Параллельное и распределенное программирование, грид-технологии, программирование на графических процессорах

УДК 004.272.3

Мунерман В.И.

Смоленский государственный университет, г. Смоленск, Россия

РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ДАННЫХ В ОБЛАЧНЫХ СИСТЕМАХ

Аннотация

Рассмотрены методы использования технологии облачных вычислений для разработки программного обеспечения, реализующего параллельную обработку распределенных данных. Рассмотрен метод параллельной реализации операции JOIN на основе принципа симметричного горизонтального распределения. Даны описания виртуальных программно-аппаратных комплексов, которые реализуют распараллеливание на уровне обработки запросов и на уровне операции JOIN средствами Microsoft Azure. Приведены экспериментальные данные, подтверждающие эффективность предложенного подхода.

Ключевые слова

Облачные системы и вычисления; параллельная обработка данных; массовая обработка данных; архитектура программно-аппаратных комплексов; методы доступа.

Munerman V.I.

Smolensk State University, Smolensk, Russia

THE IMPLEMENTATION OF PARALLEL DATA PROCESSING IN CLOUD SYSTEMS

Abstract

The methods of using cloud computing technology for the development of software that implements parallel processing of distributed data are considered. The method of parallel implementation of the JOIN operation based on the symmetric horizontal distribution principle is considered. There are descriptions of virtual software and hardware systems that implement parallelization at the level of processing requests and at the level of the JOIN operation using Microsoft Azure. Experimental data confirming the effectiveness of the proposed approach are presented.

Keywords

Cloud systems and computing; parallel data processing; massively data processing; software-hardware complexes architecture; data access methods.

Введение

Статья посвящена рассмотрению методов использования технологии облачных вычислений для разработки программного обеспечения, реализующего параллельную обработку распределенных данных. Она представляет собой развитие методов, предложенных в работе [1].

Облачные вычисления определяются как вычислительные концепции, которые подразумевают большое количество компьютеров, подключенных через сеть реального времени, подобную Интернет. Как и грид- системы [6] облачные системы служат основой для распределенных вычислений по сети, а значит, дают возможность запуска программы или приложения на многих подключенных компьютерах одновременно. Облачные вычисления основаны на данных, приложениях, инфраструктуре и различных платформах для разработчиков приложений, расположенных в глобальной сети (WAN).

Цель работы заключается в исследовании возможности применения облачных технологий для параллельной обработки больших структурированных данных посредством распределения этих данных между большим

количеством вычислителей.

Анализ ресурсов облачных систем

Облачные системы предоставляют программисту большой набор инструментальных средств разработки приложений. Они объединены в подсистему, которая называется Платформа как сервис (Platform as a Service – PaaS). PaaS – это предоставление облачных вычислений, при котором потребитель получает доступ к использованию таких технологических платформ, как: операционные системы, системы управления базами данных, связующее программное обеспечение, средства разработки и тестирования приложений, размещённых в облачной системе [2]. Далее рассматриваются методы и ресурсы, необходимые для параллельной обработки распределенных данных, которые обеспечивает разработчикам PaaS облачной системы Windows Azure.

Первый основной ресурс – это виртуальная машина. Виртуальная машина конструируется пользователем в соответствии с его требованиями. Минимальные возможности – это одноядерный процессор с 768 Мбайтами памяти, а наиболее мощный вариант – восьмиядерный процессор с 56 Гбайтами памяти. На виртуальной машине устанавливается операционная система Windows Server. Если виртуальная машина используется для работы с базами данных, то на ней также устанавливается СУБД Microsoft SQL Server Azure. Версия и настройка программного обеспечения определяется мощностью машины.

Второй основной ресурс – это отдельная база данных. Программисту предоставляется возможность создания необходимого количества баз данных на облачных ресурсах. Эти БД могут быть физически независимыми друг от друга, то есть располагаться на различных облачных хранилищах данных и обрабатываться различными облачными процессорами. Для создания БД и выполнения запросов к ней используется СУБД Microsoft SQL Server Azure.

