Обработка запросов в СУБД для кластерных систем*

А.В. Лепихов, Л.Б. Соколинский
Южно-Уральский государственный университет

22 июня 2009 г.

Аннотация

Работа посвящена проблеме эффективной обработки запросов в кластерных вычислительных системах. Представлен оригинальный подход к размещению и репликации данных на узлах кластерной системы. На основе этого подхода разработан метод балансировки загрузки. Предложен метод эффективной параллельной обработки запросов для кластерных систем, основанный на описанном методе балансировки загрузки. Приведены результаты вычислительных экспериментов и выполнен анализ эффективности предложенных подходов.

1 Введение

На сегодняшний день существует целый ряд систем баз данных, обеспечивающих параллельную обработку запросов. Системы баз данных, предназначенные для обработки OLAP-запросов используются для управления петабайтными массивами данных. Например, СУБД Greenplum основанная на технологии MapReduce [1] выполняет глубинный анализ 6.5 Пбайт данных на 96-узловом кластере в компании eBay. СУБД Hadoop обрабатывает 2.5 Пбайт данных на кластере, состоящем из 610 узлов для популярного web-сервиса facebook. В области параллельной обработки OLTP-запросов существует ряд коммерческих параллельных СУБД среди которых наиболее известными являются Teradata, Oracle Exadata и DB2 Parallel Edition.

В настоящее время исследования в данной области ведутся в направлении самонастройки СУБД [2], балансировки загрузки и связанной с ней проблемы размещения данных [3], оптимизации параллельных запросов [4] и эффективного использования современных многоядерных процессоров [5, 6].

Одной из важнейших задач в параллельных СУБД является балансировка загрузки. В классической работе [7] было показано, что перекосы, возникающие при обработке запросов в параллельных системах баз данных без совместного использования ресурсов, могут приводить к практически полной деградации производительности системы.

*Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (проект 09-07-00241-а).

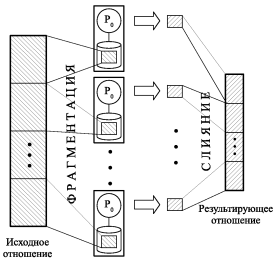

Рис. 1: Фрагментный параллелизм.

В работе [8] предложено решение проблемы балансировки загрузки для систем без совместного использования ресурсов, основанное на репликации. Данное решение позволяет уменьшить накладные расходы на передачу данных по сети в процессе балансировки загрузки. Однако этот подход применим в весьма узком контексте пространственных баз данных в специфическом сегменте диапазонных запросов. В работе [3] задача балансировки загрузки решается путем частичного перераспределения данных перед началом выполнения запроса. Данный подход уменьшает суммарное количество пересылок данных между вычислительными узлами в ходе обработки запроса, однако накладывает серьезные требования на скорость межпроцессорных коммуникаций.

В настоящей работе предложен метод параллельной обработки запросов, основанный на оригинальном подходе к размещению базы данных, названном частичным зеркалированием. Данный метод решает задачи эффективной обработки запросов и балансировки загрузки в кластерных системах.

Статья организована следующим образом. В разделе 2 описывается метод параллельной обработки запросов в СУБД для кластерных систем. В разделе 3 предлагается стратегия размещения данных на кластерных системах и алгоритм балансировки загрузки. В разделе 4 приведены результаты вычислительных экспериментов, которые позволяют оценить практическую значимость предлагаемых в работе методов и алгоритмов. В заключении суммируются основные результаты и выводы, полученные в данной статье, и намечаются направления дальнейших исследований.

2 Организация параллельной обработки запросов

Основой параллельной обработки запросов в реляционных системах баз данных является фрагментный параллелизм (см. рис. 1). Данная форма параллелизма предполагает фрагментацию отношения, являющегося аргументом реляционной операции, по дискам многопроцессорной системы. Способ фрагментации определяется функцией фрагментации ϕ, которая для каждого кортежа отношения вычисляет номер процессорного узла, на котором должен быть размещен этот кортеж. Запрос параллельно выполняется на всех процессорных узлах в виде набора параллельных агентов [9], каждый из которых обрабатывает отдельный фрагмент отношения на выделенном ему процессорном узле. Полученные агентами результаты сливаются в результирующее отношение. Несмотря на то, что каждый параллельный агент в процессе выполнения запроса независимо обрабатывает свой фрагмент отношения, для получения корректного результата необходимо выполнять пересылки кортежей.

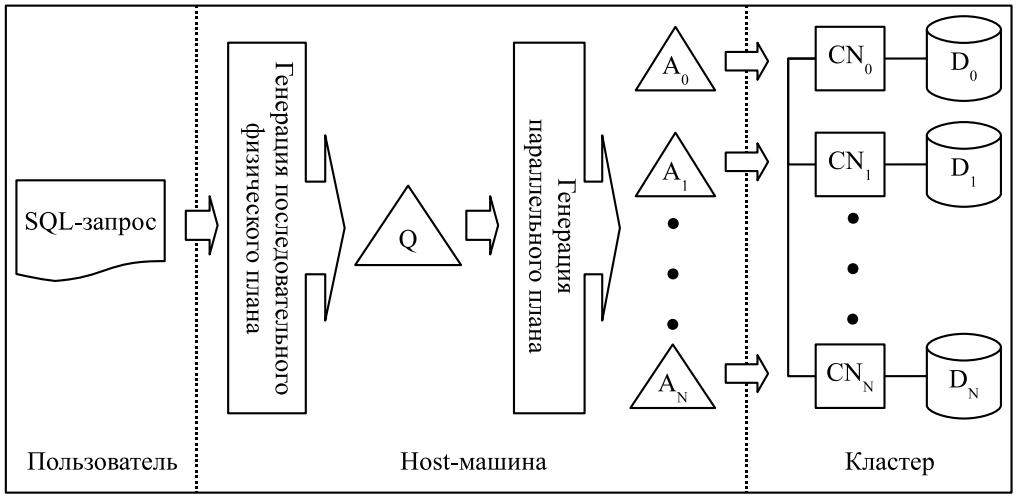

Рис. 2: Схема обработки запроса в параллельной СУБД для кластерных систем. Q – последовательный физический план, A_i – параллельный агент, CN_i – вычислительный узел.

