PACS: 89.20.Ff

© 2007 г. П. С. Костенецкий,
А. В. Лепихов,
Л. Б. Соколинский, д-р физ.-мат. наук
(Южно-Уральский государственный университет)

ТЕХНОЛОГИИ ПАРАЛЛЕЛЬНЫХ СИСТЕМ БАЗ ДАННЫХ ДЛЯ ИЕРАРХИЧЕСКИХ МНОГОПРОЦЕССОРНЫХ СРЕД[1]

Предлагается новый подход к размещению данных и балансировке загрузки в многопроцессорных системах реляционных баз данных с иерархической архитектурой. Описана модель DMM, позволяющая моделировать и исследовать произвольные многопроцессорные иерархические конфигурации в контексте приложений класса OLTP. Рассмотрен важный подкласс многопроцессорных иерархий, названных симметричными. Для симметричных иерархий предложена новая стратегия размещения данных, базирующаяся на методе частичного зеркалирования. Получены аналитические оценки затрат дискового пространства на репликацию данных. Для симметричных иерархий, обладающих некоторой регулярностью, доказаны теоремы, дающие оценку трудоемкости формирования реплик. Предложен эффективный метод балансировки загрузки, использующий технику частичного зеркалирования. Представленные методы ориентированы на использование в кластерах и Grid-системах.

1. Введение

В настоящее время все большее распространение получают иерархические многопроцессорные архитектуры. В многопроцессорной системе с иерархической архитектурой процессорные устройства, память, диски и пр. связываются друг с другом в соответствии с некоторой иерархией. На первом уровне иерархии находятся процессорные ядра, размещенные на одном кристалле. На втором уровне находятся многоядерные процессоры, объединенные в многопроцессорные модули с общей памятью – SMP. На третьем уровне SMP-модули объединяются в кластер с помощью высокоскоростной соединительной сети. Четвертый уровень представляют *корпоративные* Grid-системы, включающие в себя несколько кластеров. Корпоративные Grid-системы могут объединяться в *кооперативные* Grid-объединения на базе Интернет. И так далее. Одним из наиболее важных приложений для многопроцессорных систем являются параллельные системы баз данных, способные хранить и обрабатывать петабайты данных [1]. Параллельным системам баз данных посвящено большое количество работ, обзор которых можно найти в [2], однако проблематика систем баз данных с иерархической многопроцессорной архитектурой до настоящего времени исследовалась мало.

В статье рассматриваются вопросы организации параллельных систем баз данных, ориентированной на эффективное использование многопроцессорных иерархий.

[1]Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (проект 06-07-89148) и Южно-Уральского государственного университета (грант 2006-112).

Выделяются следующие аспекты: моделирование иерархических архитектур, распределение данных и балансировка загрузки. Иерархические архитектуры порождают большое количество различных классов конфигураций. Некоторая классификация таких архитектур дана в [3]. Исследование подобных архитектур затруднено, так как практическое конструирование мультипроцессоров требует больших финансовых затрат, связанных с приобретением и реконфигурацией дорогостоящего оборудования.

Как следствие актуальной становится задача разработки моделей представления многопроцессорных систем баз данных, которые позволяли бы исследовать различные многопроцессорные конфигурации без их аппаратной реализации. Моделирование одноуровневых архитектур для оперативной обработки транзакций было выполнено Стоунбрейкером и Бхайдом [4,5]. В [6,7] моделируются некоторые классы двухуровневых конфигураций, однако в общем виде многопроцессорные иерархические конфигурации не исследовались. В связи с этим возникает проблема выбора оптимального класса конфигураций для определенного класса приложений баз данных. В данной работе описана модель DMM (Database Multiprocessor Model), позволяющая моделировать и исследовать произвольные многопроцессорные иерархические конфигурации в контексте приложений класса OLTP (On-Line Transaction Processing) [8].

Проблема распределения данных и связанная с ней проблема балансировки загрузки в параллельных системах баз данных без совместного использования ресурсов использовались в целом ряде работ (см., например, [9–12]), однако все эти исследования были ориентированы, главным образом, на одноуровневые многопроцессорные системы. В статье предлагается стратегия размещения данных для иерархических вычислительных систем и алгоритм балансировки загрузки, основанный на методе частичного зеркалирования. Данный алгоритм используется при обработке запросов в прототипе параллельной системы баз данных «Омега» [13].

Статья организована следующим образом. В разделе 2 приводится описание DMM-модели. Представлены модели аппаратной платформы и операционной среды, приведены стоимостная модель и модель транзакций. В разделе 3 рассматриваются стратегия размещения данных и алгоритм балансировки загрузки в иерархической вычислительной системе. Вводится формальное определение симметричной многопроцессорной иерархии. Описывается механизм репликации данных на основе сегментов. Доказываются теоремы, дающие оценки для суммарного размера реплик и трудоемкости формирования реплик при отсутствии помех. Представлен алгоритм балансировки загрузки, основанный на описанном механизме репликации. В разделе 4 суммируются полученные результаты и обсуждаются направления дальнейших исследований.