Третий ресурс, который может быть использован для решения вспомогательных задач параллельной обработки распределенных данных, – хранилище таблиц. К таблицам можно применять только простые запросы, не содержащие бинарных операций типа операции соединения. Таблицы могут использоваться как индексные файлы при индексно-последовательной организации данных.

Далее рассматриваются **методы взаимодействия** прикладных программ с этими основными ресурсами. При этом предполагается, что:

- организация данных реализована на основе принципа симметричного горизонтального распределения;
- программа реализует параллельное выполнение независимых потоков, которые вместе реализуют массовую обработку данных.

Общая схема взаимодействия прикладной программы с основными облачными ресурсами показана на рисунке 1.

Прикладная программа, которая реализует массовую обработку данных, может располагаться на рабочей станции пользователя или на виртуальной машине в облаке. В последнем случае пользователь получает к ней доступ с помощью удаленного рабочего стола.

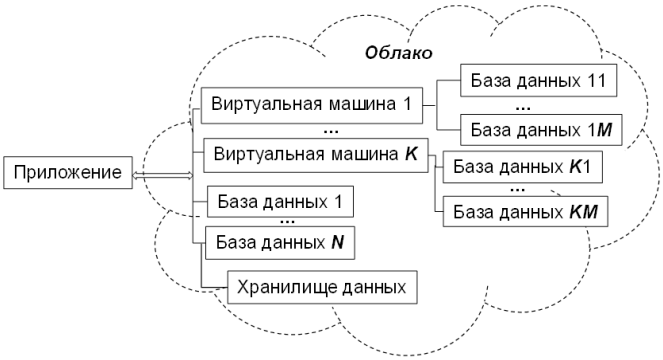

Рис. 1. Взаимодействие между прикладной программой и облачными ресурсами

Обработка данных, расположенных в облаке, может быть реализована различными способами.

Первый способ реализует параллельную обработку на уровне запросов. В этом случае алгебраическое выражение запроса разбивается на подвыражения, которые могут выполняться одновременно на разных процессорах или ядрах одного многоядерного процессора.

Пример 1. Пусть даны две таблицы с одинаковыми схемами: $P(A, B)$ и $Q(A, B)$, и таблица $R(A, C)$. Эти таблицы участвуют в запросе, в котором сначала выполняется операция объединения таблиц P и Q, а затем операция JOIN над результатом объединения и таблицей R. Этот запрос имеет вид:

SELECT *R.A*, *PUQ.B*, *R.C* FROM *R* INNER JOIN *PUQ* ON *R.A* =*PUQ.A*;

PUQ – запрос на объединение таблиц *P* и *Q*:

SELECT *P.A*, *P.B* FROM *P* UNION SELECT *Q.A*, *Q.B* FROM *Q*;

Этот запрос может быть преобразован в эквивалентный запрос:

SELECT *R.A*, *P.B*, *R.C* FROM *R* INNER JOIN *P* ON *R.A* = *P.A*

UNION

SELECT *R.A*, *Q.B*, *R.C* FROM *R* INNER JOIN *Q* ON *R.A* = *Q.A*;

(операции JOIN и UNION подчиняются закону дистрибутивности).

Схема взаимодействия прикладной программы с распределенными между двумя БД в облаке данными и запросами приведена на рисунке 2.

Прикладная программа может располагаться как на виртуальной машине в облаке, так и на

компьютере пользователя. В обоих случаях связь прикладной программы с базами данных осуществляется по глобальной сети. Эта связь может быть реализована в модели ODBC. В первой БД выполняется операция JOIN над таблицами P и R, которые могут располагаться как в этой БД, так и в других. Во второй БД выполняется операция JOIN над таблицами Q и R, которые также могут располагаться как в этой БД, так и в других.