Для организации таких пересылок в соответствующие места дерева плана запроса вставляется оператор **exchange** [10].

Оператор **exchange** идентифицируется в дереве плана своим номером и функцией распределения ψ, которая для каждого входного кортежа вычисляет номер вычислительного узла, где должен быть обработан данный кортеж. Оператор **exchange** выполняет пересылки кортежей между параллельными агентами, используя коммуникационные каналы, каждый из которых задается парой (номер узла, номер порта). При этом в качестве номера узла фигурирует номер параллельного агента, а в качестве порта – номер оператора **exchange**.

Опишем общую схему организации параллельной обработки запросов в параллельной СУБД для кластерных систем. Мы предполагаем, что вычислительная система представляет собой кластер, состоящий из N вычислительных узлов (см. рис. 2). Будем считать, что каждое отношение базы данных, задействованное в обработке запроса, фрагментировано по всем узлам вычислительной системы. В соответствии с данной схемой, обработка SQL-запроса состоит из трех этапов.

На первом этапе SQL-запрос передается пользователем на выделенную host-машину, где транслируется в некоторый последовательный физический план. На втором этапе последовательный физический план преобразуется в параллельный план, представляющий собой совокупность параллельных агентов. Это достигается путем вставки оператора обмена **exchange** в соответствующие места дерева запроса.

На третьем этапе параллельные агенты пересылаются с host-машины на соответствующие вычислительные узлы, где интерпретируются исполнителем запросов. Результаты выполнения агентов объединяются корневым оператором **exchange** на нулевом узле, откуда передаются на host-машину. Роль host-машины может играть любой узел вычислительного кластера.

Поясним цикл обработки запроса в параллельной системе баз данных на следующем примере. Пусть необходимо вычислить $Q = R \bowtie S$ – естественное соединение двух отношений R и S по некоторому общему атрибуту Y. Пусть отношение R фрагментировано по атрибуту соединения на двух вычислительных узлах CN_0 и CN_1 в

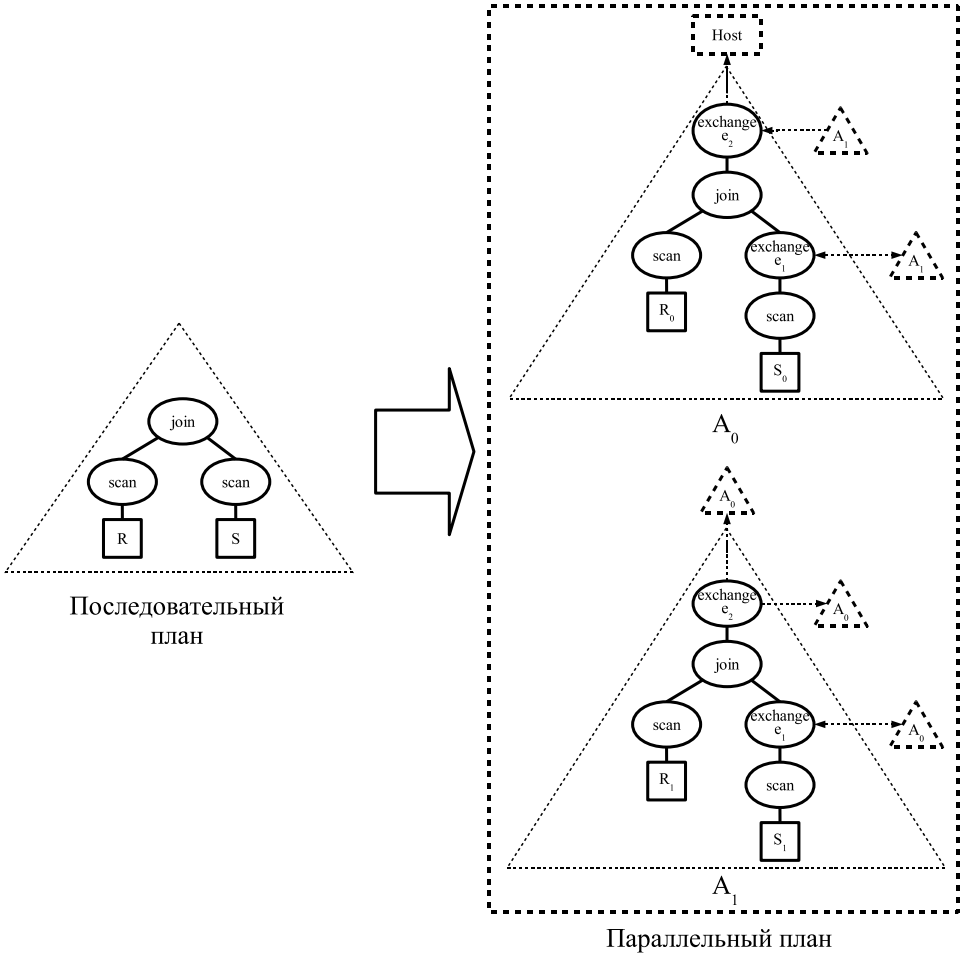

Рис. 3: Последовательный и параллельный планы для запроса $Q = R \bowtie S$.

