Уральское отделение Российской академии наук
Институт математики и механики

Алгоритмы и программные средства параллельных вычислений — 2006. Вып. 9

НЕКОТОРЫЕ АСПЕКТЫ ОРГАНИЗАЦИИ ПАРАЛЛЕЛЬНЫХ СИСТЕМ БАЗ ДАННЫХ ДЛЯ МУЛЬТИПРОЦЕССОРОВ С ИЕРАРХИЧЕСКОЙ АРХИТЕКТУРОЙ*

П. С. Костенецкий, А. В. Лепихов, Л. Б. Соколинский

Введение

В настоящее время все большее распространение получают иерархические многопроцессорные архитектуры. В многопроцессорной системе с иерархической архитектурой процессорные устройства, память, диски и проч. 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 Grid-системы, включающие в себя несколько кластеров. Корпоративные Grid-системы могут объединяться в кооперативные Grid-объединения на базе Интернет. И так далее.

*Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (проект 06-07-89148) и Южно-Уральского государственного университета (проект 2006112).

Одним из наиболее важных приложений для многопроцессорных систем являются параллельные системы баз данных, способные хранить и обрабатывать петабайты данных [9]. Параллельным системам баз данных посвящено большое количество работ, обзор которых можно найти в [8]. Однако проблематика систем баз данных с иерархической многопроцессорной архитектурой была исследована мало.

В статье рассматриваются вопросы организации параллельных систем баз данных, ориентированной на эффективное использование многопроцессорных иерархий. Здесь мы выделяем следующие три аспекта: моделирование иерархических архитектур, распределение данных и балансировка загрузки, параллельная организация выполнения запросов.

Иерархические архитектуры порождают большое число различных классов конфигураций. Некоторая классификация таких архитектур дана в [20]. 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 всех одноуровневых архитектур для оперативной обработки транзакций было выполнено Стоунбрейкером и Бхайдом [1,2]. В работах [4, 14] 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 [15].

Проблеме распределения данных и связанной с ней проблеме балансировки загрузки в параллельных системах баз данных без совместного использования ресурсов посвящено большое количество работ (см., например, [3,5,11,13]), однако данная проблематика в контексте иерархических многопроцессорных систем до настоящего времени практически не исследовалась. В статье предлагается стратегия размещения данных для иерархических вычислительных систем и алгоритм балансировки загрузки, основанный на методе частичного зеркалирования. Алгоритм используется при обработке запросов в прототипе параллельной системы баз данных "Омега" [18].

Еще одной серьезной проблемой является построение механизма распараллеливания запросов, масштабируемого вверх по иерархии. В данной статье описана новая модель распараллеливания запросов, ориентированная на иерархические многопроцессорные архитектуры. Предложенная модель позволяет выполнять на мультипроцессорной системе много различных запросов одновременно. Каждый из этих запросов может быть автоматически распараллелен на любом уровне многопроцессорной иерархии. Это достигается путем введения специального оператора обмена, инкапсулирующего в себе все механизмы, необходимые для реализации внутриоперационного параллелизма.

Статья организована следующим образом. В разделе 1 приводится описание DMM модели. Представлены модели аппаратной платформы и операционной среды, приведены стоимостная модель и модель транзакций. В разделе 2 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 3 обсуждается организация параллельной обработки запросов применительно к иерархическим многопроцессорным системам баз данных с использованием разработанного нами алгоритма балансировки загрузки. В разделе 4 суммируются полученные результаты и обсуждаются направления дальнейших исследований.

1. Моделирование иерархических архитектур

Данный раздел посвящен моделированию многопроцессорных иерархических конфигураций. Предлагается новая модель *DMM (Database Multiprocessor Model)*, 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.

1.1. Модель аппаратной платформы

Аппаратное обеспечение параллельной системы баз данных представляется в виде *DM*-дерева. *DM-дерево* — это ориентированное дерево [17], 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 *ин-*

терконнекта, 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.

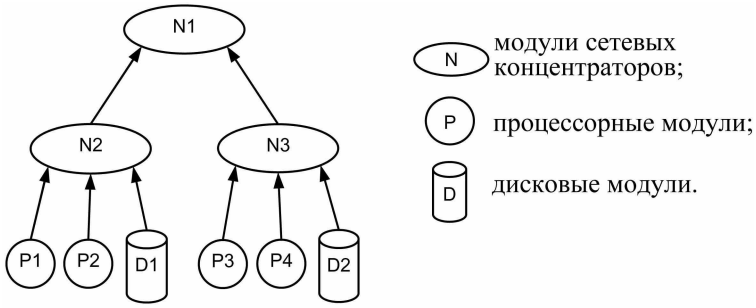

Рис. 1: Пример DM-дерева.

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.

1.2. Модель операционной среды

В рамках модели DMM наименьшей неделимой единицей обработки данных является *пакет*. Мы предполагаем, что все пакеты одного размера и имеют заголовок. Заголовок пакета

включает в себя адреса отправителя и получателя, а также другую вспомогательную информацию. Передача пакета может соответствовать передаче одного или нескольких кортежей в реальной системе баз данных.

Поскольку рассматриваются только OLTP приложения, мы можем пренебречь накладными расходами на обмены между двумя процессорами через общую оперативную память и затратами на обработку данных внутри процессоров. В модели 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 мы предполагаем, 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 ещё 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.

```
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.

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.

```
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.

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, $F(N)$ — роди-

```
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.

тельский модуль узла N, $\mathfrak{P}$ — множество процессорных модулей, $r(P)$ — количество 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.