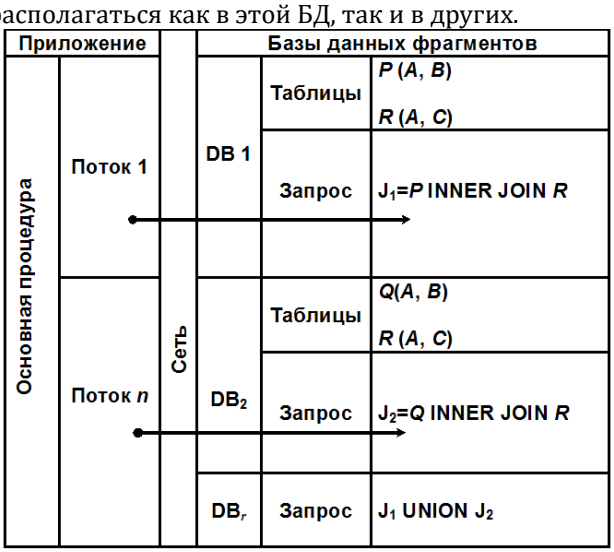

Рис. 2. Взаимодействие прикладной программы с БД в облаке

Прикладная программа запускает два параллельных потока, которые инициируют соответствующие им JOIN-запросы и отслеживают их состояния. Когда выполнение обоих JOIN-запросов заканчивается, приложение запускает выполнение UNION-запроса в базе данных, в которой должен быть размещен результат. Базы данных обрабатываются независимо друг от друга, поэтому можно считать, что для решения задачи создан виртуальный программно-аппаратный комплекс. Его архитектура имеет много общего с такими распространенными архитектурами как симметричное мультипроцессирование (SMP) и массовый параллелизм (MPP). В первом случае прикладная программа и сервер баз данных располагаются на одной виртуальной машине, во втором, они распределены по серверам облака. Следует отметить, что такой подход позволяет использовать реализованный в СУБД параллелизм операций чтения/записи и операций совместной обработки данных

Второй способ основан на параллельной реализации операций, которые составляют запрос. Далее рассматривается использование облачной технологии для параллельной реализации операции JOIN. Эта операция выбрана потому, что она имеет более высокую вычислительную сложность, чем остальные операции, определенные в моделях данных. Для параллельной реализации выбрана следующая разновидность операции JOIN:

P INNER JOIN Q ON $\pi(P.K, Q.K)$;

Таблицы P и Q имеют разные схемы, в каждую из которых входит простой или составной ключ K. Предикат π определен на множествах экземпляров ключа K в таблицах P и Q. Далее, для простоты, но без нарушения общности, предполагается, что в операции JOIN используется наиболее естественный предикат π: $P.K = Q.K$.

Построение параллельной реализации операции JOIN

Для решения этой задачи необходимо использовать промежуточную модель данных. Промежуточная модель должна обеспечить решение проблемы эффективного отображения алгоритма, который реализует эту операцию, на архитектуру программно-аппаратного комплекса. Теоретико-множественная модель (файловая модель) [3] лучше, чем остальные модели, подходит для этой цели.

Основной агрегат данных в теоретико-множественной модели – файл, который определяется как фактор-множество множества однотипных записей X по отношению эквивалентности, порожденному множеством K, и обозначается – X_K.

Соответствие между файлами и таблицами устанавливается легко, потому что запись файла есть способ представления схемы таблицы. Запись должна содержать набор таких полей (множество ключей K), которые можно использовать для построения составного ключа. Основное требование состоит в том, что по множеству ключей K файлу должна соответствовать таблица во второй или (лучше) в третьей нормальной форме. В этом случае файл называется строго упорядоченным.