виде двух фрагментов R_0 и R_1, т.е.

$$R = R_0 \cup R_1, \qquad R_0 \cap R_1 = \emptyset, \qquad \pi_Y(R_0) \cap \pi_Y(R_1) = \emptyset,$$

где π – операция проекции.

Обозначим $\phi : R \to \{0, 1\}$ – функция фрагментации для отношения R. Имеем

$$\forall u, v \in R : u.Y = v.Y \Rightarrow \phi(u) = \phi(v).$$

Здесь $u.Y$ и $v.Y$ обозначают значение атрибута Y в кортежах u и v соответственно. Тогда существует функция $\phi_Y : \pi_Y(R) \to \{0, 1\}$ такая, что

$$\forall r \in R : \phi(r) = \phi_Y(r.Y).$$

Пусть отношение S фрагментировано по некоторому другому атрибуту Z на тех же двух вычислительных узлах CN_0 и CN_1 в виде двух фрагментов S_0 и S_1, т.е.

$$S = S_0 \cup S_1, \qquad S_0 \cap S_1 = \emptyset, \qquad \pi_Z(S_0) \cap \pi_Z(S_1) = \emptyset, Z \neq Y.$$

Тогда последовательный физический план запроса Q и соответствующий ему параллельный план будут иметь вид, изображенный на рис. 3. Параллельный план в

данном случае включает в себя двух агентов A_0 и A_1, которые будут выполняться на вычислительных узлах CN_0 и CN_1 соответственно. Для того, чтобы естественное соединение выполнялось в параллельном плане корректно, нам необходимо вставить оператор **exchange** e_1 между оператором **join** и оператором **scan** для отношения S. При этом функция распределения для e_1 будет иметь вид

$$\psi_1(s) = \phi_Y(s.Y).$$

Для сбора кортежей результирующего отношения на узле агента A_0 после оператора **join** добавляем еще один оператор **exchange** e_2, функция распределения которого имеет вид

$$\psi_2(x) = 0.$$

3 Размещение данных и балансировка загрузки

3.1 Фрагментация и сегментация данных

Распределение базы данных в кластерной вычислительной системе задается следующим образом [11]. Каждое отношение разбивается на непересекающиеся горизонтальные фрагменты, которые размещаются на различных вычислительных узлах. При этом предполагается, что кортежи фрагмента некоторым образом упорядочены, что этот порядок фиксирован для каждого запроса и определяет последовательность считывания кортежей в операции сканирования фрагмента. Будем называть этот порядок *естественным*. На практике естественный порядок может определяться физическим порядком следования кортежей или индексом.

Каждый фрагмент на логическом уровне разбивается на последовательность сегментов фиксированной длины. Длина сегмента измеряется в кортежах и является атрибутом фрагмента. Разбиение на сегменты выполняется в соответствии с естественным порядком и всегда начинается с первого кортежа. В соответствии с этим последний сегмент фрагмента может оказаться неполным.

Количество сегментов фрагмента F обозначается как $S(F)$ и может быть вычислено по формуле

$$S(F) = \left\lceil \frac{T(F)}{L(F)} \right\rceil.$$

Здесь $T(F)$ обозначает количество кортежей во фрагменте F, $L(F)$ – длину сегмента для фрагмента F.

3.2 Репликация данных

Пусть фрагмент F_0 располагается на диске $d_0 \in \mathfrak{D}$ кластерной системы. Полагаем, что на каждом диске $d_i \in \mathfrak{D}(i > 0)$ располагается *частичная реплика* F_i, включающая в себя некоторое подмножество (возможно пустое) кортежей фрагмента F_0.

Наименьшей единицей репликации данных является сегмент. Длина сегмента реплики всегда совпадает с длиной сегмента реплицируемого фрагмента:

$$L(F_i) = L(F_0), \forall d_i \in \mathfrak{D}.$$

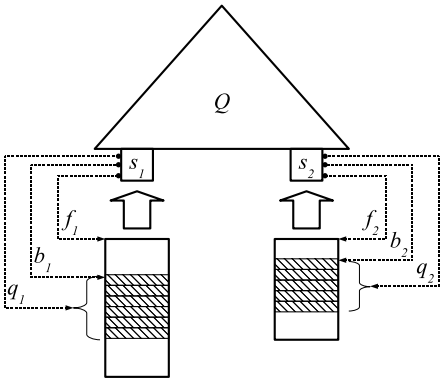

Рис. 4: Параллельный агент с двумя входными потоками.

Размер реплики F_i задается коэффициентом репликации

$$\rho_i \in \mathbb{R}, 0 \leq \rho_i \leq 1,$$

являющимся атрибутом реплики F_i, и вычисляется по следующей формуле

$$T(F_i) = T(F_0) - \lceil (1 - \rho_i) \cdot S(F_0) \rceil \cdot L(F_0).$$

Естественный порядок кортежей реплики F_i определяется естественным порядком кортежей фрагмента F_0. При этом номер N первого кортежа реплики F_i вычисляется по формуле:

$$N(F_i) = T(F_0) - T(F_i) + 1.$$

Для пустой реплики F_i будем иметь $N(F_i) = T(F_0)+1$, что соответствует признаку «конец файла».

Описанный механизм репликации данных позволяет использовать в кластерных системах простой и эффективный метод балансировки загрузки, описываемый в разделе 3.3.

3.3 Метод балансировки загрузки

3.3.1 Схема работы параллельного агента

Пусть задан некоторый запрос Q, имеющий n входных отношений. Пусть $\mathfrak{Q}$ – параллельный план запроса Q. Каждый агент $Q \in \mathfrak{Q}$ имеет n входных потоков $s_1, \ldots, s_n$. Каждый поток $s_i (i = 1, \ldots, n)$ задается четырьмя параметрами:

1. f_i – указатель на фрагмент отношения;
2. q_i – количество сегментов в отрезке, подлежащем обработке;
3. b_i – номер первого сегмента в обрабатываемом отрезке;
4. a_i – индикатор балансировки: 1 – балансировка допускается, 0 – балансировка не допускается.