2. Моделирование иерархических архитектур

Предлагается новая модель *DMM*, позволяющая моделировать и исследовать произвольные многопроцессорные иерархические конфигурации в контексте приложений класса OLTP. Модель DMM включает в себя модели аппаратного и программного обеспечения, а также стоимостную модель.

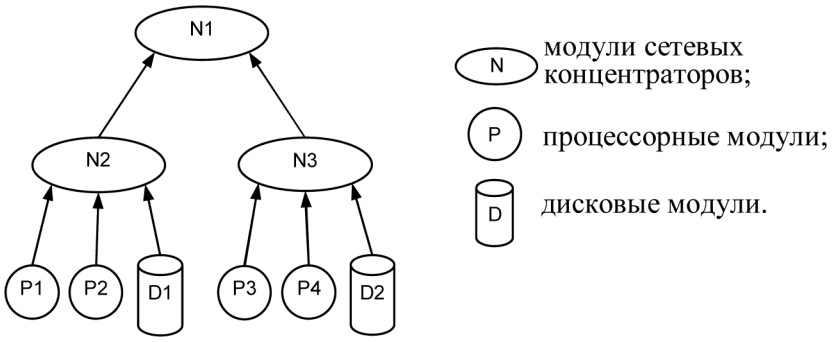

Рис. 1. Пример DM-дерева

2.1. Модель аппаратной платформы

Аппаратное обеспечение параллельной системы баз данных представляется в виде DM-дерева. *DM-дерево* — это ориентированное дерево [14], узлы которого относятся к одному из трех классов:

1) процессорные модули;

2) дисковые модули;

3) модули сетевых концентраторов.

Дуги дерева соответствуют потокам данных. *Процессорные модули* являются абстрактным представлением реальных процессорных устройств. *Дисковые модули* представляют накопители на жестких магнитных дисках. *Модули сетевых концентраторов* используются для представления произвольного *интерконнекта*, соединяющего различные процессорные и дисковые устройства. В качестве интерконнекта могут фигурировать как отдельные сетевые устройства (коммутатор, концентратор и др.), так и системная шина, соединяющая процессор с периферийными устройствами. Модель DMM не предусматривает представление модулей оперативной памяти, так как в задачах OLTP время взаимодействия процессоров и оперативной памяти несоизмеримо меньше времени обменов данными с внешними устройствами.

На структуру DM-дерева накладываются следующие ограничения:

1) корнем DM-дерева может быть только модуль сетевого концентратора;

2) дисковые и процессорные модули не могут иметь дочерних узлов, т. е. они всегда являются листьями DM-дерева.

Пример двухуровневого DM-дерева изображен на рис. 1.

2.2. Модель операционной среды

В рамках модели DMM наименьшей неделимой единицей обработки данных является *пакет*. Предполагается, что все пакеты имеют одинаковый размер. Пакет содержит заголовок, включающий в себя адрес отправителя, адрес получателя и другую вспомогательную информацию. Передача пакета может соответствовать передаче одного или нескольких кортежей в реальной системе баз данных.

Поскольку рассматриваются только приложения класса OLTP, то можно пренебречь накладными расходами на обмены между двумя процессорами через общую оперативную память и затратами на обработку данных внутри процессоров. В моде-

```
if r(P) < s_r then
    Поместить E
       с адресом P
       в очередь D;
    r(P)++;
else
    wait;
end if
```

Рис. 2. Алгоритм чтения пакета процессорным модулем

ли DMM любой процессорный модуль может обмениваться данными с любым дисковым модулем. С каждым дисковым модулем и модулем сетевого концентратора в модели DMM ассоциируется *очередь*, в которую помещаются пересылаемые пакеты.

Модель DMM допускает асинхронный обмен пакетами в том смысле, что процессорный модуль может инициализировать новый обмен, не дожидаясь завершения предыдущего. Однако будем предполагать, что процессорный модуль может иметь в каждый момент не более s_r незавершенных операций чтения и s_w незавершенных операций записи.

Время работы системы в модели DMM делится на дискретные промежутки, называемые *тактами*. Такт определяется как фиксированная последовательность шагов, семантика которых будет определена ниже.

Пусть $\mathfrak{P}$ обозначает множество всех процессорных модулей DM-дерева, $\mathfrak{D}$ — множество всех дисковых модулей, $\mathfrak{N}$ — множество всех модулей сетевых концентраторов, $\mathfrak{M} = \mathfrak{P} \cup \mathfrak{D} \cup \mathfrak{N}$ — множество всех узлов DM-дерева. Для произвольного $M \in \mathfrak{M}$ введем следующие обозначения: $F(M)$ — родительский модуль узла M, $T(M)$ — поддерево с корнем в вершине M.

Процессорный модуль $P \in \mathfrak{P}$ может инициировать операции чтения и записи пакетов. Определим их семантику следующим образом.

Операция чтения. Пусть процессорному модулю P требуется прочитать пакет E с диска $D \in \mathfrak{D}$. Если процессор P ранее инициализировал s_r еще незавершенных операций чтения, то он переводится в состояние ожидания. Если количество незавершенных операций чтения меньше s_r, то в очередь диска D помещается пакет E с адресом получателя $\alpha(E) = P$ и адресом отправителя $\beta(E) = D$. На рис. 2 представлен псевдокод данного алгоритма, где $r(P)$ — количество незавершенных операций чтения процессора P, s_r — максимальное допустимое число незавершенных операций чтения.

