вило, определены на одних и тех же состояниях. Полное описание процессов маршрутизации в сети с n узлами предполагает наличие n переходных матриц вида *(2)*. При этом система уравнений *(3)*, описывающая массовые процессы маршрутизации в сети, является нелинейной.

$$p_{jk} = (l_{jk} - C_{jk}) \cdot m / l_{jk}$$

для $r_{jk} \geq 1$,

$$p_{jk} = (1 - r_{jk}) r_{jk}^{m_{jk}} / (1 - r_{jk}^{m_{jk}+1})$$ для

$r_{jk} < 1$, (3)

$$P^{(l)} = \| p_{ik}^{(l)} \|_{n-1,n-1}$$

$$p_{ik}^{(l)} = \sum_{s=1}^{2^r} \Omega_i^{(s)} x_{ik} .$$

где *1/m* - средняя длина пакетов,

l_{ij} - интенсивность потока в ребре *jk*,

C_{jk} - пропускная способность ребра *jk*,

$W_i^{(s)}$ - вероятность возникновения ситуации (X_{iW}),

p_{ik} - вероятность блокировки канала *ik*,

ρ_{jk} - коэффициент использования канала,

$P_{ik}^{(l)}$ - вероятность отправки пакета из узла i в узел k для искомого узла l.

Численное решение системы нелинейных уравнений *(3)* для заданной сети, трафика и условий функционирования позволяет осуществить определение вероятностно-временных характеристик сети, провести оценку используемых алгоритмов маршрутизации, способов управления потоками и т.п. Как видно из *(4)*, поиск решения системы численным методом носит итерационный характер.

$$p^{(b)}{}_{jk} = (l^{(b-1)}{}_{jk} - C_{jk}) \cdot m / l^{(b-1)}{}_{jk}$$ д

ля $r^{(b-1)}{}_{jk} \geq 1$,

$$p^{(b)}{}_{jk} = (1 - r_{jk}) r_{jk}^{m_{jk}} / (1 - r_{jk}^{m_{jk}+1})$$ для

$r^{(b-1)}{}_{jk} < 1$,

$$P_l^{(b)} = \| p_{ik}^{(b)} \|_{n-1,n-1},$$

$$p_{ik}^{(b)} = \sum_{s=1}^{2^r} \Omega_i^{(s)(b)} x^{(b)}{}_{ik} , \quad (4)$$

$$l^{(b)}{}_{jk} = (\sum_{l}^{n} \sum_{i \neq l}^{n} l_{il}^{(b)} \cdot f_{ij}^{(b)} \cdot q_{jk}^{(b)}) /$$

$$/(1 - s^{(b)}_{jk}) + l_s^{(b)} .$$

где b – номер шага,

$f^{(b)}_{ij}$ - соответствующая строка фундаментальной матрицы F на шаге b,

$q^{(b)}_{jk}$ - соответствующая строка фундаментальной матрицы Q на шаге b,

$s^{(b)}_{jk}$ - среднеквадратичное отклонение интенсивности потока на шаге b,

$\lambda^{(b)}_s$ - служебный поток на шаге b.

Идентификация параметров модели процесса маршрутизации, близких к оптимальным значениям, возможна в ходе итерационного процесса поиска решения системы нелинейных уравнений *(4)*. После введения в итерационный процесс поиска решения системы потоковых уравнений пошаговой процедуры коррекции конфигурационных параметров алгоритма маршрутизации становится возможным нахождение их оптимальных значений для заданной сети и трафика.

МОДИФИКАЦИЯ ДЕРЕВЬЕВ РАЗБОРА ДЛЯ ПАРАЛЛЕЛЬНОГО ИСПОЛНЕНИЯ ЗАПРОСА В СУБД

Локшин М.В.

Основным средством для работы с таблицами, содержащими миллионы строк, является использование какой-либо формы разделения данных и применение алгоритмов для параллельной обработки данных с целью обеспечения приемлемой скорости ответа на пользовательский запрос.

Рассмотрим систему, обеспечивающую работу распределенной СУБД и состоящей из N серверов. Предположим, что пользователь может отправить запрос на языке SQL к любому из N серверов и получить один и тот же ответ от всех серверов (на момент начала исполнения запроса). Такую работу системы можно организовать, к примеру, с использованием одного из методов репликации данных (всей базы, или только части таблиц). В этих условиях возможно создание системы обеспечивающей параллельную обработку SQL запросов, принцип работы которой описан в [1].

Из [2] известно, что схема начальной стадии компиляции запроса состоит из четырех этапов: запрос (текстовое представление) – синтаксический анализатор – препроцессор – генератор логического плана запроса – переписчик логического плана запроса. Дополним эту схему двумя этапами – синтаксический анализатор параллельного запроса и препроцессор параллельного запроса, которые будут предшествовать четырем классическим этапам компиляции. Препроцессор параллельного запроса, в отличие от классической схемы (где он предназначен для замены обозначений деревьями разбора и семантического контроля), в предлагаемой новой схеме модифицирует дерево запроса с целью выделения поддеревьев запроса пригодных для параллельного исполнения. В результате его работы формируется набор новых запросов, обработка которых, в дальнейшем, строится по классической схеме. Преобразования деревьев разбора запроса проводятся препроцессором с использованием заранее известного набора правил, с целью получения эквивалентного запроса. В некоторых случаях после проведения преобразований могут потребоваться дополнительные операции над наборами отношений, возвращаемых запросами.

