3.21. Распределенные СУБД. Распределенные транзакции. Алгоритмы реализации распределенных транзакций. Вертикальное и горизонтальное разделение данных (partitioning).

Распределенные базы данных

Под распределенной (Distributed DataBase - DDB) обычно подразумевают базу данных, включающую фрагменты из нескольких баз данных, которые располагаются на различных узлах сети компьютеров, и, возможно управляются различными СУБД. Распределенная база данных выглядит с точки зрения пользователей и прикладных программ как обычная локальная база данных. В этом смысле слово "распределенная" отражает способ организации базы данных, но не внешнюю ее характеристику. ("распределенность" базы данных невидима извне).

Определение Дэйта.

Лучшее, на мой взгляд, определение распределенных баз данных (DDB) предложил Дэйт (C.J. Date) в [1]. Он установил 12 свойств или качеств идеальной DDB:

- Локальная автономия (local autonomy)
- Отсутствие опоры на центральный узел(no reliance on central site)
- Непрерывное функционирование (continuous operation)
- Независимость от расположения (location independence)
- Независимость от фрагментации (fragmentation independence)
- Независимость от репликации (replication independence)
- Обработка распределенных запросов (distributed query processing)
- Управление распределенными транзакциями (distributed transaction processing)
- Аппаратная независимость (hardware independence)
- Независимость от операционных систем (operationg system independence)
- Независимость от сети (network independence)
- Независимость от типа баз данных (database independence)

Локальная автономия

Это качество означает, что управление данными на каждом из узлов распределенной системы выполняется локально. База данных, расположенная на одном из узлов, является неотъемлемым компонентом распределенной системы. Будучи фрагментом общего пространства данных, она, в то же время функционирует как полноценная локальная база данных; управление ею выполняется локально и независимо от других узлов системы.

Отсутствие опоры на центральный узел

В идеальной системе все узлы равноправны и независимы, а расположенные на них базы являются равноправными поставщиками данных в общее пространство данных. База данных на каждом из узлов самодостаточна - она включает полный собственный словарь данных и полностью защищена от несанкционированного доступа. Выполнение первого пункта автоматически гарантирует выполнение этого пункта.

Непрерывное функционирование

Это качество можно трактовать как возможность непрерывного доступа к данным (известное "24 часа в сутки, семь дней в неделю") в рамках DDB вне зависимости от их расположения и вне зависимости от операций, выполняемых на локальных узлах. Это качество можно выразить лозунгом "данные доступны всегда, а операции над ними выполняются непрерывно".

Независимость от расположения

Это свойство означает полную прозрачность расположения данных. Пользователь, обращающийся к DDB, ничего не должен знать о реальном, физическом размещении данных в узлах информационной системы. Все операции над данными выполняются без учета их местонахождения. Транспортировка запросов к базам данных осуществляется встроенными системными средствами.

Независимость от фрагментации

Это свойство трактуется как возможность распределенного (то есть на различных узлах) размещения данных, логически представляющих собой единое целое. Существует фрагментация двух типов: *горизонтальная* и *вертикальная*. Первая означает хранение строк одной таблицы на различных узлах (фактически, хранение строк одной логической таблицы в нескольких идентичных физических таблицах на различных узлах). Вторая означает распределение столбцов логической таблицы по нескольким узлам.

Рассмотрим пример, иллюстрирующий оба типа фрагментации. Имеется таблица employee (emp_id, emp_name, phone), определенная в базе данных на узле в Фениксе. Имеется точно такая же таблица, определенная в базе данных на узле в Денвере. Обе таблицы хранят информацию о сотрудниках компании. Кроме того, в базе данных на узле в Далласе определена таблица emp_salary (emp_id, salary). Тогда запрос "получить информацию о сотрудниках компании" может быть сформулирован так:

```
SELECT * FROM employee@phoenix, employee@denver ORDER BY emp_id
```

В то же время запрос "получить информацию о заработной плате сотрудников компании" будет выглядеть следующим образом:

```
SELECT employee.emp_id, emp_name, salary FROM employee@denver, employee@phoenix, emp_salary@dallas ORDER BY emp_id
```

Независимость от репликации

Тиражирование данных - это асинхронный (в общем случае) процесс переноса изменений объектов исходной базы данных в базы, расположенные на других узлах распределенной системы. В данном контексте прозрачность тиражирования означает возможность переноса изменений между базами данных средствами, невидимыми пользователю распределенной системы. Данное свойство означает, что тиражирование возможно и достигается внутрисистемными средствами.