На рис. 4 изображен пример параллельного агента с двумя входными потоками.

Параллельный агент Q может находиться в одном из двух состояний: *активном* и *пассивном*. В *активном* состоянии Q последовательно считывает и обрабатывает кортежи из всех входных потоков. При этом в ходе обработки динамически изменяются значения параметров q_i и b_i для всех $i = 1, \ldots, n$. В *пассивном* состоянии агент Q не выполняет никаких действий. На начальном этапе выполнения запроса выполняется инициализация агента, в результате которой происходит определение параметров всех входных потоков. Затем агент переводится в активное состояние и начинает обработку фрагментов, ассоциированных с его входными потоками. В каждом фрагменте обрабатываются только те сегменты, которые входят в отрезок, определяемый параметрами потока, ассоциированного с данным фрагментом. После того, как все назначенные сегменты во всех входных потоках обработаны, агент переходит в пассивное состояние.

3.3.2 Алгоритм балансировки загрузки

При выполнении параллельного плана запроса одни агенты могут завершить свою работу и находиться в пассивном состоянии в то время, как другие агенты будут продолжать обработку назначенных им отрезков. Тем самым возникает ситуация *перекоса*. Нами предлагается следующий алгоритм балансировки загрузки, использующий репликацию данных [11].

Пусть возникла ситуация, когда параллельный агент $\bar{Q} \in \mathfrak{Q}$ закончил обработку назначенных ему сегментов во всех входных потоках и перешел в пассивное состояние, в то время как агент $\tilde{Q} \in \mathfrak{Q}$ все еще продолжает обработку своей порции данных и необходимо произвести балансировку загрузки. Будем называть простаивающий агент $\bar{Q}$ *лидером*, а перегруженный агент $\tilde{Q}$ – *аутсайдером*. В этой ситуации будет выполняться процедура балансировки загрузки между лидером $\bar{Q}$ и аутсайдером $\tilde{Q}$, которая заключается в передаче части необработанных сегментов от агента $\tilde{Q}$ агенту $\bar{Q}$. Схема алгоритма балансировки изображена на рис. 5 (использован Си-подобный псевдокод). При балансировке загрузки используется внешняя по отношению к этой процедуре *функция балансировки* **Delta**, которая вычисляет количество сегментов соответствующего входного потока, передаваемых от аутсайдера $\tilde{Q}$ лидеру $\bar{Q}$.

Для эффективного использования описанного алгоритма балансировки загрузки необходимо решить следующие задачи.

1. При наличии простаивающих агентов-лидеров, необходимо выбрать некоторый агент-аутсайдер, который будет являться объектом балансировки. Способ выбора агента-аутсайдера будем называть стратегией выбора аутсайдера.

2. Необходимо решить, какое количество необработанных сегментов данных необходимо передать от аутсайдера лидеру. Функцию, вычисляющую это число, будем называть функцией балансировки.

3.3.3 Стратегия выбора аутсайдера

Рассмотрим многопроцессорную вычислительную систему T. Пусть $\mathfrak{Q}$ – параллельный план запроса $\mathbb{Q}$, Ψ – множество узлов вычислительной системы T, на кото-

```
/* Процедура балансировки загрузки между параллельными агентами Q̄
(лидер) и Q̃ (аутсайдер). */
ū = Node(Q̄); // указатель на узел агента Q̄.
pause Q̃; // Переводим Q̃ в пассивное состояние.
for (i=1; i<=n; i++) {

  if(Q̃.s[i].a == 1) {
    f̃_i = Q̃.s[i].f; // фрагмент, обрабатываемый агентом Q̃.
    r̄_i = Re(f̃_i, ū); // реплика фрагмента f̃_i на узле ū.
    δ_i = Delta(Q̃.s[i]); // количество передаваемых сегментов.
    Q̃.s[i].q− = δ_i;
    Q̄.s[i].f = r̄_i;
    Q̄.s[i].b = Q̃.s[i].b + Q̃.s[i].q;
    Q̄.s[i].q = δ_i;
  } else
      print("Балансировка не разрешена.");
};
activate Q̃ // Переводим Q̃ в активное состояние.
activate Q̄ // Переводим Q̄ в активное состояние.
```

Рис. 5: Алгоритм балансировки загрузки двух параллельных агентов.

рых осуществляется выполнение параллельного плана $\mathfrak{Q}$. Пусть в процессе обработки запроса в некоторый момент времени агент-лидер $\bar{Q} \in \mathfrak{Q}$, расположенный на узле $\bar{\psi} \in \Psi$, закончил свою работу и перешел в пассивное состояние. Необходимо из множества агентов параллельного плана $\mathfrak{Q}$ выбрать некоторый агент-аутсайдер $\tilde{Q} \in \mathfrak{Q}(\tilde{Q} \neq \bar{Q})$, которому будет помогать агент-лидер $\bar{Q}$. Будем предполагать, что агент $\tilde{Q}$ располагается на узле $\tilde{\psi} \in \Psi$, и что $\tilde{\psi} \neq \bar{\psi}$. Обозначим через $\tilde{\rho}$ коэффициент репликации, задающий размер реплики $\tilde{f}_i$ для фрагмента $\bar{f}_i$.