Операция записи. Пусть процессорному модулю P требуется записать пакет E на диск $D \in \mathfrak{D}$. На рис. 3 представлен псевдокод алгоритма, инициирующего запись.

Здесь $w(P)$ — количество незавершенных операций записи процессора P, s_w — максимальное допустимое число незавершенных операций записи.

Модуль сетевого концентратора $N \in \mathfrak{N}$ осуществляет перманентную передачу пакетов по соединительной сети, выполняя алгоритм, изображенный на рис. 4.

Здесь E – пакет, $\alpha(E)$ – адресат пакета E, $T(N)$ – поддерево с корнем N, $F(N)$ – родительский модуль узла N, $\mathfrak{P}$ – множество процессорных модулей, $r(P)$ – количе-

```
if w(P) < s_W then
     Поместить пакет E
      с адресом D в очередь
      родительского сетевого
      концентратора;
     w(P)++;
else
     wait;
end if
```

Рис. 3. Алгоритм записи пакета процессорным модулем

```
Извлечь пакет E из очереди N;
if α(E) ∉ T(N) then
     Поместить E в очередь F(N);
else
     Найти максимальное поддерево U
          дерева T(N), содержащее α(E);
     if T(α(E))=U then
          if α(E)∈𝔓 then
               r(α(E))-- ;
          else
               Поместить E в очередь α(E);
          end if
     else
          Поместить E в очередь R(U);
     end if
end if
```

Рис. 4. Алгоритм пересылки пакета сетевым концентратором

ство незавершенных операций чтения процессора P, $R(U)$ – корень поддерева U.

Дисковый модуль $D \in \mathfrak{D}$ осуществляет перманентное чтение и запись пакетов, выполняя алгоритм, изображенный на рис. 5, где $\beta(E)$ – отправитель пакета.

В модели DMM процесс обработки данных организуется в виде цикла, выполняющего стандартную последовательность шагов, называемую *тактом*. Такт определяется как следующая последовательность действий:

1) каждый модуль сетевого концентратора обрабатывает все пакеты, ожидающие передачи;

2) каждый активный процессорный модуль выполняет одну операцию чтения или записи;

3) каждый дисковый модуль обрабатывает один пакет из своей очереди.

Очевидно, что в этом случае в очереди любого концентратора не может одновременно находиться более $|\mathfrak{P}| + |\mathfrak{D}|$ пакетов, а в очереди любого диска не может одновременно находиться более $s_r s_w |\mathfrak{P}|$ пакетов.

2.3. Стоимостная модель

С каждым модулем $M \in \mathfrak{M}$ связывается *коэффициент трудоемкости* $h_M \in \mathbb{R}$, $1 \leqslant h_M < +\infty$. Так как время обработки процессором одного пакета для OLTP-приложений приблизительно в $10^5 - 10^6$ раз меньше, чем время обмена с диском или передачи по сети, то полагаем

$$h_p = 1, \ \forall P \in \mathfrak{P} .$$

Так как модуль сетевого концентратора за один такт может передавать несколько пакетов, то для каждого модуля сетевого концентратора $N \in \mathfrak{N}$ вводится функция помех

$$f_N(m_i^N) = e^{\frac{m_i^N}{\delta_N}} .$$

Здесь m_i^N обозначает число пакетов, проходящих через N на i-м такте; $\delta_N > 1$ – масштабирующий коэффициент. Таким образом, время, требуемое модулю сетевого концентратора N для выполнения i-го такта, вычисляется по формуле

$$t_i^N = h_N f_N(m_i^N), \qquad \forall N \in \mathfrak{N} .$$

Общее время работы системы, затраченное на обработку смеси транзакций (см. раздел 1.4) в течении k тактов, вычисляется по формуле

$$t = \sum_{i=1}^{k} \max(\max_{N \in \mathfrak{N}}(t_i^N), \max_{D \in \mathfrak{D}}(h_D)) .$$

2.4. Модель транзакций

Транзакция Z моделируется путем задания двух групп процессов ρ и ω: $Z = \{\rho, \omega\}$. Группа ρ включает в себя *читающие* процессы, группа ω – *пишущие* процессы. Процессы абстрагируют операции чтения (записи), выполняемые при обработке транзакций.

Количество читающих процессов определяется количеством дисков, с которых транзакция Z производит чтение данных, по одному процессу на каждый диск. Аналогичным образом, количество пишущих процессов определяется количеством дисков, на которые выполняется запись при выполнении транзакции Z. При этом если один и тот же диск используется и для чтения, и для записи, то ему сопоставляется два отдельных процесса – один пишущий и один читающий.

