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

является как бы механизмом, через который можно запустить находящийся внутри компонента код. Отчуждаемость также означает, что экземпляр компонента может быть создан динамически, и что для этого не обязательно использовать всякого рода компиляторы и интерпретаторы.

По сути компонент - это класс, предоставляющий информацию о себе (метаинформацию), экземпляры которого можно создавать динамически (не имея никакой статической информации о нем).

Практически любой класс в .NET отвечает этим требованиям - метаинформация создаются для любого элемента класса (будь он трижды скрытым), экземпляр любого класса можно динамически создать, и любой класс помещается в сборки (один или более исполнимых модулей), которые можно распространять независимо. Таким образом, получается, что любой класс в .NET может выступать как компонент. Но на самом деле это не так. И причиной тому наличие в библиотеке .NET отдельного класса Component. Любой класс, что бы иметь возможность взаимодействовать с интегрированной средой разработки должен быть унаследован от класса Component.

На основе выше приведённого описания основных концепций компонентной модели .NET, можно сделать вывод о том что .NET компоненты обладают следующими преимуществами по сравнению с компонентами, в основе которых лежат иные концепция и технология:

- Возможность интегрировать компонент в любую среду разработки, поддерживающую соответствующие стандарты Microsoft
- Возможность написания и распространения компонент сторонними разработчиками
- Возможность написания компонент в различных средах разработки и на различных языках программирования, поддерживающих соответствующие стандарты Microsoft

Таким образом, становится очевидным выбор в пользу использования компонентной технологии .NET, для разработки мультиверсионных компонент.

СПИСОК ЛИТЕРАТУРЫ

1. Владислав Чистяков. «.Net – классы, компоненты и контролы» RSDN Magazine №3 2003г.

2. Котенок А.В. Построение среды мультиверсионного исполнения программных модулей. Вестник НИИ СУВПТ: Сб. научн. трудов; Красноярск: НИИ СУВПТ.- 2003. Вып. 14.- С. 13-21.

ЗАДАЧА БАЛАНСИРОВКИ ТРАФИКА ДЛЯ ОБЕСПЕЧЕНИЯ БЕСПЕРЕБОЙНОЙ РАБОТЫ СЕТИ

Подерский И.С., Кравец О.Я.

Одна из проблем, возникающая в работе сети с коммутацией пакетов – перегрузка её отдельных участков, которая в свою очередь может парализовать работу всей сети. Повышения надежности можно достичь, равномерно распределив нагрузку на каналы и узлы сети. В том случае, когда нагрузка распределена равномерно на все узлы и каналы, будет достигнут максимальный резерв производительности.

Каждый канал в сети характеризуется своей пропускной способностью q_{ij}. Тогда загрузка определяется как $r_{ij} = \frac{l_{ij}}{mq_{ij}}$, где l_{ij} - поток по соответствующей дуге, а $\frac{1}{m}$ - средняя длина пакета. Если в сети M каналов, то средняя загрузка сети, имеет вид: $r_{cp} = \frac{1}{M}\sum_{i=1}^{M}\frac{l_{ij}}{mq_{ij}}$.

Для обеспечения равномерной загрузки каналов нужно минимизировать дисперсию загрузки каждого канала относительно средней загрузки. То есть:

$$\sum_{i=1}^{N}\sum_{j\in\Gamma^{+}(x_i)}\left(\frac{1}{m}\frac{l_{ij}}{q_{ij}} - r_{cp}\right)^2 \to \min$$

здесь N - количество узлов в сети, $\Gamma^{+}(x_i)$ - множество входящих в x_i дуг ($\Gamma^{-}(x_i)$ - множество исходящих из x_i дуг).

Для обеспечения равномерной загрузки узлов достаточно представить каждый узел x_{ij} в виде пары узлов x_{ij}^{+} и x_{ij}^{-}. Узлу x_{ij}^{+} будут инцидентны все входящие дуги узла x_{ij}, а узлу x_{ij}^{-} - все исходящие. Дуге соединяющей узлы x_{ij}^{+} и x_{ij}^{-} нужно назначить пропускную способность, соответствующую производительности узла x_{ij}^{-}. Теперь задача обеспечения равномерной загрузки узлов сведена к обеспечению равномерной загрузки каналов.

При балансировке трафика по каналам необходимо соблюсти требования сохранения потоков в сети и ограничение трафика пропускной способностью канала. Условие ограничения трафика пропускной способностью описывается неравенствами вида:

$$\forall i, j = 1..N,\ \ l_{ij} \le q_{ij}$$

Обозначим через K_i и L_i - соответственно, трафик, порожденный i-м узлом и трафик, предназначенный i-му узлу. Тогда условие сохранения потока будет иметь вид:

$$\forall i = 1..N,$$

$$\sum_{j\in\Gamma^{+}(x_i)} l_{ij} = K_i - L_i + \sum_{j\in\Gamma^{-}(x_i)} l_{ij},$$