```
Извлечь пакет E из очереди D;
if α(E)∈𝔇 then
       w(β(E))-- ;
else
       Поместить E в очередь родительского узла;
end if
```

Рис. 5: Алгоритм пересылки пакета дисковым модулем.

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 — более $s_r s_w |\mathfrak{P}|$ пакетов.

1.3. Стоимостная модель

С каждым модулем $M \in \mathfrak{M}$ связывается *коэффициент трудоемкости* $h_M \in \mathbb{R}$, $1 \le h_M < +\infty$. 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 $N \in \mathfrak{N}$ мы вводим функцию 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. п. 1.4) в течении k тактов, вычисляется по формуле

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

1.4. 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 $(i = 1..k)$ 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* $g(p_\phi)$. Функция активности $g(p_\phi) = G$ представляет собой функцию дискретной случайной величины G, закон распределения которой задаётся таблицей

g	1	0
p	p_ϕ	$1 - p_\phi$

.

На каждом такте работы все активные процессорные модули должны выполнять операции чтения-записи (см. п. 1.2). 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. Мы будем называть такой процесс *активным*. Функция активности имеет следующую семантику значений: 1 — процесс активен на данном такте, 0 — процесс не активен.

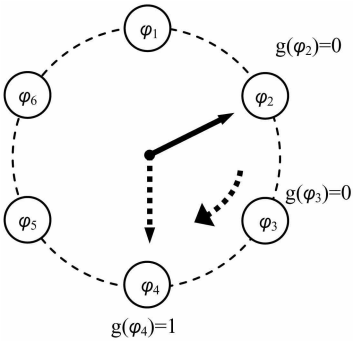

Рис. 6: Выбор активного процесса.

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).

2. Распределение данных и балансировка загрузки

Данный раздел посвящен описанию стратегии размещения данных и алгоритма балансировки загрузки в иерархической вычислительной системе. В разделе 2.1 вводится формальное определение симметричной многопроцессорной иерархии.

Раздел 2.2 посвящен описанию механизма фрагментации и балансировки загрузки, основанному на введении понятия сегмента. В разделе 2.4 представлен алгоритм балансировки загрузки, основанный на механизме репликации, описанном в 2.3.

2.1. Симметричные иерархии

Архитектура многопроцессорной системы зачастую является определяющим фактором при разработке стратегии размещения данных. В этом разделе мы построим модель симметричной иерархической многопроцессорной системы баз данных. В основе данной модели лежит понятие DM-дерева, введенное в разделе 1.1.

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.

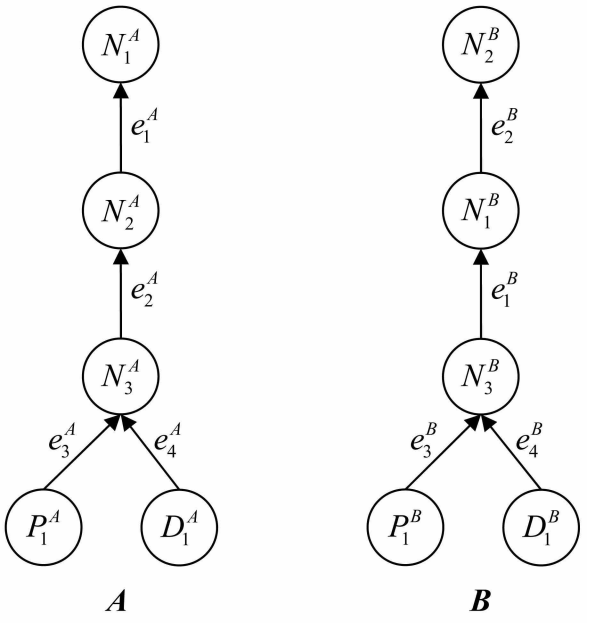

Рис. 7: Пример отображения, не являющегося изоморфизмом: $f(n_i^A) = n_i^B$, $f(p_i^A) = p_i^B$, $f(d_i^A) = d_i^B$, $g(e_i^A) = e_i^B$.

Заметим, что условие 2 из определения изоморфизма не является избыточным. Это подтверждается примером, изображенным на рис. 7. Действительно, используя нумерацию узлов и дуг, указанную на рис. 7, определим отображения f и g следующим образом (для любого i):

$$f(N_i^A) = N_i^B, \quad f(P_i^A) = P_i^B, \quad f(D_i^A) = D_i^B, \quad g(e_i^A) = e_i^B.$$

Очевидно, что отображения f и g удовлетворяют всем требованиям изоморфизма, кроме условия 2 (мы полагаем коэффициент трудоемкости для всех узлов равным 1). Однако мы не можем признать $q = (f, g)$ изоморфизмом DM-дерева A на DM-дерево B, так как здесь нарушается отношение подчиненности узлов (узел w является *подчиненным* по отношению к узлу ν, если существует дуга, направленная от w к ν). Действительно, в DM-дереве A узел N_2^A подчинен узлу N_1^A, а в DM-дереве B узел $f(N_2^A) = N_2^B$ имеет в подчинении узел $f(N_1^A) = N_1^B$.

Определим *уровень узла* по отношению к дереву T рекурсивно следующим образом [16]. 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 поддерева* дерева T мы будем понимать уровень корня этого поддерева 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 [17].

Мы 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 работы [10, 12]). Определим *степень узла* как количество дуг, входящих в этот узел. В симметричном дереве все узлы одного уровня имеют одинаковую степень, называемую *степенью данного уровня*.

2.2. Фрагментация и сегментация данных

Размещение базы данных на узлах многопроцессорной иерархической системы задается следующим образом. Каждое отношение разбивается на непересекающиеся фрагменты, которые размещаются на различных дисковых модулях. При этом мы используем горизонтальную фрагментацию [13] и предполагаем, что кортежи фрагмента некоторым образом упорядочены, что этот порядок фиксирован для каждого запроса и определяет последовательность считывания кортежей в операции сканирования фрагмента. Мы 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

$$S(F) = \left\lceil \frac{E(F)}{L(F)} \right\rceil . \qquad (1)$$

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

2.3. Репликация данных

Раздел посвящен описанию механизма репликации данных на основе сегментов. В п. 2.3.1 вводится понятие коэффициента репликации. В п. 2.3.2 описывается метод частичного зеркалирования, использующий функцию репликации, сопоставляющую каждому уровню иерархии определенный коэффициент репликации. Доказываются теоремы, дающие оценки для суммарного размера реплик. П. 2.3.3 посвящен проблеме выбора функции репликации. Вводится понятие регулярной иерархии. Доказываются теоремы, дающие оценки для трудоемкости формирования реплик при отсутствии помех.

2.3.1. Алгоритм построения реплики