Модель DMM допускает выполнение на одном процессоре смеси параллельных транзакций. При этом каждая транзакция Z_i $(i = 1, \ldots, k)$ представляется своей собственной парой групп читающих и пишущих процессов: $Z_i = \{\rho_i, \omega_i\}$. Все множество процессов, моделирующих выполнение смеси транзакций, определяется следующим образом:

$$\Phi = \bigcup_{i=1}^{k} (\rho_i \cup \omega_i) .$$

Для каждого процесса $\phi \in \Phi$ задается вероятность срабатывания p_ϕ, т. е. вероятность обращения к диску, ассоциированному с этим процессом. В соответствии с этой вероятностью определяется *функция активности* $\mathbf{g}(p_\phi)$. Функция активности $\mathbf{g}(p_\phi) = G$ представляет собой функцию дискретной случайной величины G, закон распределения которой задается табл. 1.

На каждом такте работы все активные процессорные модули должны выполнять операции чтения-записи (см. раздел 2.2). В соответствии с этим активный процессорный модуль должен выбрать некоторый процесс $\phi \in \Phi$ и произвести операцию чтения с диска или записи на диск, ассоциированный с ϕ. Будем называть такой процесс *активным*. Функция активности имеет следующую семантику значений: 1 – процесс активен на данном такте, 0 – процесс не активен.

Выбор активного процесса осуществляется следующим образом. Все процессы из множества Φ организуются в циклический список. Вводится указатель на текущий элемент списка (его начальная позиция может быть произвольной). При выборе активного процесса производится циклический просмотр списка начиная с текущего элемента. Перебор прекращается, как только для какого-либо процесса из списка функция активности принимает значение 1 (см. рис. 6).

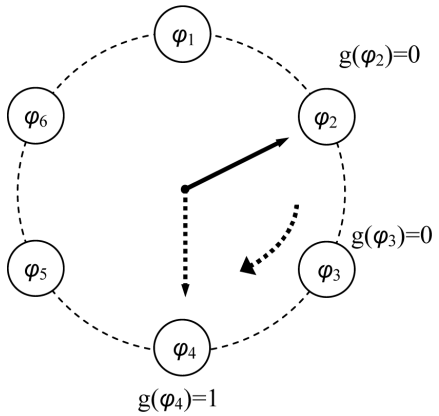

Рис. 5. Выбор активного процесса

3. Распределение данных и балансировка загрузки

Данный раздел посвящен описанию стратегии размещения данных и алгоритма балансировки загрузки в иерархической многопроцессорной системе.

3.1. Симметричные иерархии

Архитектура многопроцессорной системы зачастую является определяющим фактором при разработке стратегии размещения данных. В этом разделе строится модель симметричной иерархической многопроцессорной системы баз данных. В основе данной модели лежит понятие DM-дерева, введенное в разделе 2.1.

Дадим определение изоморфизма двух DM-деревьев. Пусть $\mathfrak{E}_T$ – множество дуг DM-дерева T; h_M – коэффициент трудоемкости узла $M \in \mathfrak{M}_T$ в DM-дереве T. DM-деревья A и B называются *изоморфными*, если существуют взаимно однозначное отображение f множества $\mathfrak{M}_A$ на множество $\mathfrak{M}_B$ и взаимно однозначное отображение g множества $\mathfrak{E}_A$ на множество $\mathfrak{E}_B$ такие, что:

(1) узел ν является конечным узлом дуги e в дереве A тогда и только тогда, когда узел $f(\nu)$ является конечным узлом дуги $g(e)$ в дереве B;

(2) узел w является начальным узлом дуги e в дереве A тогда и только тогда, когда узел $f(w)$ является начальным узлом дуги $g(e)$ в дереве B;

(3) $P \in \mathfrak{P}_A \Longleftrightarrow f(P) \in \mathfrak{P}_B$;

(4) $D \in \mathfrak{D}_A \Longleftrightarrow f(D) \in \mathfrak{D}_B$;

(5) $N \in \mathfrak{N}_A \Longleftrightarrow f(N) \in \mathfrak{N}_B$;

(6) $h_M = h_{f(M)}$.

Упорядоченную пару отображений $q = (f, g)$ будем называть *изоморфизмом* DM-дерева A на DM-дерево B.

Определим *уровень узла* по отношению к дереву T рекурсивно следующим образом [15]. Уровень корня дерева T равен нулю, а уровень любого другого узла на единицу больше, чем уровень корня минимального поддерева дерева T, содержащего данный узел.

Под *уровнем* $l(M)$ поддерева M дерева T будем понимать уровень корня поддерева M в дереве T.

Два поддерева одного уровня называются *смежными*, если в дереве существует узел, являющийся общим родителем по отношению к корневым узлам данных поддеревьев.

Определим *высоту* ориентированного дерева T как максимальный уровень поддерева в этом дереве [14].

Будем называть DM-дерево T высоты H *симметричным*, если выполняются следующие условия:

(1) любые два смежных поддерева уровня $l < H$ являются изоморфными;

(2) любое поддерево уровня $H-1$ содержит в точности один диск и один процессор.

