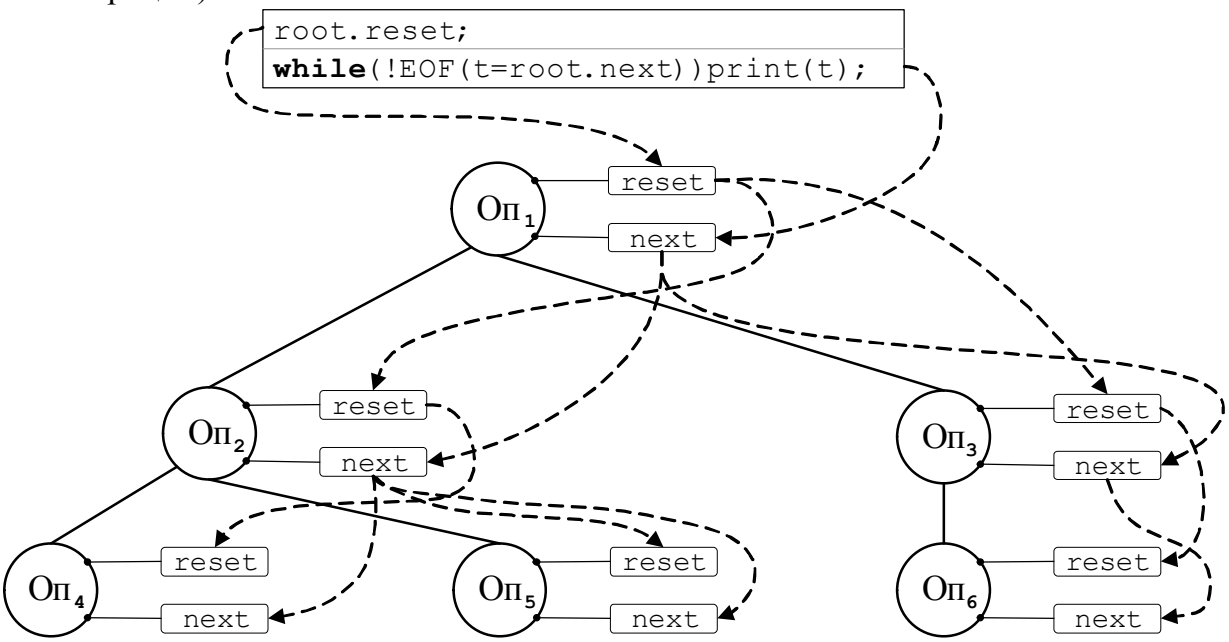

Рис. 1. Алгоритм выполнения плана запроса на базе итераторной модели.

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 СУБД может выбрать один из следующих кодов: "соединение вложенными циклами", "соединение сортировкой и слиянием", "соединение хешированием". При выполнении этой операции узел обращается к скобочным шаблонам своих дочерних узлов.

Выполнение запроса в параллельной СУБД

Основной формой параллельной обработки запросов является фрагментный параллелизм (рис. 2). Каждое отношение (таблица) делится на части, называемые фрагментами. Фрагменты отношения распределяются по различным процессорным узлам многопроцессорной системы. Способ фрагментации определяется функцией фрагментации ψ, которая для каждого кортежа отношения вычисляет номер процессорного узла, на котором должен быть размещен этот кортеж. На рис. 2 изображена фрагментация отношения "Поставщик" (П) по атрибуту Код_П (код поставщика) на основе метода диапазонов. В данном случае функция фрагментации имеет вид: ψ(П) = П.Код_П div 10. Здесь div – операция деления нацело. В простейшем случае запрос параллельно выполняется на всех процессорных узлах. Полученные фрагменты сливаются в результирующее отношение.

Если оба отношения, участвующие в соединении, фрагментированы не по атрибуту соединения, то СУБД придется перераспределять между процессорами оба входных отношения. При этом в качестве функции распределения для обоих

отношений СУБД может взять любую (но одну и ту же) функцию распределения по атрибуту соединения, которая отправляет кортежи с одинаковыми значениями атрибута соединения на один и тот же процессорный узел.