Пусть фрагмент F_0 размещается на дисковом модуле $D_0 \in \mathfrak{D}_T$ многопроцессорной иерархической системы T, и на каждом дисковом модуле $D_i \in \mathfrak{D}_T$ $(i > 0)$ 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:

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

Количество кортежей $E(F_i)$ в реплике F_i задается *коэффициентом репликации*

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

являющимся атрибутом реплики F_i, и вычисляется по формуле

$$E(F_i) = E(F_0) - \lceil (1 - \mu_i) \cdot S(F_0) \rceil \cdot L(F_0). \qquad (2)$$

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

$$C(F_i) = E(F) - E(F_i) + 1.$$

Для пустой реплики F_i будем иметь $C(F_i) = E(F_0) + 1$, что соответствует признаку "конец файла".

2.3.2. Метод частичного зеркалирования

Пусть заданы симметричное DM-дерево T высоты $H > 1$ и функция репликации $r(l)$, сопоставляющая каждому уровню $l < H$ дерева T коэффициент репликации $\mu = r(l)$.

Мы полагаем, что $r(H-1) = 1$. 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.

Определим функцию репликации $r(l)$ для значений $l \leq H-2$.

Пусть фрагмент F_0 располагается на диске $D_0 \in \mathfrak{D}_T$. Мы 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

$$\{M_0, M_1, ..., M_{H-2}\}, \qquad (3)$$

обладающую следующими свойствами:

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

для всех $0 \le j \le H-2$. Здесь $l(M_j)$ обозначает уровень поддерева M_j. Очевидно, что для любого симметричного дерева T существует только одна такая последовательность. Найдем наибольший индекс $j \ge 1$ такой, что

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

Мы полагаем

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

Для формирования реплики F_i на диске D_i мы используем алгоритм, описанный в п. 2.3.1 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:*

$$E(F_i) = r(l(M)-1) \cdot E(F_0) + O(L(F_0)),$$

где $L(F_0)$ — длина сегмента для фрагмента F_0.

Доказательство. В соответствии с (2) имеем

$$E(F_i) = E(F_0) - \lceil (1-\mu(F_i)) \cdot S(F_0) \rceil \cdot L(F_0).$$

Отсюда получаем

$$E(F_i) = E(F_0) - (1-\mu(F_i)) \cdot S(F_0) \cdot L(F_0) + O(L(F_0)). \tag{5}$$

Так как $D_0 \in \mathfrak{D}_M$, $D_i \in \mathfrak{D}_{M'}$ и поддеревья M и M' являются смежными, то минимальное поддерево $\hat{M}$, содержащее диски D_0 и D_i будет иметь уровень $l(\hat{M}) = l(M) - 1$. Тогда в соответствии с (4) получаем $\mu(F_i) = r(l(M) - 1)$. Подставив это значение в (5), будем иметь

$$E(F_i) = E(F_0) - (1 - r(l(M) - 1)) \cdot S(F_0) \cdot L(F_0) + O(L(F_0)).$$

Подставив вместо $S(F_0)$ значение из (1), получим

$$\begin{aligned}
\mathrm{E}(F_i) = \mathrm{E}(F_0) - (1 - r\,(l(M) - 1)) \cdot \left\lceil \frac{\mathrm{E}(F_0)}{\mathrm{L}(F_0)} \right\rceil \cdot \mathrm{L}(F_0) + \\
+ \mathrm{O}\,(\mathrm{L}(F_0)) = \\
= \mathrm{E}(F_0) - (1 - r\,(l(M) - 1)) \cdot \frac{\mathrm{E}(F_0)}{\mathrm{L}(F_0)} \cdot \mathrm{L}(F_0) + \mathrm{O}\,(\mathrm{L}(F_0)) = \\
= \mathrm{E}(F_0) - (1 - r\,(l(M) - 1))\,\mathrm{E}(F_0) + \mathrm{O}\,(\mathrm{L}(F_0)) = \\
= r\,(l(M) - 1)\,\mathrm{E}(F_0) + \mathrm{O}\,(\mathrm{L}(F_0))\,.
\end{aligned}$$

Т е о р е м а д о к а з а н а.

Заметим, что размер сегмента $L(F_0)$ является параметром репликации и не связан с фрагментацией базы данных. Таким образом, мы можем считать, что $L(F_0)$ 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 $R(F_0)$= $= \sum_{i=1}^{|\mathfrak{D}_T|} E(F_i)$ — суммарное количество кортежей во всех репликах фрагмента F_0. Тогда*

$$R(F_0) = E(F_0) \cdot \sum_{j=0}^{H-2} r(j)(\delta_j - 1) \prod_{k=j+1}^{H-2} \delta_k + O(L(F_0)). \qquad (6)$$

Доказательство. Доказательство проведем индукцией по высоте H дерева T.

Пусть $H = 2$. Тогда число дисков в дереве T равно δ_0. В соответствии с теоремой 1 каждая реплика будет иметь размер $T_0(F_0) = r(0)E(F_0) + O(L(F_0))$. Следовательно, суммарное количество кортежей во всех репликах фрагмента F_0 имеет следующую оценку

$$\begin{aligned} \mathrm{R}(F_0) &= (\mathrm{E}\,(F_0)\,r\,(0) + \mathrm{O}\,(\mathrm{L}(F_0))) \cdot (\delta_0 - 1) = \\ &= \mathrm{E}\,(F_0)\,r\,(0)\,(\delta_0 - 1) + \mathrm{O}\,(\mathrm{L}\,(F_0))\,, \end{aligned} \qquad (7)$$

что согласуется с (6) при $H = 2$. Пусть $H > 2$. Тогда дерево T содержит δ_0 поддеревьев высоты $H - 1$:

$$M_0, M_1, ..., M_{\delta_0 - 1}.$$

Обозначим через $R_j(F_0)$ суммарное количество кортежей во всех репликах фрагмента F_0, расположенных на всех дисках поддерева M_j. Мы имеем

$$R(F_0) = \sum_{j=0}^{\delta_0 - 1} R_j(F_0). \qquad (8)$$

Без ограничения общности можно считать, что $D_0 \in \mathfrak{D}(M_0)$. Тогда в силу симметричности дерева T из (8) получаем

$$R(F_0) = R_0(F_0) + (\delta_0 - 1)R_1(F_0). \qquad (9)$$

В соответствии с теоремой 1 любая реплика F_i, располагающаяся в поддереве M_1, имеет следующий размер

$$E(F_i) = r(0)E(F_0) + O(L(F_0)). \tag{10}$$

В силу симметричности дерева T, суммарное количество дисков в поддереве M_1 равно $\prod_{k=1}^{H-2} \delta_k$. Учитывая этот факт, из (10) получаем

$$\mathrm{R}_1(F_0) = r\,(0)\,\mathrm{E}\,(F_0) \prod_{k=1}^{H-2} \delta_k + \mathrm{O}\,(\mathrm{L}(F_0))\,. \tag{11}$$

С другой стороны, по предположению индукции имеем

$$\mathrm{R}(F_0) = \mathrm{E}(F_0) \sum_{j=1}^{H-2} r(j)(\delta_j - 1) \prod_{k=j+1}^{H-2} \delta_k + \mathrm{O}\,(\mathrm{L}(F_0))\,. \tag{12}$$

Подставляя в (9) значения правых частей из (11) и (12), имеем