В общем случае, при выполнении операции JOIN предикат π определен на множестве $M \subset K$. Тогда файлы нестрого упорядочены, а таблицы, им соответствующие, будут в первой нормальной форме. В этом случае таблицы как и файлы состоят из классов эквивалентности, в которых все строки содержат один и тот же экземпляр составного ключа K. Тогда для параллельной реализации операции JOIN можно использовать метаданные, определяющие распределение строк таблицы по классам эквивалентности. Такой подход называется индексно-последовательным и основан на одноименном методе доступа – ISAM.

Для распределения пар классов эквивалентности между базами данных можно использовать известный алгоритм упаковки рюкзака и эвристический алгоритм бустрофедона [4], названный так по аналогии с одним из древних методов письма. Второй алгоритм для N баз данных состоит из следующих шагов.

1. Из таблиц P и Q формируется сводная индексная таблица *ComInd*, упорядоченная по убыванию значений поля *MM*.

2. Первые N пар классов эквивалентности таблиц P и Q, которые отмечены в первых N строках таблицы *ComInd*, последовательно добавляются в таблицы-фрагменты P_i и Q_i баз данных DB_i (i=1, …, N).
3. Последние N пар классов эквивалентности таблиц P и Q, которые отмечены в последних N строках таблицы *ComInd*, в обратном порядке (i=N, …, 1) последовательно добавляются в таблицы-фрагменты P_i и Q_i баз данных DB_i.
4. Прямой (обратный) счетчик строк таблицы *ComInd* увеличивается (уменьшается) на N.
5. Пункты 2-4 повторяются до тех пор, пока не будет исчерпана таблица *ComInd*.

В [4] проведен совместный анализ алгоритма упаковки рюкзака и алгоритма бустрофедона. Анализ показал, что с увеличением N:

1. Результаты работы алгоритмов сближаются. Это позволяет отдать предпочтение более быстрому (полиномиальному) алгоритму бустрофедона.
2. Величина $R_{max} - R_{min}$ возрастает. При этом уменьшается количество пар классов эквивалентности, которые размещаются в таблицах-фрагментах P_i и Q_i баз данных DB_i (i=1, …, N). В предельном случае, когда число пар классов равно количеству вычислителей, время выполнения операции JOIN близко к времени декартова произведения пары классов эквивалентности, соответствующих величине R_{max}.

Далее показано, как симметричное горизонтальное распределение таблиц может быть реализовано исключительно средствами СУБД.

Пусть K_1, …, K_n – множество значений ключа K. Таблицы $P(K, …)$ и $Q(K, …)$ имеют схемы, которые содержат ключ K и произвольные наборы полей (как правило, различные). Таблице P можно поставить в соответствие индексную таблицу со схемой $indP(K, I, M)$. Эта таблица – есть множество строк $indP=\{<K_1^*, i_1, m_1>, \ldots, <K_m^*, i_m, m_m>\}$. i_j – индекс первой записи класса эквивалентности, строки которого содержат экземпляр K_j ключа K, m_j – количество записей в этом классе эквивалентности. Поле I имеет тип Счетчик строк, а поле M – целочисленное. Таблица *indP* может быть получена в результате двух запросов:

1. $Q1$=INSERT INTO *indPt* (*K*, *M*) SELECT *P.K*, 1 AS *M* FROM *P* ORDER BY *P.K*;
2. $Q2$=SELECT *indPt.K*, First(*indPt.M*) AS *I*, Sum(*IndPt.M*) AS *M* INTO *indP* FROM *indPt* GROUP BY *indPt.K* ORDER BY *indPt.K*;

Запрос $Q1$ формирует промежуточную таблицу, которая содержит столько же строк, что и таблица *P*. Каждая строка этой таблицы содержит свой номер, значение ключа *K* из соответствующей строки таблицы *P* и поле *M*, содержащее значение 1. Запрос $Q2$ завершает построение таблицы *indP* посредством операции группировки. Этому двухпроходному алгоритму соответствует простой однопроходный алгоритм, который легко разработать на любом языке программирования, связанным с СУБД, или на языках манипулирования данными, такими как Tranzact-SQL или PL-SQL. Листинг 1 демонстрирует реализацию однопроходного алгоритма построения индексного файла для индексно-последовательного доступа к таблице *P*.