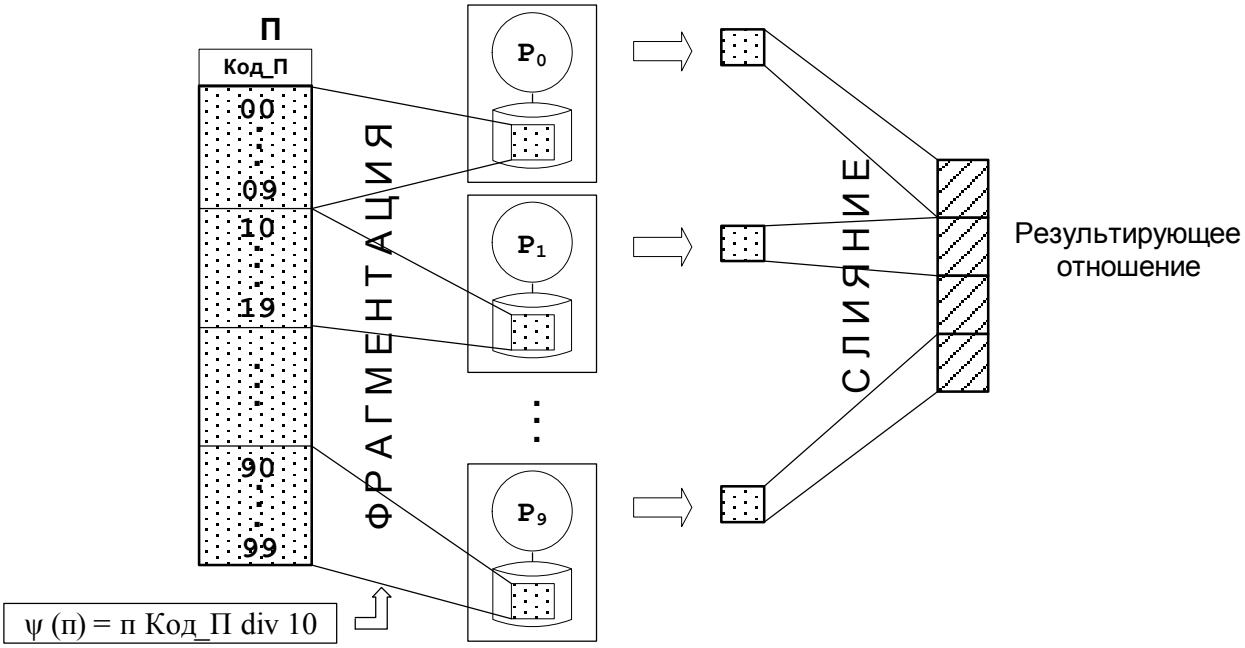

Рис. 2. Фрагментный параллелизм.

Для организации межпроцессорных обменов в соответствующие места дерева плана запроса СУБД вставляет специальный оператор exchange. Оператор exchange реализуется на основе использования стандартного скобочного шаблона. Оператор exchange имеет два специальных параметра, определяемых пользователем: номер порта обмена и указатель на функцию распределения. Функция распределения для каждого кортежа вычисляет логический номер процессорного узла, на котором данный кортеж должен быть обработан. Параметр "порт обмена" позволяет включать в дерево запроса произвольное количество операторов exchange (для каждого оператора указывается свой уникальный порт обмена).

Структура оператора обмена exchange изображена на рис. 3. Оператор exchange является составным оператором и включает в себя четыре оператора: gather, scatter, split и merge.