Для выбора агента-аутсайдера используется механизм рейтингов. Каждому агенту параллельного плана в процессе балансировки загрузки присваивается рейтинг, задаваемый вещественным числом. В качестве аутсайдера всегда выбирается агент, имеющий максимальный положительный рейтинг. Если таковые агенты отсутствуют, то освободившийся агент $\bar{Q}$ просто завершает свою работу. Если сразу несколько агентов имеют максимальный положительный рейтинг, то в качестве аутсайдера из их числа выбирается тот, к которому дольше всего не применялась процедура балансировки загрузки.

Для вычисления рейтинга оптимистическая стратегия использует рейтинговую функцию $\gamma : \mathfrak{Q} \to \mathbb{R}$ следующего вида:

$$\gamma(\tilde{Q}) = \tilde{a}_i \cdot sgn(\max_{1 \leq i \leq n} \tilde{q}_i - B) \cdot \tilde{\rho} \cdot \vartheta \cdot \lambda.$$

Здесь λ – некоторый положительный весовой коэффициент, регулирующий влияние коэффициента репликации на величину рейтинга; B – целое неотрицательное число, задающее нижнюю границу количества сегментов, которое можно передавать при балансировке нагрузки; $\tilde{\rho}$ – коэффициент репликации, задающий размер реплики у агента $\bar{Q}$ для фрагментов агента $\tilde{Q}$; ϑ – статистический коэффициент,

принимающий одно из следующих значений:

- (-1) – количество необработанных сегментов данных у агента-аутсайдера меньше нижней границы B;
- 0 – агент-аутсайдер не участвовал в процессе балансировки;
- *целое больше нуля* – количество успешно завершенных операций балансировки, в которых принимал участие данный агент-аутсайдер.

3.3.4 Функция балансировки

Функция балансировки Δ для каждого потока $\tilde{s_i}$ агента-аутсайдера $\tilde{Q}$ определяет количество сегментов, передаваемых агенту-лидеру $\bar{Q}$. на обработку. В простейшем случае можно положить

$$\Delta(\tilde{s_i}) = \left\lceil \frac{\min\left(\tilde{q_i}, S(\tilde{f_i}) \cdot \tilde{\rho}\right)}{N} \right\rceil,$$

где N зависит от количества параллельных агентов, участвующих в обработке запроса.

Функция $S(\tilde{f_i})$, введенная в разделе 3.1, вычисляет количество сегментов фрагмента $\tilde{f_i}$. Таким образом, функция Δ расщепляет необработанные сегменты фрагмента $\tilde{f_i}$ на N отрезков и передает от $\tilde{Q}$ к $\bar{Q}$ один из отрезков. Такая функция балансировки обеспечивает равномерное распределение нагрузки агента $\tilde{Q}$ между вновь появляющимися агентами-лидерами.

4 Вычислительные эксперименты

Для исследования предложенного метода балансировки загрузки было проведено три серии вычислительных экспериментов на базе разработанного прототипа параллельной СУБД «Омега» [12]. При проведении экспериментов были поставлены следующие цели:

- получить оценки оптимальных значений параметров для предложенного алгоритма балансировки загрузки;
- исследовать влияние алгоритма балансировки загрузки на показатель масштабируемости СУБД;
- выполнить анализ эффективности алгоритма балансировки загрузки для различных значений перекосов по атрибуту фрагментации и по атрибуту соединения;

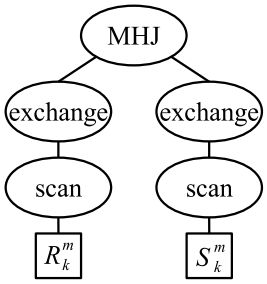

Рис. 6: Параллельный план запроса $R \bowtie S$.

4.1 Операция соединения методом хеширования в оперативной памяти

Для исследования алгоритма балансировки загрузки в прототипе параллельной СУБД «Омега» был использован алгоритм θ-соединения хешированием в оперативной памяти. Данный алгоритм используется в современных СУБД при обработке запросов в тех случаях, когда одно из соединяемых отношений целиком помещается в оперативной памяти.

Современные кластерные системы имеют значительную суммарную оперативную память. Например в рейтинге TOP50 [13] средняя суммарная оперативная память кластерной системы превышает один терабайт. В контексте приложений систем баз данных это означает, что при использовании фрагментного параллелизма отношение размером меньше терабайта может быть фрагментировано по процессорным узлам таким образом, что каждый фрагмент этого отношения целиком поместится в оперативную память. В бинарных реляционных операциях одно из отношений часто значительно превосходит другое по размеру. Применительно к операции соединения меньшее отношение называется *опорным*, а большее – *тестируемым*. В реальных приложениях баз данных случаи, когда опорное отношение имеет размер более терабайта, достаточно редки. Поэтому алгоритм соединения методом хеширования применительно к кластерным системам имеет важное практическое значение.

Алгоритм выполнения реляционной операции соединения методом хеширования в оперативной памяти можно разделить на две фазы. На *фазе построения* выполняется инициализация операции соединения и строится хеш-таблица опорного отношения R в оперативной памяти. При этом каждый процессорный узел выполняет следующие операции:

1. Выполняет покортежное сканирование своего фрагмента отношения R. Для каждого кортежа вычисляется значение функции распределения ψ_R, вырабатывающей номер узла-приемника данного кортежа. Кортеж передается на узел-приемник.

2. Строит хеш-таблицу в оперативной памяти при помощи функции хеширования $h(t)$, используя кортежи, переданные ему на шаге 1.

На *фазе сравнения* выполняется обработка тестируемого отношения S и соединение с кортежами отношения R. При этом каждый процессорный узел выполняет следующие операции:

1. Выполняет покортежное сканирование своего фрагмента отношения S. Для каждого кортежа вычисляется значение функции распределения ψ_S. Кортеж передается на узел-приемник.