Условие 2 в определении симметричности DM-дерева представляет собой абстрактную модель SMP-системы в том смысле, что в контексте многопроцессорных иерархий все процессоры SMP-системы могут рассматриваться как один "мегапроцессор", а все диски – как один "мегадиск". Очевидно, что балансировка загрузки на уровне SMP-системы должна решаться принципиально другими методами, так как SMP-система имеет общую память и все диски в равной мере доступны всем процессорам (см. по этому вопросу [16, 17]). Определим *степень узла* как количество дуг, входящих в этот узел. В симметричном дереве все узлы одного уровня имеют одинаковую степень, называемую *степенью данного уровня*.

3.2. Фрагментация и сегментация данных

Размещение базы данных на узлах многопроцессорной иерархической системы задается следующим образом. Каждое отношение разбивается на непересекающиеся горизонтальные фрагменты [12], которые размещаются на различных дисковых модулях. При этом предполагаем, 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 $\mathrm{S}(F)$ и может быть вычислено по формуле

$$\mathrm{S}(F) = \left\lceil \frac{\mathrm{T}(F)}{\mathrm{L}(F)} \right\rceil . \tag{1}$$

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

3.3. Репликация данных

Пусть фрагмент F_0 размещается на дисковом модуле $D_0 \in \mathfrak{D}_T$ многопроцессорной иерархической системы T. Полагаем, что на каждом дисковом модуле $D_i \in \mathfrak{D}_T$ находится *частичная реплика* F_i, включающая в себя некоторое подмножество (возможно пустое) кортежей фрагмента F.

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

$$\mathrm{L}(F_i) = \mathrm{L}(F_0), \qquad \forall D_i \in \mathfrak{D}_T.$$

Размер реплики F_i задается *коэффициентом репликации*

$$\mu_i \in \mathbb{R}, \qquad 0 \leqslant \mu_i \leqslant 1,$$

являющимся атрибутом реплики F_i, и вычисляется по формуле

$$\mathrm{T}(F_i) = \mathrm{T}(F_0) - \lceil (1 - \mu_i)\,\mathrm{S}(F_0) \rceil\, \mathrm{L}(F_0). \tag{2}$$

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

$$\mathrm{N}(F_i) = \mathrm{T}(F) - \mathrm{T}(F_i) + 1.$$

Для пустой реплики F_i будем иметь $\mathrm{N}(F_i) = \mathrm{T}(F_0)+1$, что соответствует признаку "конец файла".

Описанный механизм репликации данных позволяет использовать в многопроцессорных иерархиях простой и эффективный метод балансировки загрузки, описываемый в разделе 3.6.

3.4. Метод частичного зеркалирования

Пусть задано симметричное DM-дерево T высоты $H > 1$. Определим *функцию репликации* $r(l)$, сопоставляющую каждому уровню $l < H$ дерева T коэффициент репликации $\mu = r(l)$. Полагаем, что всегда $r(H-1) = 1$. Это мотивируется тем, что уровень иерархии $H-1$ включает в себя поддеревья высоты 1, которым соответствуют SMP-системы. В SMP-системе все диски в равной мере доступны любому процессору, поэтому нет необходимости в физической репликации данных. На логическом уровне балансировка загрузки осуществляется путем сегментирования исходного фрагмента, т. е. сам фрагмент играет роль своей реплики.

Пусть фрагмент F_0 располагается на диске $D_0 \in \mathfrak{D}_T$. Будем использовать следующий метод для построения реплики F_i на диске $D_i \in \mathfrak{D}_T$ $(i > 0)$, называемый *методом частичного зеркалирования*. Построим последовательность поддеревьев дерева T

$$\{M_0, M_1, ..., M_{H-2}\}, \tag{3}$$

обладающую следующими свойствами:

$$\begin{cases} l(M_j) = j, \\ D_0 \in \mathfrak{D}(M_j) \end{cases}$$

для всех $0 \leqslant j \leqslant H-2$. Здесь $l(M_j)$ обозначает уровень поддерева M_j.

Очевидно, что для любого симметричного дерева T существует только одна такая последовательность. Найдем наибольший индекс $j \geqslant 1$ такой, что

$$\{D_0, D_i\} \subset \mathfrak{D}(M_j).$$

Полагаем

$$\mu_i = r(j). \tag{4}$$

Для формирования реплики F_i на диске D_i будем использовать алгоритм, описанный в разделе 3.3, с коэффициентом репликации, определяемым по формуле (4).

Следующая теорема дает оценку для размера реплики.

Теорема 1. Пусть T — симметричное DM-дерево высоты H>1. Пусть фрагмент F_0 располагается на диске $D_0 \in \mathfrak{D}_T$. Пусть M — поддерево дерева T такое, что $0 < l(M) < H$ и $D_0 \in \mathfrak{D}_M$. Пусть M' – произвольное смежное с M поддерево дерева T. Тогда для любого $D_i \in \mathfrak{D}_{M'}$ справедлива следующая оценка для размера реплики F_i фрагмента F_0, размещенной на диске D_i:

$$\mathrm{T}(F_i) = r(l(M) - 1)\,\mathrm{T}(F_0) + O(\mathrm{L}(F_0)),$$

где $\mathrm{L}(F_0)$ – длина сегмента для фрагмента F_0.

Доказательство опубликовано в [18].