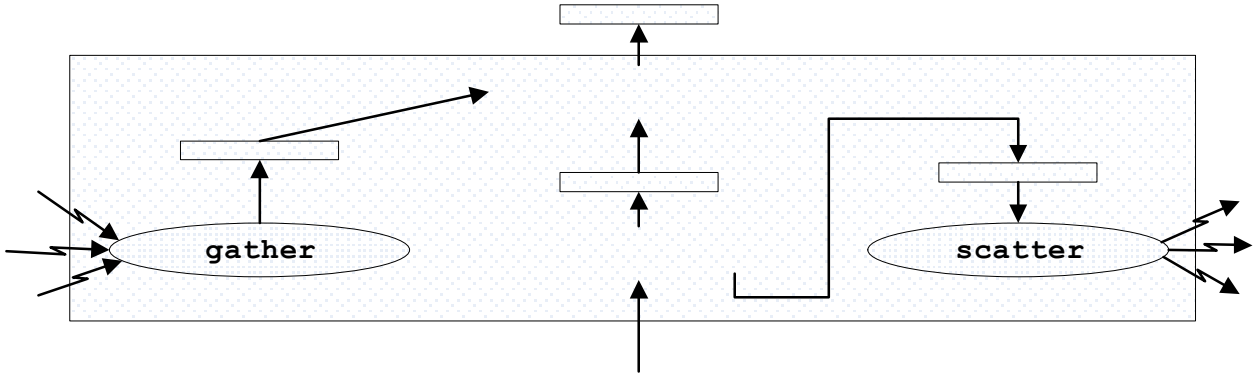

Рис. 3. Структура оператора exchange.

Все указанные операторы реализуются на базе стандартного скобочного шаблона. Split –бинарный оператор, который осуществляет разбиение кортежей, поступающих из входного потока, на две группы: свои и чужие. Свои – это кортежи, которые должны быть обработаны на данном процессорном узле. Они направляются в выходной буфер оператора split. Чужие, т.е. кортежи, которые должны быть обработаны на процессорных узлах, отличных от данного, помещаются оператором split в выходной буфер правого дочернего узла, в качестве которого фигурирует оператор scatter. Здесь выходной буфер оператора split играет роль входного потока данных.

Нульарный оператор scatter извлекает кортежи из своего выходного буфера и пересылает их на соответствующие процессорные узлы, используя заданный номер порта. Запись кортежа в порт может быть завершена только после того, как реципиент выполнит операцию чтения из данного порта.

Нульарный оператор gather выполняет перманентное чтение кортежей из указанного порта со всех процессорных узлов, отличных от данного. Считанные кортежи помещаются в выходной буфер оператора gather.

Оператор merge определяется как бинарный, который забирает кортежи из выходных буферов своих дочерних узлов и помещает их в собственный выходной буфер.

Общая схема обработки запроса в параллельной СУБД изображена на рис. 4. В соответствие с этой схемой запрос на языке SQL преобразуется в некоторый последовательный план.

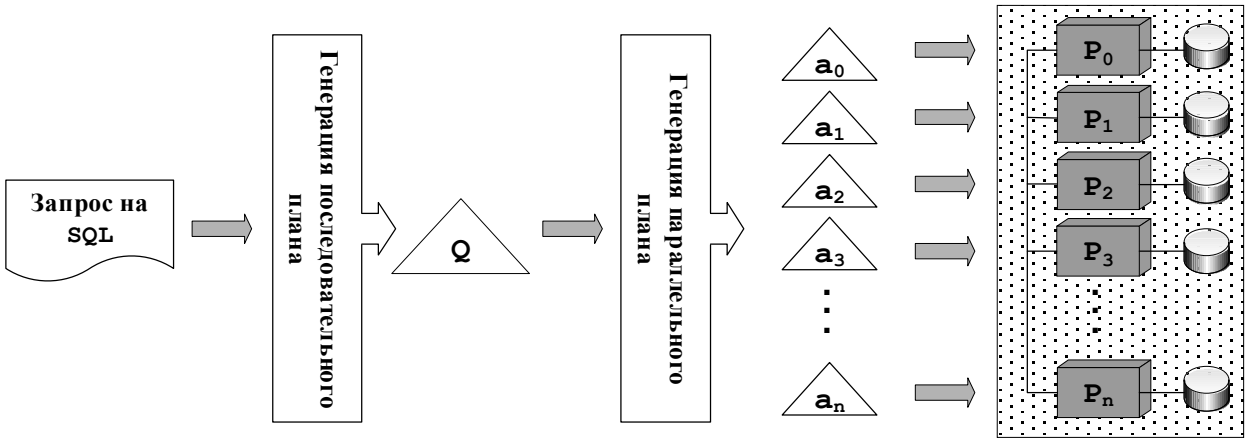

Рис. 4. Обработка запроса в параллельной СУБД.