Обработка распределенных запросов

Это свойство DDB трактуется как возможность выполнения операций выборки над распределенной базой данных, сформулированных в рамках обычного запроса на языке

SQL. То есть операцию выборки из DDB можно сформулировать с помощью тех же языковых средств, что и операцию над локальной базой данных.

Управление распределенными транзакциями

Это качество DDB можно трактовать как возможность выполнения операций обновления распределенной базы данных (INSERT, UPDATE, DELETE), не разрушающее целостность и согласованность данных. Эта цель достигается применением двухфазового или двухфазного протокола фиксации транзакций (two-phase commit protocol), ставшего фактическим стандартом обработки распределенных транзакций. Его применение гарантирует согласованное изменение данных на нескольких узлах в рамках распределенной (или, как ее еще называют, глобальной) транзакции.

Аппаратная независимость

Это свойство означает, что в качестве узлов распределенной системы могут выступать компьютеры любых моделей и производителей - от мэйнфреймов до "персоналок".

Независимость от операционных систем

Это качество вытекает из предыдущего и означает многообразие операционных систем, управляющих узлами распределенной системы.

Независимость от сети

Доступ к любым базам данных может осуществляться по сети. Спектр поддерживаемых конкретной СУБД сетевых протоколов не должен быть ограничением системы с распределенными базами данных. Данное качество формулируется максимально широко - в распределенной системе возможны любые сетевые протоколы.

Независимость от типа баз данных

Это качество означает, что в распределенной системе могут мирно сосуществовать СУБД различных производителей, и возможны операции поиска и обновления в базах данных различных моделей и форматов.

Исходя из определения Дэйта, можно рассматривать DDB как слабосвязанную сетевую структуру, узлы которой представляют собой локальные базы данных. Локальные базы данных автономны, независимы и самоопределены; доступ к ним обеспечиваются СУБД, в общем случае от различных поставщиков. Связи между узлами - это потоки тиражируемых данных. Топология DDB варьируется в широком диапазоне - возможны варианты иерархии, структур типа "звезда" и т.д. В целом топология DDB определяется географией информационной системы и направленностью потоков тиражирования данных.

Посмотрим, во что выливается некоторые наиболее важные свойства DDB, если рассматривать их практически.

Целостность данных

В DDB поддержка целостности и согласованности данных, ввиду свойств 1-2, представляет собой сложную проблему. Ее решение - синхронное и согласованное изменение данных в нескольких локальных базах данных, составляющих DDB - достигается применением протокола двухфазной фиксации транзакций. Если DDB однородна - то есть на всех узлах данные хранятся в формате одной базы и на всех узлах функционирует одна и та же СУБД, то используется механизм двухфазной фиксации транзакций данной СУБД. В случае же неоднородности DDB для обеспечения согласованных изменений в нескольких базах данных используют менеджеры распределенных транзакций. Это, однако, возможно, если участники обработки распределенной транзакции - СУБД, функционирующие на узлах системы, поддерживают XA-интерфейс, определенный в спецификации DTP консорциума X/Open. В настоящее время XA-интерфейс имеют CA-OpenIngres, Informix, Microsoft SQL Server, Oracle, Sybase.

Если в DDB предусмотрено тиражирование данных, то это сразу предъявляет дополнительные жесткие требования к дисциплине поддержки целостности данных на узлах, куда направлены потоки тиражируемых данных. Проблема в том, что изменения в данных инициируются как локально - на данном узле - так и извне, посредством тиражирования. Неизбежно возникают конфликты по изменениям, которые необходимо отслеживать и разрешать.