Этому двухпроходному алгоритму соответствует простой однопроходный алгоритм, который легко разработать на любом языке программирования, связанном с СУБД, или на языках манипулирования данными, такими как Tranzact-SQL или PL-SQL. Листинг 1 демонстрирует реализацию однопроходного алгоритма построения индексного файла для индексно-последовательного доступа к таблице *P*.

Листинг 1. Построение индексного файла средствами языка Tranzact-SQL

```
1.  DECLARE @K <тип>, @currentK <тип>, @I int, @M int, @RowCount int
2.  DECLARE @j int
3.  DELETE FROM indP
4.  DECLARE cursorP CURSOR FOR SELECT K FROM P ORDER BY K
5.  OPEN cursorP
6.  SET @RowCount =@@Cursor_Rows
7.  SET @j=0
8.  FETCH NEXT FROM cursorP INTO @K
9.  SET @currentK =@K
10. SET @I=1
11. SET @M=1
12. WHILE @j<@ RowCount -1
13. BEGIN
14.     FETCH NEXT FROM cursorP INTO @K
15.     IF @currentK =@K
16.         SET @M=@M+1
17.     ELSE
18.     BEGIN
19.             INSERT INTO indP (K, I, M) VALUES (@currentK, @I, @M)
20.         SET @currentK =@K
21.         SET @I=@I+@M
22.         SET @M=1
23.     END
24.     SET @j=@j+1
25. END
26. INSERT INTO indP (K, I, M) VALUES (@currentK, @I, @M)
```

Подготовительная часть алгоритма (строки 8-11) формирует заготовку первой строки таблицы *indP*. Эта строка содержит первое по порядку значение ключа *K*. Основная часть алгоритма заключена в теле цикла (строки 13-25). Здесь при

совпадении значения ключа *K* в очередной строке таблицы *P* и в заготовке текущей строки таблицы *indP* накапливается значение поля *M*, в противном случае производится вывод текущей строки в таблицу *indP* и формируется заготовка новой строки этой таблицы. По завершении цикла последняя сформированная строка выводится в таблицу *indP* (строка 26).

Таблица *indQ* для таблицы *Q* может быть получена аналогично.

Очевидно, что таблица-результат операции JOIN будет содержать только те классы эквивалентности, которые принадлежат пересечению индексных таблиц *indP*∩*indQ*. Запрос, в результате которого получается это пересечение, имеет вид:

SELECT *indP.M***indQ.M* AS *MM*, *indP.K*, *indP.I*, *indQ.I*, *indP.M*, *indQ.M*
INTO *ComInd* FROM *indP*, *indQ*
WHERE *indP.K* = *indB.K*
ORDER BY *indP.M***indQ.M*

Результат запроса – таблица *ComInd*. Она содержит составной ключ *K*, поля *I* и *M* обеих индексных таблиц, а также поле *MM*, по которому она упорядочена. Поле *MM* вычисляется как произведение количества строк в обоих классах эквивалентности, строки которых содержат одно и то же значение составного ключа *K*. Его значение определяет число выходных строк, которое получится в результате обработки этих классов эквивалентности операцией JOIN.

Использование таблицы *ComInd* позволяет распределить таблицы *P* и *Q* между несколькими независимыми базами данных. Если имеется *N* таких баз данных, то эти таблицы разделяются на строки в таблицу *indP* фрагментов. Каждая база данных $DBSnp_i$ (i=1, …, N) содержит пару фрагментов <P_i, Q_i > таблиц *P* и *Q*. При этом классы эквивалентности, записи которых содержат одинаковые значения составного ключа *K*, полностью расположены в одном и только одном фрагменте. Такой способ распределения таблиц соответствует принципу симметричного горизонтального распределения данных. Таблица *ComInd* соответствует файлу *indD*. Алгоритм бустрофедона, реализующий симметричное горизонтальное распределение таблиц *P* и *Q* также может быть реализован средствами языка Tranzact-SQL так, как это показано в листинге 2.