Данный последовательный план преобразуется в параллельный план, представляющий собой совокупность *n* идентичных параллельных агентов, которые реализуют те же операции, что и последовательный. 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.

Оценка времени выполнения запроса в параллельной системе баз данных и метод выбора архитектуры системы

Наиболее распространенной системой классификации параллельных систем баз данных является система, предложенная Майклом Стоунбрейкером (Michael Stonebraker) [5, 7] (рис. 5). Здесь: *P* – процессор; *M* – модуль оперативной памяти; *D* – дисковое устройство; *N* – соединительная сеть. Эта классификация неполная, имеются и другие конфигурации. В частности архитектура SN может быть реализована на основе персональных компьютеров.

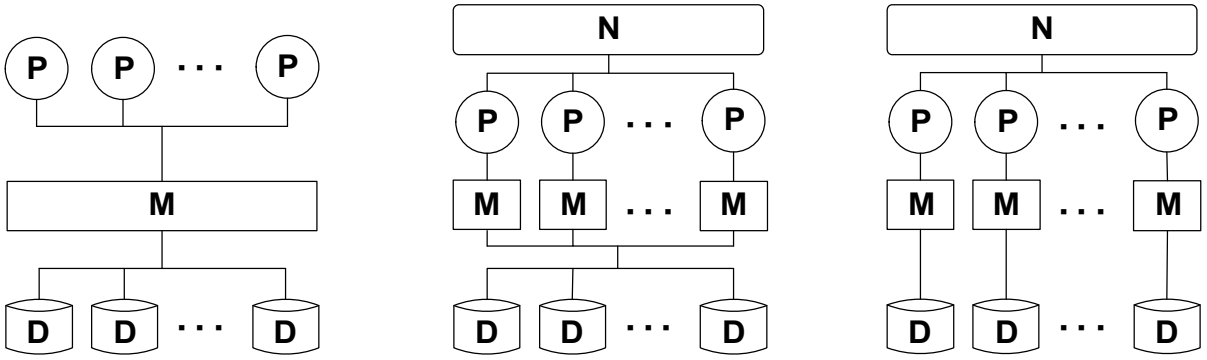

Рис. 5. Классификация Стоунбрейкера.

Ниже предлагаются оценки времени выполнения запроса к базе данных с планом $\pi_A(\sigma_F(R))$ для различных конфигураций параллельных систем баз данных.

Используя подход, предложенный в [4], для этого запроса можно вывести следующее преобразование Лапласа-Стилтьеса (ПЛС) времени выполнения:

$$\varphi(s) = G(\varphi_D(s)\varphi_M(s)(1 - P_F(1 - \varphi_N(s)))), \quad (1)$$

где $G = z^{V/n}$ – производящая функция числа записей фрагментной таблицы R, обрабатываемой на одном процессоре; V – общее число записей в таблице R; n – число процессоров (или персональных компьютеров в кластере); $\varphi_D(s)$ – ПЛС времени чтения записи фрагментированной таблицы с диска (с учетом очереди к дисковому массиву); $\varphi_M(s)$ – ПЛС времени чтения записи фрагментированной таблицы из оперативной памяти (с учетом очереди к шине памяти); $\varphi_N(s)$ – ПЛС времени межпроцессорного обмена при передаче результирующей записи по шине N; P_F – вероятность, что запись удовлетворяет условию поиска F (эта вероятность рассчитывается по известным формулам [4]) .

Будем считать, что каждое из преобразований Лапласа-Стилтьеса $\varphi_D(s)$, $\varphi_M(s)$, $\varphi_N(s)$ соответствует времени пребывания в СМО M/M/1.

Действительно, обработку в ресурсе (дисковом массиве, оперативной памяти, сети) можно описать с помощью модели ремонтника [8], где в качестве приборов выступают процессоры, а в качестве ремонтника – ресурс (шина). При достаточно большом числе процессоров эту модель можно аппроксимировать разомкнутой СМО M/G/1 при загрузке ремонтника $\rho < 1$.