Заметим, что размер сегмента $\mathrm{L}(F_0)$ является параметром репликации и не связан с фрагментацией базы данных. Таким образом, можно считать, что $\mathrm{L}(F_0)$ является константой, значение которой мало относительно общего размера базы данных, и им можно пренебречь.

Оценка суммарного размера всех реплик фрагмента может быть получена с помощью следующей теоремы.

Теорема 2. Пусть T — симметричное DM-дерево высоты H>1. Пусть фрагмент F_0 располагается на диске $D_0 \in \mathfrak{D}_T$. Обозначим степень уровня l дерева T как δ_l. Обозначим через $R(F_0)$= $\sum_{i=1}^{|\mathfrak{D}_T|} \mathrm{T}(F_i)$ суммарное количество кортежей во всех репликах фрагмента F_0. Тогда

$$R(F_0) = \mathrm{T}(F_0) \sum_{j=0}^{H-2} r(j)(\delta_j - 1) \prod_{k=j+1}^{H-2} \delta_k + O(\mathrm{L}(F_0)). \tag{5}$$

Доказательство опубликовано в [18].

3.5. Выбор функции репликации

При выборе функции репликации $r(l)$ целесообразно учитывать коэффициенты трудоемкости узлов DM-дерева. Очевидно, что в симметричном DM-дереве все вершины уровня l имеют одинаковую трудоемкость $h(l)$, которую будем называть *трудоемкостью уровня l*.

Назовем симметричное DM-дерево T *регулярным*, если для любых двух уровней l и l' дерева T справедливо

$$l < l' \qquad \Rightarrow \qquad h(l) \geqslant h(l')\,, \tag{6}$$

т. е. чем выше уровень в иерархии, тем больше его трудоемкость.

Следующая теорема позволяет получить оценку трудоемкости покортежного формирования реплики в регулярном DM-дереве.

Теорема 3. Пусть T – регулярное DM-дерево высоты $H>1$. Пусть фрагмент F_0 располагается на диске $D_0 \in \mathfrak{D}_T$. Пусть M – поддерево дерева T такое, что $0<l(M)<H-1$ и $D_0 \in \mathfrak{D}_M$. Пусть M' – произвольное смежное с M поддерево дерева T; F_i – реплика фрагмента F_0, размещенная на диске $D_i \in \mathfrak{D}_{M'}$. Обозначим через $\tau(F_i)$ трудоемкость покортежного формирования реплики F_i при отсутствии помех. Тогда

$$\tau(F_i) = h\left(l(M)-1\right) r\left(l(M)-1\right) \mathrm{E}\left(F_0\right) + \mathrm{O}\left(h_0\right),$$

где $h_0 = h(0)$ – коэффициент трудоемкости корня дерева T.

Оценка трудоемкости покортежного формирования всех реплик фрагмента без учета помех может быть получена с помощью следующей теоремы.

Теорема 4. Пусть T – регулярное DM-дерево высоты $H > 1$. Пусть фрагмент F_0 располагается на диске $D_0 \in \mathfrak{D}_T$. Обозначим степень уровня l дерева T как δ_l. Обозначим через $\tau(F_0)= \sum\limits_{i=1}^{|\mathfrak{D}_T|} t(F_i)$ суммарную трудоемкость покортежного формирования всех реплик фрагмента F_0 без учета помех. Тогда

$$\tau(F_0) = \mathrm{E}\left(F_0\right) \sum_{j=0}^{H-2} h\left(j\right) r\left(j\right) \left(\delta_j - 1\right) \prod_{k=j+1}^{H-2} \delta_k + \mathrm{O}\left(h_0\right). \tag{7}$$

Доказательство опубликовано в [18].

Определим рекурсивно *нормальную* функцию репликации $r(l)$ следующим образом:

1) для $l = H-2$: $\quad r(H-2) = \dfrac{1}{h(H-2)(\delta_{H-2}-1)}$;

2) для $0 \leqslant l \leqslant H-2$: $\quad r(l) = \dfrac{r(l+1)h(l+1)(\delta_{l+1}-1)}{h(l)(\delta_l - 1)\delta_{l+1}}$.

Справедлива следующая теорема.

Теорема 5. Пусть T – регулярное DM-дерево высоты $H>1$. Пусть $\mathbb{F}$ – множество фрагментов, составляющих базу данных. Пусть $\mathbb{R}$ – множество всех реплик всех фрагментов из множества $\mathbb{F}$, построенных с использованием нормальной функции репликации. Пусть $\mathrm{T}(\mathbb{F})$ – размер базы данных в кортежах (здесь предполагается, что все кортежи имеют одинаковую длину в байтах), $\tau(\mathbb{R})$ – суммарная трудоемкость покортежного формирования всех реплик без учета помех. Тогда

$$\tau(\mathbb{R}) \approx k\,\mathrm{T}(\mathbb{F}),$$

где k – некоторая константа, не зависящая от $\mathbb{F}$.

Доказательство опубликовано в [18].

Данная теорема показывает, что при использовании нормальной функции репликации трудоемкость обновления реплик в регулярной многопроцессорной иерархической системе пропорциональна размеру обновляемой части базы данных при условии, что соединительная сеть обладает достаточной пропускной способностью.