Прозрачность расположения

Это качество DDB в реальных продуктах должно поддерживаться соответствующими механизмами. Разработчики СУБД придерживаются различных подходов. Рассмотрим пример из Oracle. Допустим, что DDB включает локальную базу данных, которая размещена на узле в Лондоне. Создадим вначале ссылку (database link), связав ее с символическим именем (london_unix), транслируемым в IP-адрес узла в Лондоне.

CREATE PUBLIC DATABASE LINK london.com CONNECT TO london_unix USING oracle_user_ID;

Теперь мы можем явно обращаться к базе данных на этом узле, запрашивая, например, в операторе SELECT таблицу, хранящуюся в этой базе:

SELECT customer.cust_name, order.order_date FROM customer@london.com, order WHERE customer.cust_number = order.cust_number;

Очевидно, однако, что мы написали запрос, зависящий от расположения базы данных, поскольку явно использовали в нем ссылку. Определим customer и customer@london.com как синонимы:

CREATE SYNONYM customer FOR customer@london.com;

и в результате можем написать полностью независимый от расположения базы данных запрос:

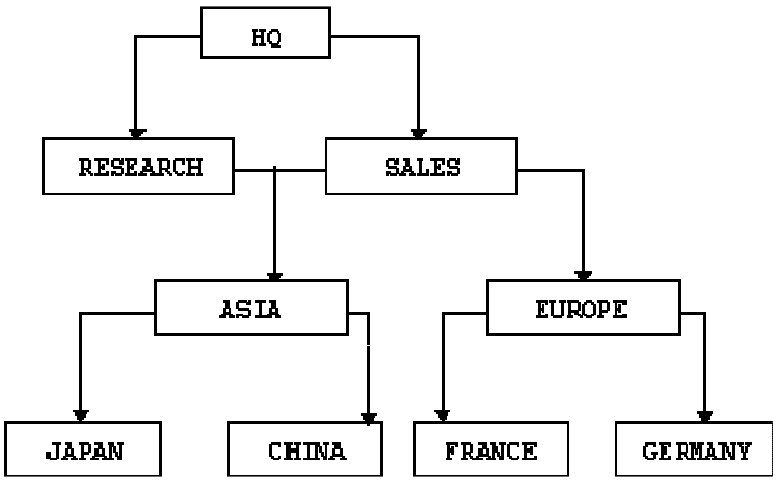
SELECT customer.cust_name, order.order_date FROM customer, order WHERE customer.cust_number = order.cust_number

Задача решается с помощью оператора SQL CREATE SYNONYM, который позволяет создавать новые имена для существующих таблиц. При этом оказывается возможным обращаться к другим базам данных и к другим компьютерам. Так, запись в СУБД Informix

CREATE SYNONYM customer FOR client@central:smith.customer

означает, что любое обращение к таблице customer в открытой базе данных будет автоматически переадресовано на компьютер central в базу данных client к таблице customer. Оказывается возможным переместить таблицу из одной базы данных в другую, оставив в первой базе ссылку на ее новое местонахождение, при этом все необходимые действия для доступа к содержимому таблицы будут сделаны автоматически.

Мы уже говорили выше о горизонтальной фрагментации. Рассмотрим пример иерархически организованной DDB, на каждом из узлов которой содержится некоторое подмножество записей таблицы customer:



С помощью CREATE SYNONYM можно определить, например, таблицу структуры customer, в которой хранятся строки с записями о клиентах компании, находящихся в Японии:

CREATE SYNONYM japan_customer FOR customer@hq.sales.asia.japan

Во многих СУБД задача управления именами объектов DDB решается путем использования глобального словаря данных, хранящего информацию о DDB: расположение данных, возможности других СУБД (если используются шлюзы), сведения о скорости передачи по сети с различной топологией и т.д.