Обработка записи базы данных в ресурсе разбита на кванты. Для дискового массива – это поиск в кэше файловой системы, обработка запроса адаптером ши-

ны сервера, поиск в кэше контроллера дискового массива, поиск в кэше диска, подвод головки к цилиндру, ожидание требуемого сектора, чтение сектора в кэш диска, запись сектора в кэш контроллера дискового массива, запись сектора в кэш файловой системы. Для шины оперативной памяти – это чтение/запись большого количества операндов и команд, выполняемых в процессоре при обработке записи базы данных. Для шины N – передача большого числа сообщений протокольной машины.

Известно [9], что СМО M/G/1 с дисциплиной квантования приближается по характеристикам к СМО M/M/1 при малом значении интервала квантования (по крайней мере, это справедливо для распределения числа заявок в СМО и среднего времени пребывания).

Время пребывания в СМО M/M/1 распределено по экспоненциальному закону. Поэтому для $\varphi_D(s)$, $\varphi_M(s)$, $\varphi_N(s)$ имеем

$$\varphi_D(s) = \frac{(\mu_D - n\lambda_D)(\mu_D + s)}{\mu_D(\mu_D - n\lambda_D + s)} \cdot \frac{\mu_D}{(\mu_D + s)} = \varphi_{DW}(s) \cdot \varphi_{DP}(s), \quad (2)$$

$$\varphi_M(s) = \frac{(\mu_M - n\lambda_M)(\mu_M + s)}{\mu_M(\mu_M - n\lambda_M + s)} \cdot \frac{\mu_M}{(\mu_M + s)} = \varphi_{MW}(s) \cdot \varphi_{MP}(s), \quad (3)$$

$$\varphi_N(s) = \frac{(\mu_N - n\lambda_N)(\mu_N + s)}{\mu_N(\mu_N - n\lambda_N + s)} \cdot \frac{\mu_N}{(\mu_N + s)} = \varphi_{NW}(s) \cdot \varphi_{NP}(s), \quad (4)$$

где n – число процессоров; $1/\mu$ – математическое ожидание времени обработки одной записи БД в ресурсе (диске, памяти, сети), пока индексы D, M, N в обозначениях будем опускать; $1/\lambda$ – математическое ожидание интервала времени между последовательными обращениями программы, выполняемой на одном процессоре, к ресурсу для обработки одной записи БД; $\varphi_P(s)$, $\varphi_W(s)$ – соответственно ПЛС времени обработки записи и ожидания ее обработки в ресурсе.

При этом должно выполняться неравенство

$$\rho = \frac{n\lambda}{\mu} < 1. \quad (5)$$

ПЛС времени выполнения запроса к базе данных (1) для разных конфигураций (см. рис. 5) различаются присутствием или отсутствием в них выражений $\varphi_{DW}(s)$, $\varphi_{MW}(s)$, $\varphi_{NW}(s)$. Все зависит от того, разделяемый ли в конфигурации ресурс (т.е. возможна к нему очередь) или нет. Если преобразование Лапласа-Стилтьеса отсутствует (в таблице отсутствие «–», наличие «+»), то в формулах (2) – (4) оно заменяется единицей.

	$\varphi_{DW}(s)$	$\varphi_{MW}(s)$	$\varphi_{NW}(s)$
SE	+	+	$\varphi_N(s)$ следует заменить на $\varphi_D(s)$, так как нет сети
SD	+	–	+
SN	–	–	+

Из (1) можно получить математическое ожидание и дисперсию времени выполнения запроса к БД:

$$M_\xi = -\varphi'(0), \quad (6)$$

$$M_{\xi 2} = \varphi^{(2)}(0), \tag{7}$$

$$\sigma_\xi^2 = M_{\xi 2} - M_{\xi 2}^2 . \tag{8}$$

Дифференцируя (1) как сложную функцию в нуле, получим, например, для конфигурации SE

$$M_\xi = -\varphi'(0) = \frac{V}{n}\left(\frac{1+\mathrm{P}_F}{\mu_D - n\lambda_D} + \frac{1}{\mu_M - n\lambda_M}\right). \tag{9}$$

С помощью формул (7) и (8) можно получить выражения для $M_{\xi 2}$ и σ_ξ^2.