$$\begin{aligned}
\mathrm{R}(F_0) &= \mathrm{E}(F_0) \sum_{j=1}^{H-2} r(j) \prod_{k=j+1}^{H-2} \delta_k + \\
&+ (\delta_0 - 1) r\,(0)\,\mathrm{E}\,(F_0) \prod_{k=1}^{H-2} \delta_k + \mathrm{O}\,(\mathrm{L}(F_0)) = \\
&= \mathrm{E}(F_0) \sum_{j=0}^{H-2} r(j)(\delta_j - 1) \prod_{k=j+1}^{H-2} \delta_k + \mathrm{O}\,(\mathrm{L}(F_0))\,.
\end{aligned}$$

Т е о р е м а д о к а з а н а.

2.3.3. 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 мы будем называть *трудоемкостью уровня* l.

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

$$l < l' \qquad \Rightarrow \qquad h(l) \geqslant h(l')\,, \tag{13}$$

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 $\tau(F_i)$ — трудоемкость покортежного формирования реплики F_i при отсутствии помех. Тогда*

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

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

Д о к а з а т е л ь с т в о. Организуем конвейерную передачу кортежей с диска $D_0 \in \mathfrak{D}(M')$ в соответствии с моделью операционной среды, описанной в разделе 1.2. Скорость работы конвейера определяется самым медленным узлом. Так как M и M' являются смежными поддеревьями, их корневые узлы имеют общего родителя уровня $l(M-1)$, который в соответствии с (13) и будет самым медленным звеном конвейера. Следовательно, трудоемкость передачи одного кортежа при полностью запущенном конвейере равна $h(l(M)-1)$. Отсюда

$$\tau(F_i) = h\left(l(M)-1\right) \mathrm{E}(F_i) + \mathrm{O}\left(h\left(l(M)-1\right)\right). \tag{14}$$

Здесь $O(h(l(M)-1))$ обозначает верхнюю границу для времени, необходимого для полного “разгона” конвейера в предположении, что высота дерева T является константой.

Так как T регулярно, то $h(l(M)-1) \leq h_0$. На основании этого из (14) получаем

$$\tau(F_i) = h\left(l(M)-1\right)\mathrm{E}(F_i) + \mathrm{O}\left(h_0\right). \tag{15}$$

По теореме 1 из (15) получаем

$$\begin{gathered}\tau(F_i) = h\left(l(M)-1\right)\cdot r\left(l(M)-1\right)\mathrm{E}\left(F_0\right) + \\ + \mathrm{O}\left(h_0\right)\mathrm{O}\left(\mathrm{L}(F_0)\right) + \mathrm{O}\left(h_0\right).\end{gathered} \tag{16}$$

Мы вправе считать, что длина сегмента не меняется в процессе формирования реплики, т.е. $L(F_0)$ является константой. Тогда из (16) получаем

$$\begin{gathered}\tau(F_i) = h\left(l(M)-1\right)\cdot r\left(l(M)-1\right)\mathrm{E}\left(F_0\right) + \mathrm{O}\left(h_0\right) + \mathrm{O}\left(h_0\right) = \\ = h\left(l(M)-1\right)\cdot r\left(l(M)-1\right)\mathrm{E}\left(F_0\right) + \mathrm{O}\left(h_0\right).\end{gathered}$$

Т е о р е м а д о к а з а н а.

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 $\tau(F_0) = \sum\limits_{i=1}^{|\mathfrak{D}_T|} t(F_i)$ — суммарная трудоемкость покортежного формирования всех реплик фрагмента F_0 без учета помех. Тогда*

$$\tau(F_0) = \mathrm{E}\left(F_0\right)\sum_{j=0}^{H-2} h\left(j\right) r\left(j\right)\left(\delta_j - 1\right)\prod_{k=j+1}^{H-2}\delta_k + \mathrm{O}\left(h_0\right). \tag{17}$$

Д о к а з а т е л ь с т в о. Доказательство проведем индукцией по высоте H дерева T.

Пусть $H = 2$. Тогда число дисков в дереве T равно δ_0. В соответствии с теоремой 3 трудоемкость покортежного формирования любой реплики фрагмента F_0 в этом случае имеет следующую оценку:

$$\tau(F_i) = h\,(0) \cdot r\,(0)\,\mathrm{E}\,(F_0) + \mathrm{O}\,(h_0)\,.$$

Следовательно, суммарная трудоемкость покортежного формирования всех реплик фрагмента F_0 без учета помех может быть оценена следующим образом:

$$\tau(F_i) = h\,(0)\,r\,(0)\,\mathrm{E}\,(F_0)\,(\delta_0 - 1) + \mathrm{O}\,(h_0)\,,$$

что согласуется с (17) при $H = 2$.

Пусть $H > 2$. Тогда дерево T содержит δ_0 поддеревьев высоты $H - 1$:

$$M_0, M_1, ..., M_{\delta_0 - 1}.$$

Обозначим через $\tau_j(F_0)$ суммарную трудоемкость формирования без учета помех всех реплик фрагмента F_0, расположенных на всех дисках поддерева M_j. Мы имеем

$$\tau(F_0) = \sum_{j=0}^{\delta_0 - 1} \tau_j(F_0). \tag{18}$$

Без ограничения общности можно считать, что $D_0 \in \mathfrak{D}(M_0)$. Тогда в силу симметричности дерева T из (18) получаем

$$\tau(F_0) = \tau_0(F_0) + (\delta_0 - 1)\tau_1(F_0). \tag{19}$$

В соответствии с теоремой 3 для любой реплики F_i, располагающейся в поддереве M_1, имеем

$$\tau(F_i) = h(0) r\,(0)\,\mathrm{E}\,(F_0) + \mathrm{O}\,(h_0)\,. \tag{20}$$

В силу симметричности дерева T суммарное количество дисков в поддереве M_1 равно $\sum\limits_{k=1}^{H-2} \delta_k$. Учитывая этот факт, из (20) получаем

$$\tau_1(F_0) = h(0) r\,(0)\, \mathrm{E}\,(F_0) \prod_{k=1}^{H-2} \delta_k + \mathrm{O}\,(h_0)\,. \qquad (21)$$

С другой стороны, по предположению индукции имеем

$$\tau_0(F_0) = \mathrm{E}\,(F_0) \sum_{j=1}^{H-2} h\,(j)\, r\,(j)\,(\delta_j - 1) \prod_{k=j+1}^{H-2} \delta_k + \mathrm{O}\,(h_0)\,. \qquad (22)$$

Подставляя в (19) значения правых частей из (21) и(22), имеем

$$\begin{aligned}
\tau(F_0) &= \mathrm{E}\,(F_0) \sum_{j=1}^{H-2} h\,(j)\, r\,(j)\,(\delta_j - 1) \prod_{k=j+1}^{H-2} \delta_k + \\
&+ (\delta_0 - 1) h(0) r\,(0)\, \mathrm{E}\,(F_0) \prod_{k=1}^{H-2} \delta_k + \mathrm{O}\,(h_0) = \\
&= \mathrm{E}\,(F_0) \sum_{j=0}^{H-2} h\,(j)\, r\,(j)\,(\delta_j - 1) \prod_{k=j+1}^{H-2} \delta_k + \mathrm{O}\,(h_0)\,.
\end{aligned}$$

Т е о р е м а д о к а з а н а.

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