Обработка распределенных запросов

Выше уже упоминалось это качество DDB. Обработка распределенных запросов (Distributed Query -DQ) - задача, более сложная, нежели обработка локальных и она требует интеллектуального решения с помощью особого компонента - оптимизатора DQ. Обратимся к базе данных, распределенной по двум узлам сети. Таблица detail хранится на

одном узле, таблица supplier - на другом. Размер первой таблицы - 10000 строк, размер второй - 100 строк (множество деталей поставляется небольшим числом поставщиков). Допустим, что выполняется запрос:

```
SELECT detail_name, supplier_name, supplier_address
FROM detail, supplier
WHERE detail.supplier_number = supplier.supplier_number;
```

Результирующая таблица представляет собой объединение таблиц detail и supplier, выполненное по столбцу detail.supplier_number (внешний ключ) и supplier.supplier_number (первичный ключ).

Данный запрос - распределенный, так как затрагивает таблицы, принадлежащие различным локальным базам данных. Для его нормального выполнения необходимо иметь обе исходные таблицы на одном узле. Следовательно, одна из таблиц должна быть передана по сети. Очевидно, что это должна быть таблица меньшего размера, то есть таблица supplier. Следовательно, оптимизатор DQ запросов должен учитывать такие параметры, как, в первую очередь, размер таблиц, статистику распределения данных по узлам, объем данных, передаваемых между узлами, скорость коммуникационных линий, структуры хранения данных, соотношение производительности процессоров на разных узлах и т.д. От интеллекта оптимизатора DQ впрямую зависит скорость выполнения распределенных запросов.

Межоперабельность

В контексте DDB межоперабельность означает две вещи. Во-первых, - это качество, позволяющее обмениваться данными между базами данных различных поставщиков. Как, например, тиражировать данные из базы данных Informix в Oracle и наоборот? Известно, что штатные средства тиражирования в составе данной конкретной СУБД позволяют переносить данные в однородную базу. Так, средствами CA-Ingres/Replicator можно тиражировать данные только из Ingres в Ingres. Как быть в неоднородной DDB? Ответом стало появление продуктов, выполняющих тиражирование между разнородными базами данных.

Во-вторых, это возможность некоторого унифицированного доступа к данным в DDB из приложения. Возможны как универсальные решения (стандарт ODBC), так и специализированные подходы. Очевидный недостаток ODBC - недоступность для приложения многих полезных механизмов каждой конкретной СУБД, поскольку они могут быть использованы в большинстве случаев только через расширения SQL в диалекте языка данной СУБД, но в стандарте ODBC эти расширения не поддерживаются.

Специальные подходы - это, например, использование шлюзов, позволяющее приложениям оперировать над базами данных в "чужом" формате так, как будто это собственные базы данных. Вообще, цель шлюза - организация доступа к унаследованным (legacy) базам данных и служит для решения задач согласования форматов баз данных при переходе к какой-либо одной СУБД. Так, если компания долгое время работала на СУБД IMS и затем решила перейти на Oracle, то ей очевидно потребуется шлюз в IMS. Следовательно, шлюзы можно рассматривать как средство, облегчающее миграцию, но не как универсальное средство межоперабельности в распределенной системе. Вообще, универсального рецепта решения задачи межоперабельности в этом контексте не существует - все определяется конкретной ситуацией, историей информационной системы и массой других факторов. DDB конструирует архитектор, имеющий в своем арсенале отработанные интеграционные средства, которых на рынке сейчас очень много.

Технология тиражирования данных

Принципиальная характеристика тиражирования данных (Data Replication - DR) заключается в отказе от физического распределения данных. Суть DR состоит в том, что любая база данных (как для СУБД, так и для работающих с ней пользователей) всегда является локальной; данные размещаются локально на том узле сети, где они обрабатываются; все транзакции в системе завершаются локально.

Тиражирование данных - это асинхронный перенос изменений объектов исходной базы данных в базы, принадлежащим различным узлам распределенной системы. Функции DR выполняет, как правило, специальный модуль СУБД - сервер тиражирования данных, называемый репликатором (так устроены СУБД CA-OpenIngres и Sybase). В Informix-OnLine Dynamic Server репликатор встроен в сервер, в Oracle 7 для использования DR необходимо приобрести дополнительно к Oracle7 DBMS опцию Replication Option.