2. Принимает переданный ему на шаге 1 кортеж и выполняет соединение с кортежами из хеш-таблицы, построенной на шаге 2 фазы построения. Формирует результирующий кортеж.

Заметим, что для корректной работы алгоритма функции распределения ψ_R и ψ_S должны удовлетворять условию

$$\forall r \in R, \forall s \in S : \theta(r, s) \Rightarrow \psi_R(r) = \psi_S(s).$$

На рис. 6 представлен общий вид агента параллельного плана запроса, реализующего операцию **MHJ** соединения хешированием в основной памяти. Оператор **scan** выполняет сканирование с диска. Операторы **exchange** вставляются в качестве левого и правого сына оператора **MHJ** и в процессе обработки запроса выполняет перераспределение кортежей, поступающих от операторов **scan**, между процессорными узлами.

4.2 Параметры вычислительных экспериментов

Для проведения экспериментов был использован вычислительный кластер «СКИФ Урал» [14]. Основные параметры среды выполнения экспериментов приведены в табл. 1. Каждое из отношений R и S состоит из пяти атрибутов, принимающих неотрицательные целочисленные значения в диапазоне от 0 до 10 млн. В ходе экспериментов выполнялось естественное соединение отношений R и S по второму атрибуту методом хеширования в оперативной памяти.

При выполнении запроса, в качестве опорного отношения фигурировало отношение R, а в качестве тестируемого – отношение S. Размер отношения R подбирался таким образом, чтобы любой его фрагмент целиком помещался в оперативную память вычислительного узла. Операция формирования хеш-таблицы не требует балансировки ввиду небольшого размера опорного отношения. Поэтому в экспериментах, при выполнении операции соединения методом хеширования в основной памяти балансировка производилась только на этапе обработки тестируемого отношения.

При проведении экспериментов использовалась тестовая база данных, состоящая из отношений с целочисленными атрибутами. Для формирования значений атрибута фрагментации использовалась вероятностная модель. В соответствии с данной моделью *коэффициент перекоса* θ, $(0 \leq \theta \leq 1)$ задает распределение, при котором каждому фрагменту назначается некоторый весовой коэффициент $p_i, (i = 1, \ldots, N)$

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

где N – количество фрагментов отношения, $H_N^s = 1^{-s} + 2^{-s} + \ldots + N^{-s}$ – N-е гармоническое число порядка s. Например, при $\theta = 0.5$ распределение весовых коэффициентов соответствует правилу «45-20», в соответствии с которым, 45% кортежей отношения будет храниться в 20% фрагментов. В нашем исследовании использовались значения коэффициента перекоса 0, 0.2, 0.5 и 0.68. Значение 0.68 соответствует

Таблица 1: Параметры среды выполнения экспериментов.

Параметр	**Значение**
Параметры вычислительной установки	
Количество процессорных узлов	128
Тип процессора	Intel Xeon E5472 (4 ядра по 3.0 GHz)
Объем оперативной памяти	8 ГБ/узел
Объем дисковой памяти	120 ГБ/узел
Тип коммуникационной сети	InfiniBand (20 ГБит/с)
Операционная система	SUSE Linux Enterprise Server 10
Параметры базы даных	
Размер отношения R	60 млн. записей
Размер отношения S	1.5 млн. записей
Параметры запроса	
Индикатор балансировки отношения R	0 (балансировка не допускается)
Индикатор балансировки отношения S	1 (балансировка допускается)

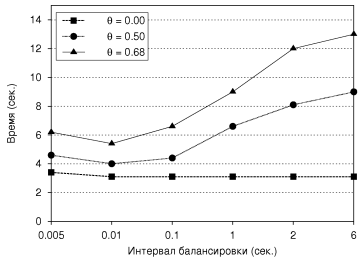

Рис. 7: Зависимость времени выполнения запроса от интервала балансировки ($n = 64, \mu = 50\%$).

правилу «80-20», значение 0.2 – правилу «25-15». Значение 0 соответствует равномерному распределению.

Для формирования значений атрибута соединения использовался коэффициент перекоса μ, в соответствии с которым кортежи подразделяются на два класса: «свои» и «чужие». Коэффициент μ указывает процентное содержание «своих» кортежей во фрагменте. «Свои» кортежи долэны быть обработаны на своем вычислительном узле. «Чужие» должны быть переданы для обработки на другие вычислительные узлы. При этом распределение значений атрибута соединения в рамках каждого из классов подчиняется равномерному закону. Например, при $\mu = 50\%$ отношение формируется таким образом, что каждый параллельный агент при выполнении соединения передает своим контрагентам по сети примерно 50% кортежей из своего фрагмента. При этом все контрагенты получат примерно одинаковое количество кортежей.

4.3 Исследование параметров балансировки загрузки

В первой серии экспериментов были исследованы следующие параметры балансировки загрузки: интервал балансировки и размер сегмента. Исследование интервала балансировки загрузки (см. рис. 7) показывает, что время выполнения запроса практически не изменяется при уменьшении интервала балансировки от 0.1 секунды до

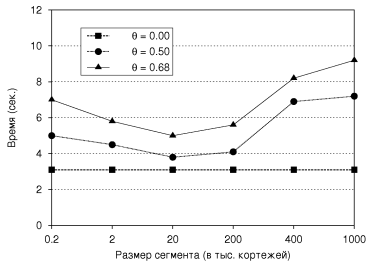

Рис. 8: Зависимость времени выполнения запроса от размера сегмента ($n = 64, \mu = 50\%$).