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

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

Справедлива следующая теорема, доказываемая непосредственно из теоремы 4 и определения нормальной функции репликации.

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 $E(\mathbb{F})$ — размер базы данных в кортежах* (здесь мы предполагаем, что все кортежи имеют одинаковую длину в байтах). $\tau(\mathbb{R})$ — суммарная трудоемкость покортежного формирования всех реплик без учета помех. Тогда

$$\tau(\mathbb{R}) \approx kE(\mathbb{F}),$$

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

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.

2.4. Алгоритм балансировки загрузки

При обработке запроса часто возникают ситуации, когда наибольшая доля работы выпадает на один процессорный модуль или целый сегмент вычислительной системы. В таких случаях после высвобождения менее загруженные процессоры берут на себя часть работы. В основе предлагаемого в данной статье алгоритма балансировки загрузки лежит метод зеркалирования [19]. Использование зеркалирования позволяет избежать пересылок дисковых данных по сети. Для обработки удаленный процессор использует реплику ближайшего к загруженному процес-

сору диска. Процессорному модулю, участвующему в балансировке, выделяется набор сегментов реплики, а не реплика целиком, что позволяет участвовать в балансировке сразу нескольким процессорным модулям.

Пусть T — симметричное DM-дерево высоты H, $P_1, ..., P_m$ — процессорные модули T, $Q_1(F_1, ..., F_l), ..., Q_m(F_1, ..., F_l)$ — параллельные агенты физического плана запроса. Каждый из входных блоков данных агента характеризуется четверкой параметров:

(1) p — указатель на фрагмент;

(2) f — номер начального сегмента;

(3) q — количество обрабатываемых сегментов;

(4) c — текущий обрабатываемый сегмент.

Параллельный агент может находиться в одном из двух состояний: выполнение и ожидание. В состоянии ожидания происходит инициализация параметров агента. В состоянии выполнения агент занимается обработкой входных данных. Данные выделяются агенту сегментами, в соответствии с естественным порядком следования кортежей. Перед началом обработки каждого сегмента агент Q_i устанавливает параметр c равным порядковому номеру текущего обрабатываемого сегмента. Выражение $c = f + q$, является условием перехода агента Q_i в состояние ожидания.

Перед началом обработки запроса каждый из входных блоков данных параллельного агента Q_i инициализируется следующими значениями:

(1) p — указатель на фрагмент F_i, расположенный на диске D_i процессорного модуля P_i, содержащего параллельный агент Q_i;

(2) f — первый сегмент фрагмента F_i;

(3) q — количество сегментов фрагмента F_i, не зеркалированных ни на одном из остальных дисковых модулей, т. е. $q = \lceil S(F_i) \cdot (1 - \max_{1 \le l \le H} r(l)) \rceil$, где $r(l)$ — функция репликации в DM-дереве .

Балансировка загрузки в процессе обработки запроса происходит следующим образом. Пусть агент Q_i завершил обработку выделенных ему данных и перешел в состояние ожидания. После этого вычисляется наиболее загруженный процессорный модуль, и агенту Q_i выделяется часть его работы. Наиболее загруженный процессорный модуль выбирается из тех процессорных модулей, расстояние до которых не превышает j. Критерием загруженности процессорного модуля является количество обработанных кортежей фрагмента на смежном с процессорным модулем диске. Чем меньше количество обработанных кортежей на смежном процессору диске, тем более загруженным считается процессорный модуль. Формально, задача выбора загруженного процессорного модуля сводится к поиску максимума функции: $m(k) = \lambda_j(P_k, P_i) \cdot \bar{S}(F_k)$, по всем $k : 1 \le k \le n$, где $\bar{S}(F_k)$ — количество необработанных сегментов фрагмента F_k. Функция λ_j принимает значение, равное единице, если расстояние от P_i до P_k не превосходит j. В противном случае, $\lambda_j = 0$. Количество выделяемых сегментов определяется по формуле

$$\left\lceil \frac{\min(\bar{S}(F_k^i) \cdot \mathrm{L}(F_k),\ E(F_k^i))}{\delta_j} \right\rceil,$$

где δ_j — степень уровня j дерева T. Номер первого кортежа

$$C(F_k) = E(F_k) - \min(\bar{S}(F_k) \cdot \mathrm{L}(F_k),\ E(F_k^i)) + 1.$$

В процессе обработки запроса может возникнуть ситуация, когда узел, содержащий агент Q_i, не сможет помочь ни одному из

смежных с ним на уровне j узлов. Это происходит в том случае, когда фрагменты первичных дисков всех смежных узлов целиком обработаны. Данная ситуация задается условием

$$\sum_{k=1}^{n} \lambda_j(D_k, D_i) \cdot \bar{S}(F_k) = 0. \qquad (23)$$

Если (23) выполняется, то наиболее загруженный процессорный модуль выбирается из процессорных модулей, находящихся на расстоянии $j+1$ от P_i. Агент Q_i завершает свою работу, если (23) истинно при $j = H - 1$.

3. Организация параллельной обработки запросов

Общая схема обработки запроса в иерархической параллельной системе баз данных представлена на рис. 8 (в данном случае мы считаем, что вычислительная система имеет двухуровневую иерархическую архитектуру и состоит из двух вычислительных узлов, каждый из которых содержит четыре процессорных модуля).