Листинг 2. Реализация симметричного горизонтального распределения средствами языка Tranzact-SQL

```
1.  DECLARE @KA <тип>, @KD <тип>, @RowCount int, @i int ORDER BY MM ASC
2.  DECLARE cursorDSC CURSOR FOR SELECT K, MM FROM ComInd ORDER BY MM DESC
3.  OPEN cursorASC
4.  OPEN cursorDSC
5.  SET @ RowCount =@@Cursor_Rows
6.  IF @ RowCount %2=0
7.  SET @RowCount =@RowCount /2
8.  ELSE
9.  SET @RowCount =@RowCount /2+1
10. SET @i=0
11. WHILE @i<@RowCount
12. BEGIN
13.     FETCH NEXT FROM cursorASC INTO @ KA
14.     FETCH NEXT FROM cursorDSC INTO @ KD
15.     IF @i % N=0
16.     BEGIN
17.         INSERT INTO PSnp1 SELECT * FROM P WHERE K=@KA
18.         INSERT INTO PSnp1 SELECT * FROM P WHERE K=@KD
19.         INSERT INTO RSnp1 SELECT * FROM R WHERE K=@KA
20.         INSERT INTO RSnp1 SELECT * FROM R WHERE K=@KD
21.     END
22.      …
23.     IF @i % N=N-1
24.     BEGIN
25.         INSERT INTO PSnpN SELECT * FROM P WHERE K=@KA
26.         INSERT INTO PSnpN SELECT * FROM P WHERE K=@KD
27.         INSERT INTO RSnpN SELECT * FROM R WHERE K=@KA
28.         INSERT INTO RSnpN SELECT * FROM R WHERE K=@KD
29.     END
30.     SET @i=@i+1
31. END
```

Средства языка Tranzact-SQL позволяют существенно использовать в процедуре, реализующей алгоритм бустрофедона, упорядоченность таблиц, что отличает его от языка SQL, отражающего свойства классической реляционной модели. Поэтому строки таблицы *ComInd* могут быть прочитаны двумя запросами по возрастанию и по убыванию значений поля *MM* (строки 2, 3). Это позволяет соединить наибольшее и наименьшее значения этого поля. В зависимости от того, четно или нечетно число строк в таблице *ComInd*, определяется количество обрабатываемых строк (строки 7-10), поскольку считывание таблицы двумя запросами с противоположным упорядочиванием позволяет читать только половину таблицы (плюс одна строка, если число строк нечетно). В теле основного цикла (строки 14-31) формируются фрагменты таблиц *P* и *Q*. Как и в случае файловой модели, после выполнения этого варианта алгоритма бустрофедона, количества строк таблицы-результата, получаемые в результате операции JOIN над фрагментами P_i и R_i таблиц *P* и *R*, будут минимально отличаться друг от друга.

Важное значение при реализации симметричного горизонтального распределения

таблиц имеет тот факт, что сами таблицы могут располагаться в БД на различных серверах, а их перемещение между серверами осуществляется автоматически с использованием оператора SELECT … INTO … с указанием сервера, на котором находится БД, содержащая таблицу-операнд.

После завершения симметричного горизонтального распределения таблиц P и Q по базам данных $DBSnp_i$ над содержащимися в этих базах данных парами фрагментов таблиц P_i и Q_i выполняются операции JOIN, по завершении которых полученные фрагменты таблицы результата сливаются в одну таблицу.