3.6. Балансировка загрузки

Пусть задан некоторый запрос $\mathbb{Q}$, имеющий n входных отношений. Пусть $\mathfrak{Q}$ – параллельный план [19] запроса $\mathbb{Q}$. 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.

Параллельный агент Q может находиться в одном из двух состояний: *активном* или *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 Q не выполняет никаких действий. На начальном этапе обработки 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.

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 [20]. Предлагается следующий *алгоритм балансировки загрузки*, использующий репликацию данных.

Пусть имеется 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 простаивающего агента $\bar{Q}$ лидером, а перегруженного агента $\tilde{Q}$ аутсайдером. В этой ситуации будем выполнять процедуру 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. 7 (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 надо решить следующие задачи.

1. При наличии простаивающих агентов-лидеров, выбрать некоторого агента-аутсайдера, который будет являться объектом балансировки. Способ выбора агента-аутсайдера будем называть *стратегией выбора аутсайдера*.

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

```
/* Процедура балансировки загрузки между параллельными агентами Q̄
(лидер) и Q (аутсайдер). */
ū =Node(Q̄); // указатель на узел агента Q̄.
pause Q; // переводим Q в пассивное состояние.
for(i=1;i<=n;i++){
  if(Q.s[i].a==1){
     f_i = Q.s[i].f ; // фрагмент, обрабатываемый агентом Q.
     r̄_i = Re(f_i, ū); // реплика фрагмента f_i на узле ū.
     Δ_i = Delta(Q.s[i]); //     количество передаваемых сегментов.
     Q.s[i].q−=Δ_i ;
     Q̄.s[i].f = r̄_i ;
     Q̄.s[i].b = Q.s[i].b + Q.s[i].q ;
     Q̄.s[i].q = Δ_i ;
  }else
    print("Балансировка не разрешена");
};
activate Q; // переводим Q в активное состояние
activate Q̄; // переводим Q̄ в активное состояние
```

Рис. 6. Алгоритм балансировки загрузки двух параллельных агентов.

Стратегия выбора аутсайдера. Определим стратегию выбора аутсайдера, которую будем называть *оптимистической*. При этом будем опираться на метод частичного зеркалирования, описанный в разделе 3.4. Рассмотрим иерархическую многопроцессорную систему со структурой в виде симметричного DM-дерева T. Пусть $\mathfrak{Q}$ – параллельный план запроса $\mathbb{Q}$, Ψ – множество узлов DM-дерева, на которых 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 некоторого агента-аутсайдера $\tilde{Q} \in \mathfrak{Q}$ ($\tilde{Q} \neq \bar{Q}$), 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 $\tilde{\bar{M}}$ минимальное поддерево дерева T, содержащее узлы $\bar{\psi}$ и $\tilde{\psi}$.

Для выбора агента-аутсайдера используем 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:

$$\gamma(\tilde{Q}) = \tilde{a}_i \operatorname{sgn}(\max_{1 \leqslant i \leqslant n}(\tilde{q}_i) - B)\lambda\, r(l(\tilde{\bar{M}})) \sum_{i=1}^{n} \tilde{q}_i.$$

Здесь λ – некоторый положительный масштабирующий коэффициент, регулирующий влияние коэффициента репликации $r(l)$ на величину рейтинга; B – целое неотрицательное число, задающее нижнюю границу количества сегментов, которое можно передавать при балансировке нагрузки. Напомним, что $l(M)$ обозначает уровень поддерева M в дереве T (см. раздел 3.1).

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 можем положить

$$\Delta(\tilde{s}_i) = \min(\lceil \tilde{q}_i/2 \rceil, r(l(\tilde{\bar{M}}))\, \mathrm{S}(\tilde{f}_i)).$$

Функция $\mathrm{S}(\tilde{f}_i)$, введенная в разделе 3.2, вычисляет количество сегментов фрагмента $\tilde{f}_i$. Таким образом, функция Δ передает от $\tilde{Q}$ к $\bar{Q}$ половину необработанных сегментов, если только реплика фрагмента $\tilde{f}_i$ на узле агента $\tilde{Q}$ не меньше этой величины. В противном случае передается столько сегментов, сколько содержится в соответствующей реплике фрагмента $\tilde{f}_i$.

4. Заключение

Введена модель симметричной многопроцессорной иерархической системы. Эта модель описывает достаточно широкий класс реальных систем и является математическим фундаментом для определения стратегии распределения данных в многопроцессорных иерархиях. Для симметричной иерархии предложен алгоритм формирования реплик, базирующийся на логическом разбиении фрагмента отношения на сегменты равной длины. На основе этого алгоритма разработан метод частичного зеркалирования, предполагающий задание функции репликации. Функция репликации отображает уровень иерархии в коэффициент репликации, который определяет размер реплики по отношению к реплицируемому фрагменту. Доказаны теоремы, позволяющие получить оценки для размеров реплик и трудоемкости их формирования без учета помех. Предложен вариант функции репликации, при котором трудоемкость обновления реплик в многопроцессорной иерархической системе пропорциональна размеру обновляемой части базы данных при условии, что соединительная сеть обладает достаточной пропускной способностью. Описан метод балансировки загрузки, опирающийся на предложенный метод частичного зеркалирования.