В соответствие с этой схемой запрос на языке SQL передается пользователем на host-машину. Там он транслируется в некоторый *последовательный физический план* [8]. Данный последовательный физический план преобразуется в *параллельный план 1-го уровня*, представляющий собой совокупность *агентов 1-го уровня*. Каждый агент 1-го уровня, кроме реляционных операций, может содержать специальные *параллельные* операции `psplit` *(расщепление)* и `pmerge` *(слияние)*, семантика которых описывается в [19]. Параллельный план 1-го уровня задает распараллеливание запроса с точностью до вычислительного узла.

На следующем этапе параллельный план 1-го уровня преобразуется в *параллельный план 2-го уровня*. При этом каждый агент 1-го уровня отображается в n агентов 2-го уровня. Здесь

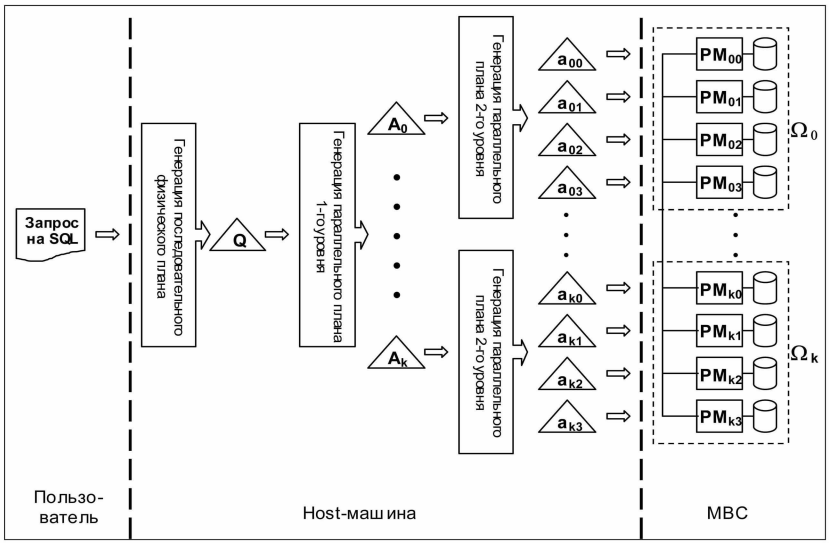

Рис. 8: Схема обработки запроса в иерархической параллельной системе баз данных. Q — последовательный физический план, A_i — агент 1-го уровня, a_{ij} — агент 2-го уровня, PM_{ij} — процессорный модуль, Ω_i — вычислительный узел.

n обозначает количество процессорных модулей в вычислительной системе (на рис. 8 $n = 4$). Это достигается путем вставки оператора обмена `exchange` [19] в соответствующие места дерева запроса. Параллельный план 2-го уровня задает распараллеливание запроса с точностью до процессорного модуля.

На завершающем этапе агенты 2-го уровня пересылаются с host-машины на соответствующие процессорные модули, где *интерпретируются* исполнителем запросов. Результаты выполнения агентов в пределах одного вычислительного узла объединя-

ются корневым оператором `exchange` на нулевом процессорном модуле. Если запрос выполнялся на нескольких вычислительных узлах, то суммарный результат получается путем выполнения операции `pmerge` на нулевом узле одного из вычислительных узлов, откуда передается на host-машину.

Рассылка параллельных агентов и передача данных в иерархической системе осуществляется на основе системы передачи сообщений, состоящей из двух подсистем: кондуктора и маршрутизатора. Кондуктор обеспечивает передачу сообщений между процессорными модулями одного вычислительного узла. Маршрутизатор обеспечивает передачу сообщений между разными вычислительными узлами.

Поясним цикл обработки запроса в параллельной системе баз данных на следующем **примере**. Пусть необходимо вычислить $\mathbf{Q} = \mathbf{R} \bowtie \mathbf{S}$ – соединение двух отношений $\mathbf{R}$ и $\mathbf{S}$ по некоторому общему атрибуту Y. Пусть $\mathbf{R}$ фрагментировано по атрибуту соединения на двух вычислительных узлах $\mathbf{\Omega_0}$ и $\mathbf{\Omega_1}$ в виде двух фрагментов R_0 и R_1, т. е.

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

где π — операция проекции [7]. Пусть $\mathbf{S}$ фрагментировано по атрибуту соединения на тех же двух вычислительных узлах $\mathbf{\Omega_0}$ и $\mathbf{\Omega_1}$ в виде двух фрагментов S_0 и S_1, т. е.

$$\mathbf{S} = S_0 \cup S_1, \qquad S_0 \cap S_1 = \emptyset, \qquad \pi_Y(S_0) \cap \pi_Y(S_1) = \emptyset.$$

Тогда последовательный физический план запроса $\mathbf{Q}$ и соответствующий ему параллельный план 1-го уровня будут иметь вид, изображенный на рис. 9. Параллельный план 1-го уровня в данном случае включает в себя двух агентов A_0 и A_1, которые будут выполняться на вычислительных узлах $\mathbf{\Omega_0}$ и $\mathbf{\Omega_1}$, соответственно. Агент A_0 в отличие от агента A_1 имеет дополнительную операцию `pmerge` в качестве корневого узла. В качестве входных потоков операции `pmerge` фигурируют склад, ассоциированный

с операцией `join`, и канал маршрутизатора, ассоциированный с агентом A_1.

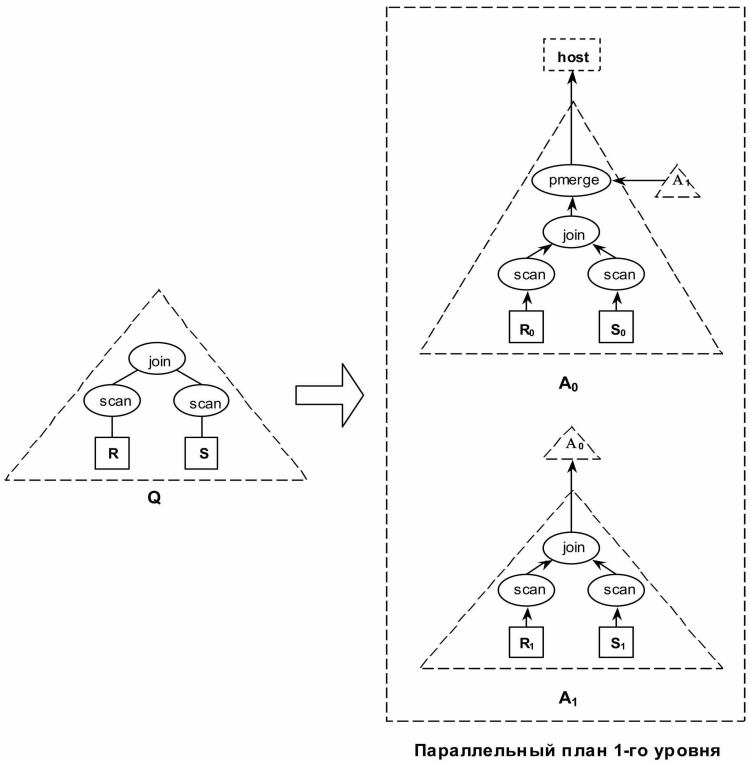

Рис. 9: Последовательный физический план и параллельный план 1-го уровня для запроса $\mathbf{Q} = \mathbf{R} \bowtie \mathbf{S}$.

В качестве выходного потока операции `pmerge` указывается стандартный поток вывода на терминал host-машины. В качестве корневого узла агента A_1 фигурирует операция `join`, вы-

ходным потоком которой является канал маршрутизатора, ассоциированный с агентом A_0.

Рассмотрим теперь на примере агента A_1 преобразование параллельного плана 1-го уровня в параллельный план 2-го уровня. Для простоты мы будем далее предполагать, что в качестве алгоритма операции соединения используется алгоритм вложенных циклов [7], причем во внутреннем цикле всегда сканируются кортежи из $\mathbf{S}$. Сначала рассмотрим ситуацию, когда R_1 и S_1 фрагментированы внутри вычислительного узла по атрибуту соединения Y с использованием одной и той же функции фрагментации:

$$R_1 = \bigcup_{j=0}^{3} R_{1j}, \qquad \bigcap_{j=0}^{3} R_{1j} = \emptyset, \qquad \forall_{i \neq j} \pi_Y(R_{1i}) \bigcap \pi_Y(R_{1j}) = \emptyset\,,$$

$$S_1 = \bigcup_{j=0}^{3} S_{1j}, \qquad \bigcap_{j=0}^{3} S_{1j} = \emptyset, \qquad \forall_{i \neq j} \pi_Y(S_{1i}) \bigcap \pi_Y(S_{1j}) = \emptyset\,,$$

$$\forall_{i \neq j} \pi_Y(R_{1i}) \bigcap \pi_Y(S_{1j}) = \emptyset\,.$$

В этом случае агент 1-го уровня A_1 преобразуется в четырех агентов 2-го уровня A_{10}, A_{11}, A_{12}, A_{13} (по числу процессорных модулей в вычислительном узле), как это показано на рис. 10.

Отметим, что все четыре агента 2-го уровня абсолютно идентичны, за исключением индексов исходных фрагментов отношений. Они получаются из агента A_1 путем вставки в корень дерева оператора обмена $\texttt{exchange}_{\texttt{e}_1}$ (мы будем проставлять индекс у каждого оператора `exchange`, чтобы различать различные операторы в пределах одного дерева). В нашей конфигурации любой оператор `exchange` создает три канала для связи со всеми процессорными модулями вычислительного узла, кроме собственного (мы предположили, что количество процессорных модулей вычислительного узла равно четырем). Канал кондуктора однозначно идентифицируется парой (C, P), где C — ло-