Используя сведения, приведенные в таблице, а также формулы (2) – (4), (1), (6) – (8), можно получить математические ожидания и дисперсии времени выполнения запроса к БД для всех конфигураций, представленных на рис. 5. С помощью этих выражений можно выполнить сравнение конфигураций по стоимости с учетом ограничения на время выполнения запроса или смеси запросов.

Для каждой *i*-й конфигурации необходимо решить задачу оптимизации:

$$C_i(n) \xrightarrow[n]{} \min,$$

$$M_{\xi i} + k\sigma_{\xi i} < T_{\text{гр}}, \tag{10}$$

где $C_i(n)$ – стоимость *i*-й конфигурации с n процессорами; $M_{\xi i}+k\sigma_{\xi i}$ – верхняя граница доверительного интервала (k = 1, 2, 3); $T_{гр}$ – предельно допустимое время выполнения запроса или смеси запросов.

Оптимальной будет конфигурация с минимальным значением $C_i(n_{opti})$. Смысл задачи (10) заключается в том, что какая-то конфигурация может быть быстрой, но иметь высокую стоимость, и наоборот.

Исследуем поведение математического ожидания времени выполнения запроса к БД для конфигурации SE в зависимости от числа процессоров n (см. выражение (9)).

Предположим, что в параллельной системе баз данных "узким местом" является подсистема ввода/вывода, т.е. справедливо неравенство

$$\frac{\mu_D}{\lambda_D} << \frac{\mu_M}{\lambda_M}. \tag{11}$$

Следует отметить, что при общей оперативной памяти условие (11) может и не выполняться.

Тогда выражение (9) можно переписать в следующем виде:

$$M_\xi = \frac{V}{n}\left(\frac{1+\mathrm{P}_F}{\mu_D - n\lambda_D} + \frac{1}{\mu_M}\right). \tag{12}$$

График зависимости математического ожидания M_ξ времени выполнения запроса от числа процессоров n в параллельной системе баз данных показан на рис. 6.

Координаты точек, обозначенных на рисунке, имеют следующие значения:

$$y_1 = M_\xi(1) = V\left(\frac{1+P_F}{\mu_D - \lambda_D} + \frac{1}{\mu_M}\right), \tag{13}$$

где n^* – точка минимума,

$$n^* = \frac{\mu_D}{\lambda_D} - \frac{\mu_M}{\lambda_D}\left(\sqrt{(1+P_F)\left(\frac{\mu_D}{\mu_M}+1+P_F\right)} - (1+P_F)\right), \qquad (14)$$

$$y^* = M_\xi(n^*) = \frac{V}{n^*}\left(\frac{1+P_F}{\mu_D - n^*\lambda_D} + \frac{1}{\mu_M}\right). \qquad (15)$$

Из (14) следует, что при возрастании μ_M значение n^* убывает. Используя правило Лопиталя, можно получить

$$\lim_{\mu_M \to \infty} n^* = \frac{\mu_D}{2\lambda_D}. \qquad (16)$$

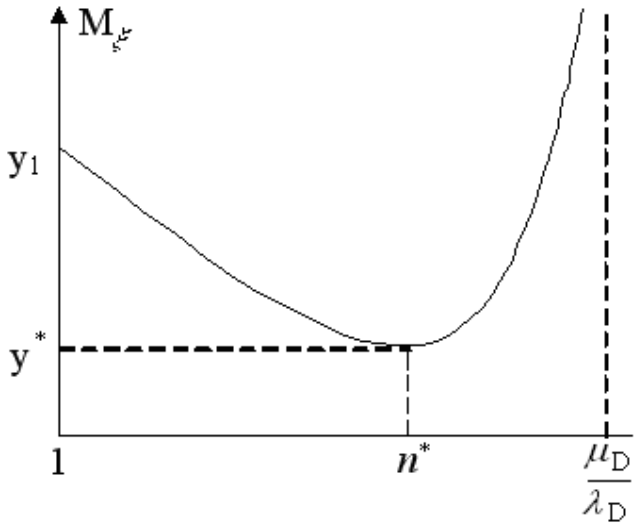

Рис. 6. График зависимости M_ξ от числа процессоров n.