Пусть база данных DB_i содержит фрагменты P_i и Q_i, каждый из которых содержит p_i пар классов эквивалентности таблиц P и Q. Операция P_i INNER JOIN Q_i создаст таблицу, которая будет содержать $R_i = \sum_{j=1}^{p_i} MM_j$ строк. Очевидно, что чем меньше значение разности $R_{max} - R_{min}$, тем более эффективно распараллеливание операции JOIN.

Симметричное горизонтальное распределение таблиц P и Q между N базами данных фактически приводит к построению виртуального программно-аппаратного комплекса с MPP архитектурой, который располагается в облачной системе. Структура такого комплекса приведена на рисунке 3.

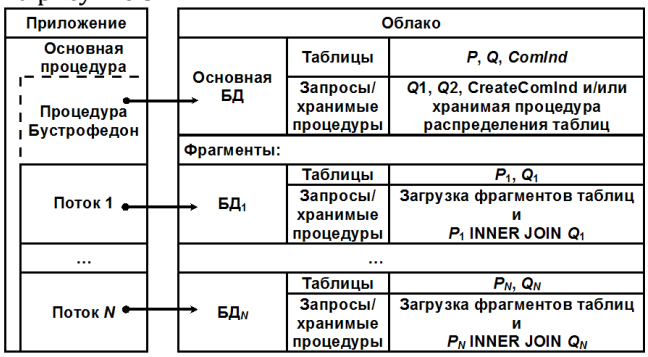

Рис.3. Структура виртуального программно-аппаратного комплекса для массовой обработки данных

Анализ параллельной реализации операции Join

Для оценки качества распараллеливания операции JOIN был проведен вычислительный эксперимент. Виртуальный комплекс был построен на основе Windows Azure – платформы Microsoft, разработанной для реализации облачных вычислений. В состав комплекса входили пять независимых баз данных, расположенных на различных и географически удаленных друг от друга серверах. Из них четыре использовались для симметричного горизонтального распределения таблиц. Результаты эксперимента приведены на рисунке 4.

Эксперимент проводился для таблиц P и Q, число строк в которых изменялось от 400000 до 2500000. Распределение таблиц производилось алгоритмом бустрофедона. На рисунке 4 приведены экспериментальные данные, подтверждающие тот факт, что применение предложенного автором метода параллельного выполнения операции JOIN дает существенное повышение производительности при реализации задач массовой обработки данных.

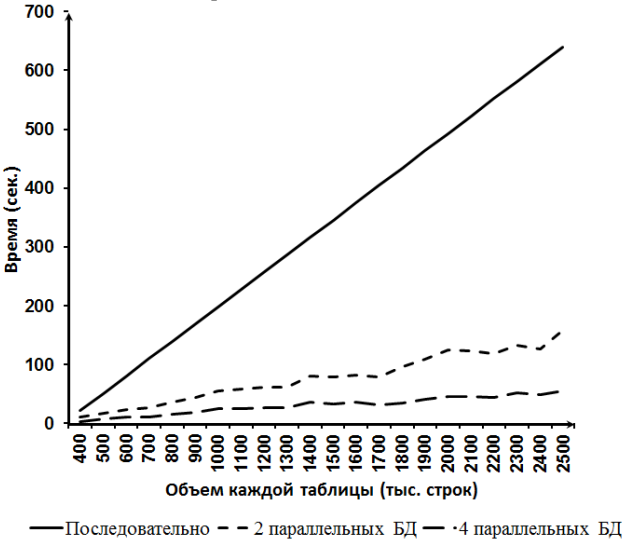

Рис.4. Сравнительный анализ времени последовательного и параллельного выполнения операции JOIN

Действительно, при увеличении объемов таблиц время параллельной реализации операции JOIN над всеми фрагментами симметрично горизонтально распределенных таблиц-операндов одновременно становится значительно меньше, чем время выполнения этой операции над нераспределенными таблицами. Из графика видно, что при распределении таблиц на два фрагмента получается почти пятикратное ускорение, а распределение на четыре фрагмента приводит к почти четырнадцатикратному ускорению.