0.005 секунды. Это означает, что накладные расходы при увеличении частоты выполнения операции балансировки узлов не оказывают существенного воздействия на суммарное время выполнения запроса. Этот эффект достигается благодаря реализации балансировки загрузки на основе асинхронных коммуникационных функций. Вместе с этим, увеличение интервала балансировки от 0.1 секунды до 6 секунд значительно ухудшает эффективность метода. Это объясняется тем, что потенциальные возможности для балансировки используются не в полной мере: агент-лидер длительное время простаивает, ожидая появления агента-аутсайдера. Проведенные эксперименты показывают, что хорошим выбором будет использование интервала балансировки равного 0.01 секунды.

Исследование влияния размера сегмента на эффективность балансировки (см. рис. 8) показывает, что сегменты небольшого размера ухудшают показатели балансировки, поскольку существенно возрастает количество «мелких» балансировок, что ведет к росту накладных расходов на перераспределение заданий между параллельными агентами. Большие значения размера сегмента также являются неэффективными, так как в этом случае размер сегмента становится сравнимым с размером фрагмента, что делает балансировку невозможной. Таким образом, результаты экспериментов показывают, что хорошим выбором для размера сегмента будет значение равное 20 000 кортежей. Эта величина составляет примерно 0.01% от средней величины фрагмента.

4.4 Исследование влияния балансировки загрузки на время выполнения запросов

Во второй серии экспериментов исследовалось влияние метода балансировки загрузки на время обработки запросов. Эксперименты проводились с четырьмя значениями перекоса по атрибуту фрагментации на 64 процессорных узлах. Во всех испытаниях данной серии коэффициент μ равен 50%.

Результаты данных испытаний показаны на рис. 9. Данные эксперименты показывают, что оптимальным выбором является значение $\rho = 0.8$, которое позволяет практически полностью устранить негативное влияние перекосов по атрибуту фрагментации. Полная репликация не дает дополнительных преимуществ, поскольку выгода от балансировки загрузки при полной репликации перекрывается увеличением накладных расходов на выполнение балансировки и поиск отрезка данных в реплике. Вместе с этим эффективность метода балансировки загрузки значительно возрастает при увеличении коэффициента перекоса θ. При перекосе «25-15» ($\theta = 0.2$) примене-

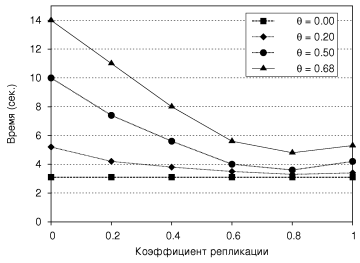

Рис. 9: Влияние коэффициента репликации ρ на время выполнения запроса ($n = 64, \mu = 50\%$).

ние балансировки загрузки позволяет сократить время выполнения запроса на 30%. В то же время, применение балансировки загрузки при перекосе «80-20» ($\theta = 0.68$) сокращает выполнение запроса на 60%. Проведенные эксперименты показывают, что метод балансировки загрузки может успешно применяться в параллельных системах баз данных для устранения дисбаланса загрузки при обработке «плохих» запросов.

4.5 Исследование масштабируемости алгоритма балансировки загрузки

Последняя серия экспериментов была посвящена исследованию масштабируемости предложенного в диссертационной работе метода балансировки загрузки. Результаты этих экспериментов представлены на рис. 10, 11 и 12.

Масштабируемость можно определить как меру эффективности распараллеливания алгоритма на многопроцессорных конфигурациях с различным количеством процессорных узлов. Одной из важнейших качественных характеристик эффективности распараллеливания является ускорение. В данной работе под ускорением понимается следующее. Пусть даны две различные конфигурации A и B параллельной системы баз данных с заданной архитектурой, различающиеся количеством процессоров и ассоциированных с ними устройств. Пусть задан некоторый тест Q. Ускорение a_{AB}, получаемое при переходе от конфигурации A к конфигурации B определяется следующей формулой

$$a_{AB} = \frac{t_{QB}}{t_{QA}},$$

где t_{QA} – время, затраченное конфигурацией A на выполнение теста Q; t_{QB} – время, затраченное конфигурацией B на выполнение теста Q.

В экспериментах, показанных на графике 10, исследовалось влияние коэффициента репликации на ускорение для коммуникационной сети Infiniband. Для эксперимента был использован перекос $\mu = 50\%, \theta = 0.5$. Результат данного эксперимента демонстрирует существенный рост ускорения обработки запроса при выполнении балансировки загрузки. Увеличение коэффициента репликации приводит к «подъему» графика ускорения, и его стремлению к линейному. Следует отметить, что для случая, когда коэффициент репликации ρ принимает значение 0.8, ускорение примерно равно линейному с коэффициентом 0.7.

В экспериментах, показанных на графике 11, исследовалось влияние перекоса по атрибуту фрагментации на ускорение при выполнении балансировки загрузки.

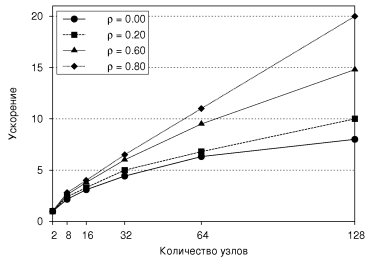

Рис. 10: Зависимость ускорения от коэффициента репликации ρ ($\theta = 0.5, \mu = 50\%$).

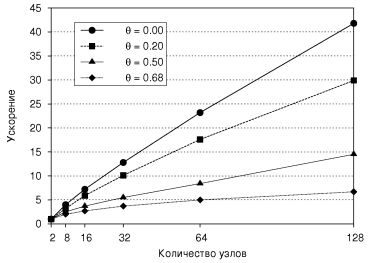

Рис. 11: Влияние перекоса по данным θ на коэффициент ускорения ($\rho = 0.50, \mu = 50\%$).

Несмотря на то, что перекосы существенно ухудшают значение ускорения, балансировка загрузки обеспечивает ощутимое ускорение даже при больших значениях коэффициента перекоса и позволяет избежать деградации производительности даже при перекосах вида «80-20» ($\theta = 0.68$).