Под эквивалентностью двух запросов здесь и далее мы будем понимать такие запросы, в результате исполнения которых формируются одинаковые во

всех атрибутах кортежей отношения с точностью до порядка следования кортежей, если не задана инструкция сортировки, и с учетом порядка следования в противном случае.

Очевидно, что некоррелированные запросы допускают параллельное исполнение, поэтому все получившиеся подзапросы в дереве разбора запроса могут быть вычислены независимо. Следует заметить, что в общем случае дальнейшее вычисление запроса согласно дереву разбора можно проводить только при получении результатов всех нижестоящих подзапросов и выражений.

Исходя из вышеизложенного замечания, можно сформулировать цели, которые должны достигаться посредством эквивалентных преобразований запросов:

1. Правило преобразования должно из исходного формировать новый запрос, содержащий заранее заданное число некоррелированных подзапросов.

2. Полученные запросы должны обладать приблизительно равной стоимостью исполнения, так как дальнейшее вычисление запроса возможно только после вычисления соответствующих подзапросов, и в случае существенного превышения времени исполнения одного подзапроса над остальными, друге узлы системы (не занятые вычислением подзапроса) могут простаивать. Таким образом, преобразования запроса должно контролировать баланс нагрузки между узлами системы путем соответствующего формирования подзапросов.

3. На верхних уровнях дерева разбора запроса преобразование должно оставлять наиболее «дешевые» операции. Под термином «дешевые» здесь подразумеваются операции, для реализации которых не требуется обработки большого количества записей, так как, к примеру, при их вычислении уже будет невозможно воспользоваться информацией содержащейся в индексах.

4. Преобразование, по возможности, не должно увеличивать объем отношений, получающихся при вычислении подзапросов, для того, чтобы исключить передачу больших объемов данных между узлами системы. Большие объемы таких передач могут серьезно замедлить исполнение запроса и уменьшить выигрыш от параллельного исполнения запроса.

СПИСОК ЛИТЕРАТУРЫ

1. М. В. Локшин, О.Я. Кравец. Построение систем для параллельной обработки запросов к СУБД. // Телематика’2004: Труды XI Всероссийской научно-методической конференции (7-10 июня 2004). – СПб:ИТМО. 2004. С. 94-95.

2. Гарсиа-Молина Г., Ульман Д., Уидом Д. Системы баз данных. Полный курс. –М. «Вильямс», 2003. – 1088 С.

ПРИМЕНЕНИЕ КОМПОНЕНТОВ .NET В СОЗДАНИИ ОТКАЗОУСТОЙЧИВОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Маймистов Д.С.

Сибирский государственный аэрокосмический университет им. академика М.Ф. Решетнева

Концепцию *мультиверсионного программирования* (МВП, multi-version programming – MVP), или *N-версионного программирования* (НВП, N-version programming – NVP) впервые представил Альгирдас Авижиенис в 1977. Основная идея МВП заключается в том, что для решения отдельных подзадач системы, используется несколько версий одного алгоритма, выполняющихся одновременно. Результаты работы этих алгоритмов анализируются, и из них выбирается один наиболее удовлетворяющий потребностям системы на данный момент времени. Выбор производится согласно внутренней логике системы. Таким образом, достигается повышение надёжности системы в целом. Различных подзадача в сложных системах, реализация, которых использует концепцию мультиверсионного программирования, может быть огромное множество. Очевидно, что для разработки таких систем необходима общая концепция и общий подход в написании алгоритмов, решающих её отдельные подзадачи. На эту роль как нельзя лучше подходит методика компонентного программирования.

Такой метод создания программного обеспечения, как компонентное программирование, появился относительно недавно. Его можно охарактеризовать как технологию создания программного обеспечения из готовых блоков. То есть программисты пытаются использовать идеи строителей, занимающихся крупнопанельным домостроением. Создание программного обеспечения из компонентов подразумевает, что компоненты будут добавляться к проекту во время разработки. При этом будет производиться их начальная настройка. Компоненты как таковые не подразумевают пользовательского интерфейса (ни для программиста, ни для конечного пользователя). В этом качестве выступают части интегрированной среды разработки и дополнительные программные дизайнеры. Первой компонентной средой был продукт, разработанный корпорацией Microsoft на заре своего существования. Впоследствии на его базе были разработаны множество других сред. Таким образом, к концу двадцатого века, компоненты стали поддерживаться почти всеми производителями интегрированных сред.

Самой развитой и совершенной компонентной моделью на сегодняшний день, является модель предложенной корпорацией Microsoft и реализованной ею в новой технологии .NET.

Определение компонента в понимании Microsoft - это объединенные в отчуждаемую форму исполняемый бинарный код и данные, которые могут использоваться для построения программных систем. Отчуждаемость подразумевает возможность использования компонента без дополнительных знаний о нем. На практике это означает, что компонент сам должен содержать сведения о себе. Компонент должен также иметь внешний (публичный) интерфейс. Интерфейс