Заключение

Таким образом, из сказанного можно сделать следующие выводы:

1) предложенные в статье методы параллельной реализации обработки данных на уровне операций позволяют получить существенное ускорение;
2) реализация этих методов не требует использования сложных и достаточно дорогостоящих аппаратных средств, таких как машины баз данных, подобные IBM Netezza и Oracle Exadata;
3) эти методы не зависят от сложности и свойств СУБД, в минимальном варианте достаточно, чтобы СУБД поддерживала язык манипулирования данными SQL и была связана с системой программирования, обеспечивающей разработку параллельных программ;
4) разработка программного обеспечения параллельной обработки данных на основе

симметричного горизонтального распределения достаточно проста и может быть реализована с использованием простых и даже непроцедурных языков манипулирования данными.

Благодарности

Результаты, приведенные в статье, были получены при поддержке корпорации Microsoft в рамках гранта на использование ресурсов облачной системы Windows Azure, благодаря которому эти результаты получили экспериментальное подтверждение.

Автор выражает благодарность доктору технических наук профессору И.Н. Синицыну за неоценимую поддержку и помощь, которая была им оказана в ходе работы над приведенными в статье результатами.

Литература

1. Munerman V.I. The experience of massive data processing in the cloud using Windows Azure (as an example). – Moscow: Highly available systems, 2, Vol. 10, 2014. – P. 3-8.
2. Redkar T., Guidici T. Windows Azure Platform, 2nd Edition. – APRESS, 2011. – P. 602.
3. Мунерман В.И. Объектно-ориентированная модель массовой обработки данных // Системы высокой доступности. – 2011. Том 7, № 4. – С. 72-74.
4. Мунерман В.И. Модели обработки больших объемов данных в системах массового параллелизма // Системы высокой доступности. – 2013. Том 9, № 1. – С. 35-43.
5. Посыпкин М.А., Сухомлин В.А., Храпов Н.П. Комбинированные распределенные инфраструктуры в науке и образовании // Современные информационные технологии и ИТ-образование. – 2015. Том 1, № 11. – 31-36.
6. Посыпкин М.А. Грид-систем из персональных компьютеров в России: текущее состояние и перспективы // Современные информационные технологии и ИТ-образование. – 2012. – № 8. – С. 951-957.

References

1. Munerman V.I. The experience of massive data processing in the cloud using Windows Azure (as an example). – Moscow: Highly available systems, 2, Vol. 10, 2014. – P. 3-8.
2. Redkar T., Guidici T. Windows Azure Platform, 2nd Edition. – APRESS, 2011. – P. 602.
3. Munerman V.I. Ob#ektno-orientirovannaja model' massovoj obrabotki dannyh // Sistemy vysokoj dostupnosti. – 2011. Tom 7, № 4. – S. 72-74.
4. Munerman V.I. Modeli obrabotki bol'shih ob#emov dannyh v sistemah massovogo parallelizma // Sistemy vysokoj dostupnosti. – 2013. Tom 9, № 1. – S. 35-43.
5. Posypkin M.A., Suhomlin V.A., Hrapov N.P. Kombinirovannye raspredelennye infrastruktury v nauke i obrazovanii // Sovremennye informacionnye tehnologii i IT-obrazovanie. – 2015. Tom 1, № 11. – 31-36.
6. Posypkin M.A. Grid-sistem iz personal'nyh komp'juterov v Rossii: tekushhee sostojanie i perspektivy // Sovremennye informacionnye tehnologii i IT-obrazovanie. – 2012. – № 8. – S. 951-957.

Поступила: 25.05.2017

Об авторе:

Мунерман Виктор Иосифович, кандидат технических наук, доцент кафедры информатики, Смоленский государственный университет, vimoon@gmail.com.

Note on the author:

Munerman Victor I., Candidate of Technical Sciences, associate professor of Phisics & Math Department, Smolensk State University, vimoon@gmail.com.