гический номер процессорного модуля в вычислительном узле, P — номер порта. В простейшем случае в качестве номера порта можно взять порядковый номер оператора `exchange` в дереве агента. Для каждого оператора `exchange` должна быть указана функция распределения, определяющая, на каком процессорном модуле вычислительного узла должен быть обработан тот или иной кортеж, помещенный во входной поток оператора `exchange`. В данном случае оператор $\texttt{exchange}_{\texttt{e}_1}$ имеет функцию распределения, задаваемую формулой

$$f(x) = 0.$$

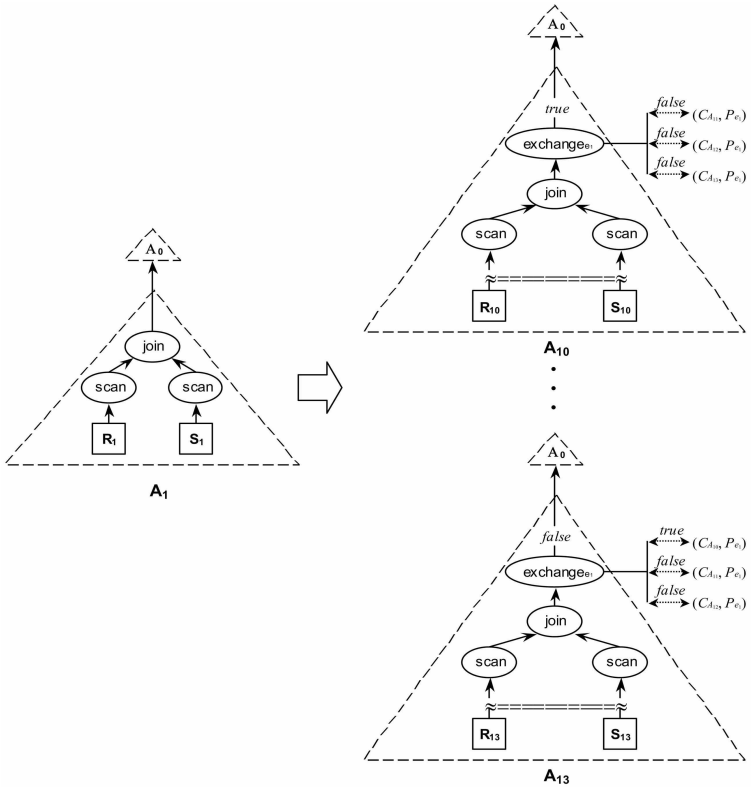

Рис. 10. Преобразование агента 1-го уровня в совокупность агентов 2-го уровня для случая, когда R_1 и S_1 одинаково фрагментированы по атрибуту соединения. $C_{A_{ij}}$ — номер процессорного модуля в вычислительном узле, на котором будет выполняться агент A_{ij}, P_{e_l} — номер порта обмена, ассоциированного с оператором `exchange` с номером e_l, $\approx$ — балансировка загрузки.

Это означает, что все кортежи, полученные в результате выполнения операции `join`, должны быть отправлены на нулевой процессорный модуль кластера $\mathbf{\Omega_1}$ (т. е. для агентов A_{11}, A_{12}, A_{13} выходной поток оператора $\texttt{exchange}_{e_1}$ будет всегда пуст). Из нулевого процессорного модуля кластера $\mathbf{\Omega_1}$ кортежи результирующего отношения будут переданы на нулевой процессорный модуль кластера $\mathbf{\Omega_0}$ в качестве входного потока операции `pmerge`.

Символы $\approx$ отмечают фрагменты отношений, к которым может быть применен алгоритм балансировки загрузки, описываемый в разделе 2.4. В данном случае балансировка может осуществляться только *синхронно* между обоими входными отношениями за счет динамического изменения общей функции фрагментации. В противном случае результат операции соединения может получиться неверным.

Предположим теперь, что R_1 фрагментирован внутри вычислительного узла произвольным образом, а S_1 — по-прежнему по атрибуту соединения Y. В этом случае агент 1-го уровня A_1 преобразуется в четырех агентов 2-го уровня так, как это показано на рис. 11. Все агенты 2-го уровня идентичны, за исключением индексов исходных фрагментов отношений. Оператор $\texttt{exchange}_{e_1}$ не может динамически меняться, поскольку в момент ее изменения в выходном складе оператора могут находиться необработанные кортежи, что может привести в итоге к неверному результату операции соединения. Тем не менее, отсутствие динамической балансировки применительно к фрагментам $S_{10}, ..., S_{13}$ не приведет к ощутимому дисбалансу в загрузке процессорных модулей вычислительного узла, так как мы сохраняем возможность использования алгоритма балансировки применительно к фрагментам $R_{10}, ..., R_{13}$, сканируемым во внешнем цикле операции соединения.

В заключение рассмотрим случай, когда оба отношения R_1 и S_1 фрагментированы внутри Омега-кластера произвольным

образом. В этом случае агент 1-го уровня A_1 преобразуется в четырех агентов 2-го уровня так, как это показано на рис. 12.

Все агенты 2-го уровня по-прежнему идентичны, за исключением индексов исходных фрагментов отношений. Оператор $\texttt{exchange}_{e_1}$ имеет функцию распределения $f(r) = 0$, как это было в предыдущих случаях. В качестве функции распределения для операторов $\texttt{exchange}_{e_2}$ и $\texttt{exchange}_{e_3}$ можно использовать произвольную, но одну и ту же функцию фрагментации по атрибуту Y. При этом мы можем применять алгоритм балансировки загрузки ко всем входным фрагментам независимо друг от друга.

4. Заключение

В данной работе были рассмотрены проблемы эффективной организации систем баз данных на мультипроцессорных вычислительных системах с иерархической архитектурой.

Предложена модель машины баз данных, позволяющая моделировать различные конфигурации иерархических вычислительных систем. На основе предложенной модели нами был спроектирован и реализован программный комплекс, получивший название ЭВМБД (Эмулятор Виртуальных Мультипроцессоров Баз Данных). Данный программный комплекс может быть использован для моделирования и сравнительного анализа различных иерархических многопроцессорных архитектур в контексте задач класса OLTP. Был проведен эксперимент по моделированию архитектуры высокопроизводительного вычислительного кластера Южно-Уральского государственного университета [21]. Полученные на эмуляторе результаты достаточно хорошо согласуются с реальными результатами, полученными при использовании прототипа параллельной СУБД Омега.

Введена модель симметричной многопроцессорной иерархической системы. 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 алгоритм балансировки загрузки, основанный на методе частичного зеркалирования.

Предложена новая модель распараллеливания запросов, ориентированная на иерархические многопроцессорные архитектуры. Данная модель позволяет выполнять на мультипроцессорной системе много различных запросов одновременно. Каждый из этих запросов может быть автоматически распараллелен на любом уровне многопроцессорной иерархии.