Специфика механизмов DR зависит от используемой СУБД. Простейший вариант DR - использование "моментальных снимков" (snapshot). Рассмотрим пример из Oracle:

```
CREATE SNAPSHOT unfilled_orders
REFRASH COMPLETE
START WITH TO_DATE ('DD-MON-YY HH23:MI:55')
NEXT SYSDATE + 7
AS SELECT customer_name, customer_address, order_date
FROM customer@paris, order@london
WHERE customer.cust_name = order.customer_number AND
order_complete_flag = "N";
```

"Моментальный снимок" в виде горизонтальной проекции объединения таблиц customer и order будет выполнен в 23:55 и будет повторятся каждые 7 дней. Каждый раз будут выбираться только завершенные заказы.

Реальные схемы тиражирования, разумеется, устроены более сложно. В качестве базиса для тиражирования выступает транзакция к базе данных. В то же время возможен перенос изменений группами транзакций, периодически или в некоторый момент времени, что дает возможность исследовать состояние принимающей базы на определенный момент времени.

Детали тиражирования данных полностью скрыты от прикладной программы; ее функционирование никак не зависят от работы репликатора, который целиком находится в ведении администратора базы данных. Следовательно, для переноса программы в распределенную среду с тиражируемыми данными не требуется ее модификации. В этом, собственно, состоит качество 6 в определении Дэйта.

Синхронное обновление DDB и DR-технология - в определенном смысле антиподы. Краеугольный камень первой - синхронное завершение транзакций одновременно на нескольких узлах распределенной системы, то есть синхронная фиксация изменений в DDB. Ее "Ахиллесова пята" - жесткие требования к производительности и надежности каналов связи. Если база данных распределена по нескольким территориально удаленным узлам, объединенным медленными и ненадежными каналами связи, а число одновременно работающих пользователей составляет сотни и выше, то вероятность того, что распределенная транзакция будет зафиксирована в обозримом временном интервале, становится чрезвычайно малой. В

таких условиях (характерных, кстати, для большинства отечественных организаций) обработка распределенных данных практически невозможна.

DR-технология не требует синхронной фиксации изменений, и в этом ее сильная сторона. В действительности далеко не во всех задачах требуется обеспечение идентичности БД на различных узлах в любое время. Достаточно поддерживать тождественность данных лишь в определенные критичные моменты времени. Можно накапливать изменения в данных в виде транзакций в одном узле и периодически копировать эти изменения на другие узлы.

Налицо преимущества DR-технологии. Во-первых, данные всегда расположены там, где они обрабатываются - следовательно, скорость доступа к ним существенно увеличивается. Во-вторых, передача только операций, изменяющих данные (а не всех операций доступа к удаленным данным), и к тому же в асинхронном режиме позволяет значительно уменьшить трафик. В-третьих, со стороны исходной базы для принимающих баз репликатор выступает как процесс, инициированный одним пользователем, в то время как в физически распределенной среде с каждым локальным сервером работают все пользователи распределенной системы, конкурирующие за ресурсы друг с другом. Наконец, в-четвертых, никакой продолжительный сбой связи не в состоянии нарушить передачу изменений. Дело в том, что тиражирование предполагает буферизацию потока изменений (транзакций); после восстановления связи передача возобновляется с той транзакции, на которой тиражирование было прервано.

DR-технология данных не лишена недостатков. Например, невозможно полностью исключить конфликты между двумя версиями одной и той же записи. Он может возникнуть, когда вследствие все той же асинхронности два пользователя на разных узлах исправят одну и ту же запись в тот момент, пока изменения в данных из первой базы данных еще не были перенесены во вторую. При проектировании распределенной среды с использованием DR-технологии необходимо предусмотреть конфликтные ситуации и запрограммировать репликатор на какой-либо вариант их разрешения. В этом смысле применение DR-технологии - наиболее сильная угроза целостности DDB. На мой взгляд, DR-технологию нужно применять крайне осторожно, только для решения задач с жестко ограниченными условиями и по тщательно продуманной схеме, включающей осмысленный алгоритм разрешения конфликтов.