В экспериментах, показанных на графике 12, было исследовано влияние величины коэффициента перекоса по атрибуту соединения на ускорение. Результаты данных экспериментов показывают, что даже в худшем случае, когда 80% кортежей оказываются «чужими», балансировка загрузки обеспечивает заметное ускорение.

5 Заключение

В данной работе были рассмотрены вопросы параллельной обработки запросов в многопроцессорных системах с кластерной архитектурой. Исследованы проблема балансировки загрузки и связанная с ней проблема размещения данных. Предложен подход к размещению фрагментов и реплик в кластерных системах, базирующийся

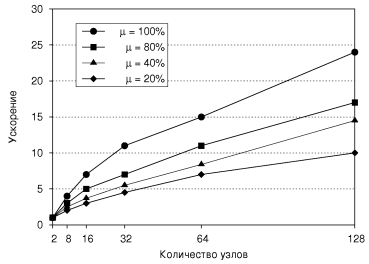

Рис. 12: Зависимость коэффициента ускорения от перекоса по значению атрибута соединения μ ($\theta = 0.5, \rho = 0.50$).

на логическом разбиении фрагмента отношения на сегменты равной длины. Описан алгоритм балансировки загрузки, в основе которого лежит метод частичного зеркалирования. Предложена стратегия выбора аутсайдера и конкретная формула для вычисления функции рейтинга. Выполнено проектирование и реализация предложенных методов и алгоритмов в прототипе параллельной СУБД «Омега». Отлаженный код системы составил более 10 000 строк на языке Си. На базе прототипа СУБД «Омега» проведены масштабные вычислительные эксперименты на кластере «СКИФ Урал». Результаты проведенных вычислительных экспериментов подтверждают эффективность предложенных методов и алгоритмов.

В качестве возможных направлений дальнейших исследований интересными представляются следующие задачи.

1. Инкапсуляция параллелизма и интеграция предложенного алгоритма балансировки загрузки в параллельной СУБД с открытым исходным кодом PostgreSQL.

2. Реализация и исследование описанного метода балансировки загрузки в СУБД для многопроцессорных иерархий, включающих в себя вычислительные кластеры на многоядерных процессорах, объединенных в среде грид.

Список литературы

[1] *Dean J., Ghemawat S.* MapReduce: simplified data processing on large clusters // Communications of ACM Vol. 51, No. 1, 2008. P. 107–113.

[2] *Chaudhuri S., Narasayya V.* Self-tuning database systems: a decade of progress // In Proceedings of the 33rd international Conference on Very Large Data Bases (Vienna, Austria, September 23 – 27, 2007). 2007. P. 3–14.

[3] *Xu Y., Kostamaa P., Zhou X., Chen L.* Handling data skew in parallel joins in shared-nothing systems // ACM SIGMOD international Conference on Management of Data Vancouver, Canada, June 09 – 12, 2008, proceedings. ACM, 2008. P. 1043–1052.

[4] *Han W., Ng J., Markl V., Kache H., Kandil M.* Progressive optimization in a shared-nothing parallel database // In Proceedings of the 2007 ACM SIGMOD international conference on Management of data (Beijing, China, June 11–14, 2007). 2007. P. 809–820.

[5] *Zhou J., Cieslewicz J., Ross K. A., Shah M.* Improving database performance on simultaneous multithreading processors // In Proceedings of the 31st international Conference on Very Large Data Bases (Trondheim, Norway, August 30 – September 02, 2005). 2005. P. 49–60.

[6] *Garcia P. Korth H. F.* Pipelined hash-join on multithreaded architectures. In Proceedings of the 3rd international Workshop on Data Management on New Hardware (Beijing, China, June 15 - 15, 2007). DaMoN '07. ACM, New York, NY, P. 1–8.

[7] *Lakshmi M.S., Yu P.S.* Effect of Skew on Join Performance in Parallel Architectures // Proceedings of the first international symposium on Databases in parallel and distributed systems, Austin, Texas, United States. IEEE Computer Society Press. 1988. P. 107–120.

[8] *Ferhatosmanoglu H., Tosun A. S., Canahuate G., Ramachandran A.* Efficient parallel processing of range queries through replicated declustering // Distrib. Parallel Databases. 2006. Vol. 20, No. 2. P. 117–147.

[9] *Костенецкий П.С., Лепихов А.В., Соколинский Л.Б.* Технологии параллельных систем баз данных для иерархических многопроцессорных сред // Автоматика и телемеханика. 2007. No. 5. С. 112–125.

[10] *Соколинский Л.Б.* Организация параллельного выполнения запросов в многопроцессорной машине баз данных с иерархической архитектурой // Программирование. 2001. No. 6. С. 13-29.

[11] *Lepikhov A. V., Sokolinsky L.B.* Data Placement Strategy in Hierarchical Symmetrical Multiprocessor Systems // Proceedings of Spring Young Researchers Colloquium in Databases and Information Systems (SYRCoDIS'2006), June 1–2, 2006. Moscow, Russia: Moscow State University. 2006. С. 31–36.

[12] Параллельная СУБД «Омега» для многопроцессорных иерархий [Сайт проекта]. URL: http://fireforge.net/projects/omega/ (дата обращения: 18.06.2009).

[13] Рейтинг TOP50: список 50 наиболее мощных компьютеров СНГ [Электронный ресурс]. URL: http://supercomputers.ru/ (дата обращения: 18.06.2009).

[14] Вычислительный кластер «СКИФ Урал» [Электронный ресурс]. URL: http://supercomputer.susu.ru/computers/ckif_ural/ (дата обращения: 18.06.2009).