В качестве основных направлений дальнейших исследований можно выделить следующие. Во-первых, это — аналитическое получение оценок трудоемкости обновления реплик с учетом помех. Во-вторых, мы планируем на базе DMM модели разработать программу, моделирующую работу иерархической многопроцессорной системы баз данных, и провести с ее помощью эксперименты по исследованию механизмов распределения данных и балансировки загрузки. В-третьих, мы предполагаем реализовать описанные методы и алгоритмы в прототипе параллельной СУБД Омега [18] для кластеров и Grid систем.

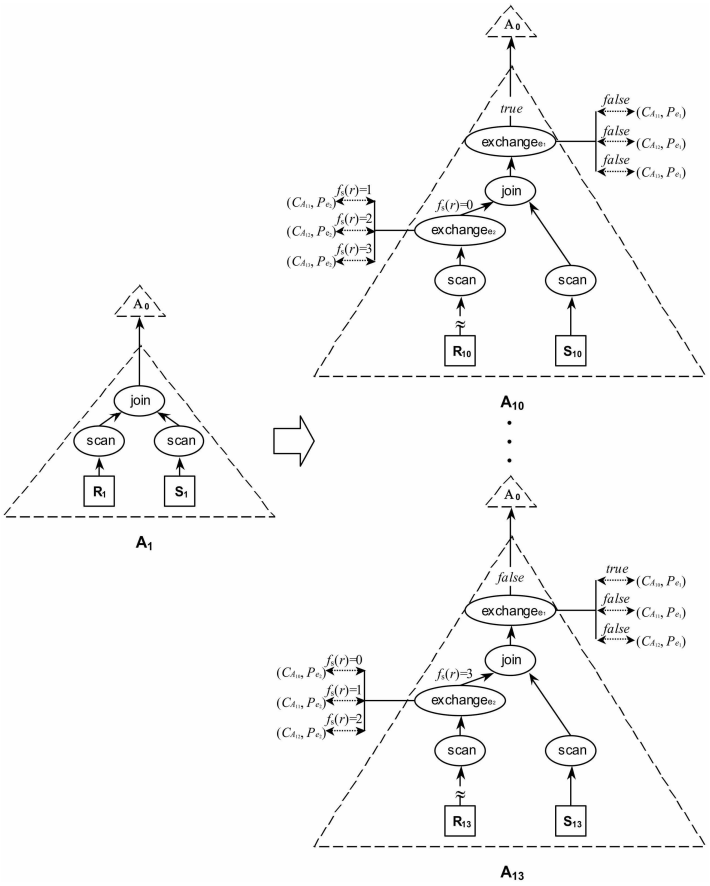

Рис. 11. Преобразование агента 1-го уровня в совокупность агентов 2-го уровня для случая, когда R_1 фрагментировано произвольным образом, а S_1 — по атрибуту соединения. $C_{A_{ij}}$ — номер процессорного модуля в вычислительном узле, на котором будет выполняться агент A_{ij}, P_{e_l} — номер порта обмена, ассоциированного с оператором `exchange` с номером e_l, f — функция распределения для оператора `exchange`, $\approx$ — балансировка загрузки.

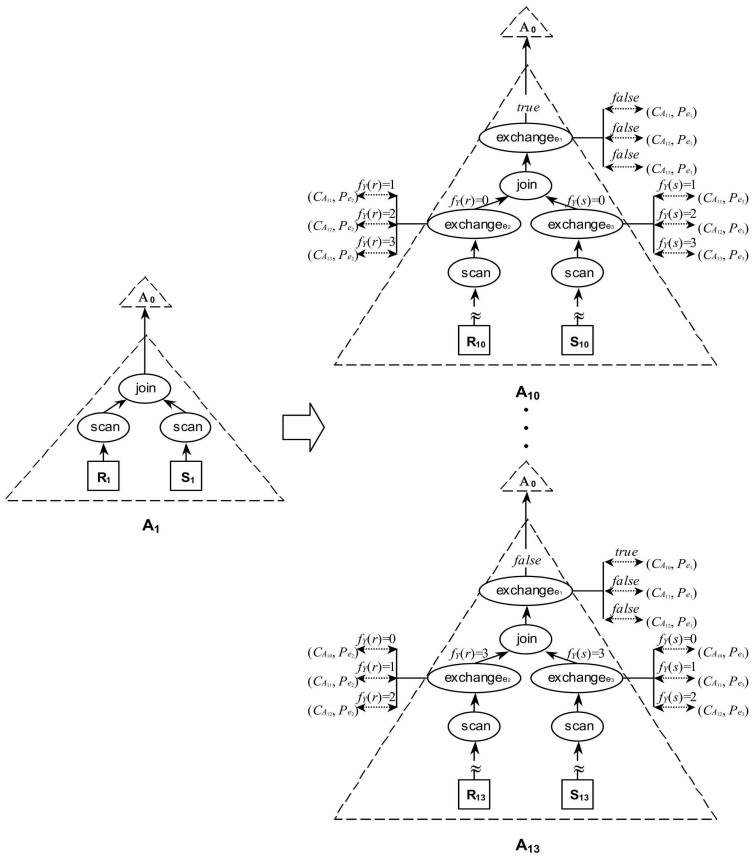

Рис. 12. Преобразование агента 1-го уровня в совокупность агентов 2-го уровня для случая, когда оба отношения R_1 и S_1 фрагментированы произвольным образом. $C_{A_{ij}}$ — номер процессорного модуля на вычислительном узле, на котором будет выполняться агент A_{ij}, P_{e_l} — номер порта обмена, ассоциированного с оператором `exchange` с номером e_l, f — функция распределения для оператора `exchange`, $\approx$ — балансировка загрузки.

СПИСОК ЛИТЕРАТУРЫ

1. *Bhide A., Stonebraker M.* A Performance Comparison of Two Architectures for Fast Transaction Processing // Proceedings of the Fourth International Conference on Data Engineering, February 1-5, 1988, Los Angeles, California, USA. IEEE Computer Society, 1988. P. 536–545.
2. *Bhide A.* An Analysis of Three Transaction Processing Architectures // Fourteenth International Conference on Very Large Data Bases (VLDB'88), August 29 — September 1, 1988, Los Angeles, California, USA, Proceedings. Morgan Kaufmann, 1988. P. 339–350.
3. *Bitton D., Gray J.* Disk Shadowing // Fourteenth International Conference on Very Large Data Bases, August 29 — September 1, 1988, Los Angeles, California, USA, Proceedings. Morgan Kaufmann. 1988. P. 331–338.
4. *Bouganim L., Florescu D., Valduriez P.* Dynamic Load Balancing in Hierarchical Parallel Database Systems // VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India. Morgan Kaufmann. 1996. P. 436–447.
5. *Chen S., Towsley D.F.* Performance of a Mirrored Disk in a Real-Time Transaction System // 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, San Diego, California, USA, May 21-24, 1991, Proceedings. Performance Evaluation Review. May 1991. Vol. 19, No. 1. P. 198–207.
6. *Foster I.T., Grossman R.L.* Blueprint for the future of high-performance networking: Data integration in a bandwidth-rich world // Communications of the ACM. 2003. Vol. 46, No.11. P. 50–57.
7. *Garcia-Molina H., Ullman J.D., Widom J.* Database System

Implementation. Prentice Hall, 2000. 653 p.

8. *Graefe G.* Query evaluation techniques for large databases // ACM Computing Surveys. 1993. Vol. 25, No.2. P. 73–169.
9. *Gray J., Liu D., DeWitt D. J., Heber G.* Scientific Data Management in the Coming Decade // SIGMOD Record. 2005. Vol. 34, No. 4.
10. *Lu H., Tan K. L.* Dynamic and Load-balanced Task-Oriented Datbase Query Processing in Parallel Systems // Advances in Database Technology — EDBT'92, 3rd Int. Conf. on Extending Database Technology, Vienna, Austria, March 23-27, 1992, Proceedings. Lect. Not. in Comp. Sc., Vol. 580. Springer. 1992. P. 357–372.
11. *Mehta M., DeWitt D.J.* Placement in Shared-Nothing Parallel Database Systems // The VLDB Journal. — 1997. Vol. 6, No. 1. P. 53–72.
12. *Omiecinski E.* Performance Analysis of a Load Balancing Hash-Join Algorithm for a Shared Memory Multiprocessor // 17th International Conference on Very Large Data Bases, September 3-6, 1991, Barcelona, Catalonia, Spain, Proceedings. Morgan Kaufmann. 1991. P. 375–385.
13. *Williams M.H., Zhou S.* Data Placement in Parallel Database Systems // Parallel database techniques / IEEE Computer society. 1998. P. 203–218.
14. *Xu Y., Dandamudi S.P.* Performance Evaluation of a Two-Level Hierarchical Parallel Database System // Proceedings of the Int. Conf. Computers and Their Applications, Tempe, Arizona. 1997. P. 242–247.
15. *Девитт Д., Грэй Д.* Параллельные системы баз данных: будущее высоко эффективных систем баз данных //СУБД. — М.: Открытые системы, 1995. №2. С. 8—31.
16. *Кнут Д.Э.* Искусство программирования. Т. 1. Основные

алгоритмы, 3-е изд. — М.: Изд. дом “Вильямс”, 2000. 720 с.

17. *Кнут Д.Э.* Искусство программирования. Т. 3. Сортировка и поиск, 2-е изд. — М.: Изд. дом “Вильямс”, 2000. 832 с.
18. Прототип параллельной СУБД Омега http://omega.susu.ru/prototype/.
19. *Соколинский Л.Б.* Организация параллельного выполнения запросов в многопроцессорной машине баз данных с иерархической архитектурой // Программирование. — М.: Наука, 2001. №6. С. 13–29.
20. *Соколинский Л.Б.* Обзор архитектур параллельных систем баз данных // Программирование. — М.: Наука, 2004. №6. С. 49–63.
21. Технические характеристики вычислительного кластера Южно-Уральского государственного университета http://cluster.susu.ru/information.