Можно показать, что производная n^* по P_F меньше нуля. Поэтому значение n^* убывает с ростом P_F $(0 \le P_F \le 1)$.

Поведение функции $M_\xi(n)$ (см. рис. 6) объясняется тем, что сначала с ростом числа процессоров время убывает благодаря распараллеливанию обработки запроса, затем возрастает из-за перегрузки подсистемы ввода/вывода.

Заключение

Проанализирован существующий способ выполнения запросов в параллельной системе баз данных. СУБД выполняет фрагментацию таблиц (отношений) базы данных и пересылку фрагментов процессорам. Оптимальный последовательный план тиражируется по процессорам, т.е. создаются n идентичных параллельных агентов. Каждый агент выполняет операции с фрагментами таблиц, переданными процессору. Результаты, полученные от всех процессоров, объединяются.

Приведено преобразование Лапласа-Стилтьеса (ПЛС) времени выполнения запроса, имеющего план $\pi_A(\sigma_F(R))$, в параллельной СУБД. Рассмотрены варианты этого преобразования для различных архитектур параллельных систем баз данных, т.е. ПЛС времени ожидания освобождения ресурса входит или нет в исход-

ное преобразование Лапласа-Стилтьеса времени выполнения запроса в зависимости от того, является ли ресурс разделяемым или нет.

Предложен способ выбора архитектуры параллельной системы баз данных по критерию стоимости с ограничением на верхнюю границу доверительного интервала времени выполнения запроса или смеси запросов.

Исследована зависимость математического ожидания времени выполнения запроса к БД от числа процессоров для конфигурации SE. Показано, что сначала с ростом числа процессоров время убывает благодаря распараллеливанию обработки запроса, затем возрастает из-за перегрузки подсистемы ввода/вывода, которая является разделяемым ресурсом.

Предполагается продолжить исследования и получить оценки времени выполнения запросов с более сложными планами реализации (например, для плана соединения таблиц).

ЛИТЕРАТУРА

1. *Девитт Дэвид, Грэй Джим.* Параллельные системы баз данных: будущее высокоэффективных систем баз данных / пер. Сергея Кузнецова, 2009 г.: [Электронный ресурс]. [http://www.citforum.ru/database/classics/parallel_f]. Проверено 10.04.2009.
2. *Boral H. and DeWitt D.* Database machines: An idea whose time has passed? A critique of the future of the database machines. In Proceedings of the 1983 Workshop on Database Machines. H.-O. Leilich and M. Missikoff, Eds., Springer-Verlag, 1983.
3. *Соколинский Л.Б., Цымблер М.Л.* Лекции по курсу "Параллельные системы баз данных": [Электронный ресурс]. [http://pdbs.susu.ru/CourseManual.html]. Проверено 10.04.09.
4. Transaction Processing Performance Council (TPC): [Электронный ресурс]. [http://www.tpc.org]. Проверено 10.04.09.
5. *Dewitt David, Gray Jim.* Parallel database systems: the future of high performance database systems // Communications of the ACM. – June 1992. – Vol. 35, №. 6. – P.1-26.
6. *Григорьев Ю.А., Плутенко А.Д.* Теоретические основы анализа процессов доступа к распределенным базам данных. – Новосибирск: Наука, 2002.
7. *Соколинский Л.Б.* Обзор архитектур параллельных систем баз данных // Программирование. – 2004. - № 6. – С.1-15.
8. *Авен О.И., Гурин Н.Н., Коган Я.А.* Оценка качества и оптимизация вычислительных систем. – М.: Наука, 1982.
9. *Яшков С.Ф.* Анализ очередей в ЭВМ. – М.: Радио и связь, 1989.

E-mail:
Григорьев Ю.А. – grigorev@iu5.bmstu.ru.