В качестве направлений дальнейших исследований можно отметить следующие. Во-первых, это – аналитическое получение оценок трудоемкости обновления реплик с учетом помех. Во-вторых, планируется на базе DMM модели разработать программу, моделирующую работу иерархической многопроцессорной системы баз данных, и провести с ее помощью эксперименты по исследованию проблем распределения данных и балансировки загрузки. В-третьих, предполагается реализовать описанные методы и алгоритмы в прототипе параллельной СУБД Омега [13] для кластеров

и Grid систем. В-четвертых, планируется исследовать эффективность использования различных функций рейтинга при балансировке загрузки. При расчете функции рейтинга планируется использовать следующую статистическую информацию об истории операций балансировки загрузки агента, выступающего в роли помощника:

1) каким агентам помогал данный помощник в процессе обработки запроса;

2) сколько раз данный помощник помогал данному агенту и какое количество сегментов получил.

СПИСОК ЛИТЕРАТУРЫ

1. *Gray J., Liu D., DeWitt D. J., Heber G.* Scientific Data Management in the Coming Decade // SIGMOD Record. 2005. V. 34. No. 4. P. 34-41.

2. *Graefe G.* Query evaluation techniques for large databases // ACM Comput. Surveys. 1993. V. 25. No. 2. P. 73-169.

3. *Соколинский Л.Б.* Обзор архитектур параллельных систем баз данных // Программирование. 2004. №6. С. 49-63.

4. *Bhide A., Stonebraker M.* A Performance Comparison of Two Architectures for Fast Transaction Processing // Proc. the 4th Int. Conf. Data Engin. Los Angeles, 1988 P. 536-545.

5. *Bhide A.* An Analysis of Three Transaction Processing Architectures // Proc. 4th Int. Conf. Very Large Data Bases. Los Angeles, 1988. P. 339-350.

6. *Bouganim L., Florescu D., Valduriez P.* Dynamic Load Balancing in Hierarchical Parallel Database Systems // Proc. 22th Int. Conf. Very Large Data Bases. Mumbai, 1996. P. 436-447.

7. *Xu Y., Dandamudi S.P.* Performance Evaluation of a Two-Level Hierarchical Parallel Database System // Proc. Int. Conf. Comput. Their Appl. Tempe, 1997. P. 242-247.

8. *Девитт Д., Грэй Д.* Параллельные системы баз данных: будущее высоко эффективных систем баз данных // СУБД. 1995. №2. С. 8-31.

9. *Bitton D., Gray J.* Disk Shadowing // Proc. 4th Int. Conf. Very Large Data Bases. Los Angeles, 1988. P. 331-338.

10. *Chen S., Towsley D.F.* Performance of a Mirrored Disk in a Real-Time Transaction System // Proc. 1991 ACM SIGMETRICS Conf. Measurement and Modeling Comput. Syst. San Diego, 1991. Performance Evaluat. Rev. V. 19. No. 1. P. 198-207.

11. *Mehta M., DeWitt D.J.* Placement in Shared-Nothing Parallel Database Systems // The VLDB J. 1997. V. 6. No. 1. P. 53-72.

12. *Williams M.H., Zhou S.* Data Placement in Parallel Database Systems // Parallel database techniques / IEEE Comput. Soc. 1998. P. 203-218.

13. *Прототип параллельной СУБД Омега* [http://omega.susu.ru/prototype/]

14. *Кнут Д.Э.* Искусство программирования. Т. 3. Сортировка и поиск, 2-е изд. М.: Изд. дом «Вильямс», 2000.

15. *Кнут Д.Э.* Искусство программирования. Т. 1. Основные алгоритмы, 3-е изд. М.: Изд. дом «Вильямс», 2000.

16. *Lu H., Tan K. L.* Dynamic and Load-balanced Task-Oriented Database Query Processing in Parallel Systems // Proc 3rd Int. Conf. Extending Database Technology. Vienna, 1992. P. 357-372.

17. *Omiecinski E.* Performance Analysis of a Load Balancing Hash-Join Algorithm for a Shared Memory Multiprocessor // Proc. 17th Int. Conf. Very Large Data Bases. 1991. P. 375-385.

18. *Лепихов А.В., Соколинский Л.Б.* Стратегия размещения данных в многопроцессорных системах с симметричной иерархической архитектурой. Технический отчет OMEGA12. ЮУрГУ. 2006. [http://omega.susu.ru/reports/TR12-06-07-89148-Y1N1.pdf]

19. *Костенецкий Л.Б., Лепихов А.В., Соколинский Л.Б.* Некоторые аспекты организации параллельных систем баз данных для мультипроцессоров с иерархической архитектурой // Алгоритмы и программные средства параллельных вычислений: (Сб. науч. тр.). УрО РАН. 2006. №9 С. 42-84

20. *Maertens H.* A Classification of Skew Effects in Parallel Database Systems // Proc. 7th Int. Euro-Par Conf. 2001. P. 291-300.