#### Распределённые базы данных

#### Введение

В условиях современного динамического развития общества и усложнения технической и социальной инфраструктуры информация становится таким же стратегическим ресурсом, как традиционные материальные и энергетические ресурсы [1]. Современные информационные технологии, позволяющие создавать, хранить, перерабатывать и обеспечивать эффективные способы представления информационных ресурсов потребителю, стали важным фактором жизни общества и средством повышения эффективности управления всеми сферами общественной деятельности. Уровень использования информации становится одним из существенных факторов успешного экономического развития и конкурентоспособности региона как на внутреннем, так и на внешнем рынке.

Осознание мировым сообществом роли информации как стратегического ресурса стимулировало разработки новых информационных технологий для получения и переработки больших объемов информации и ее хранения и предоставления пользователям. Первое место среди новых технологий занимают сетевые информационные технологии.

Информация является важнейшим стратегическим ресурсом, и наибольший экономический и социальный успех сегодня сопутствует тем странам, которые активно используют современные средства компьютерных коммуникаций и сетей, информационных технологий и систем управления информационными ресурсами. Перенесенные на электронные носители информационные ресурсы приобретают качественно новое состояние и становятся активными. Доступная для оперативного воспроизводства средствами компьютерной обработки информация является важнейшим фактором социального развития общества.

В настоящее время наиболее развитые страны мира находятся на завершающей стадии индустриального этапа развития общества и перехода к следующему, информационному, этапу развития и построения "Информационного общества" (ИО). Широкое использование информационных технологий и современных средств доступа к информации открыло принципиально иные возможности построения более сбалансированного общества, с существенно большей реализацией индивидуальных возможностей его членов. "Информационное общество" несет в себе огромный потенциал для улучшения жизни граждан и повышения эффективности социального и экономического устройства государства. Стоящий перед Россией как и перед всем миром выбор прост: либо использовать преимущества зарождающегося ИО, сводя при этом к минимуму возможные потери, либо отдаться во власть революционной стихии, вызванной информационным кризисом.

Использование Internet/Intranet технологий при создании информационных ресурсов и построении информационных систем различного назначения в последнее время стало доминирующим в мировом информационном пространстве по следующим причинам [2]. Эти технологии:

* Позволяют организовать с достаточной простотой для пользователя системы поиска нужной информации.
* Предъявляют минимальные требования как с технической стороны так и со стороны программного обеспечения к рабочему месту клиента (клиент работает со стандартным программным обеспечением и единственным требованием является поддержка работы WWW просмотрщика -- браузера одной из последних версий1).
* Поддерживают распределенные системы хранения информации и множественные методы ее хранения.
* Поддерживают работу с практически неограниченным объемом разноплановых данных (текст, графика, изображение, звук, видео, векторные карты и др.).
* Предоставляют технологически простой способ администрирования информационных систем с одного рабочего места.
* Поддерживают удаленные методы редактирования и пополнения информации.

#### 1 История вопроса

История создания компьютерных информационных систем насчитывает несколько десятилетий. За это время были созданы системы по автоматизации деятельности банков, статистических бюро, промышленных предприятий, контор, агентств по бронированию и продаже билетов и т.д. Однако бурная деятельность по созданию новых систем автоматизации не только не утихает, но и переживает в последнее время заметное оживление. Эта ситуация связана с все возрастающим значением систем обработки информации для выживания компаний в условиях высокой конкуренции, уменьшением удельной стоимости таких систем, с развитием технологий обработки и хранения информации, а также качественным изменением ситуации с развитием технологий передачи данных, в частности Internet.

Первые ИС создавались для больших ЭВМ и имели унитарную структуру, т.е. представляли собой по сути одну программу, включающую в себя все функции по хранению данных, их обработке и представлению, а также по контролю доступа к данным со стороны пользователей системы. Такая организация ИС имеет ряд достоинств. Это, в частности, централизованное хранение и обработка информации, простота администрирования системы, а также очень эффективное использование вычислительных ресурсов - для выполнения важных задач может быть выделена вся мощь вычислительной системы.

Однако обработка информации может быть осуществлена и на другой машине. Такой подход позволил создать на базе ПК и рабочих станций системы распределенной обработки информации. Первыми из этого класса систем стали системы, построенные по архитектуре файл-сервер. Основной особенностью этой архитектуры явился полный отказ от централизованных вычислений. Файл-сервер выполнял лишь функции хранения данных и не принимал участия в их обработке - эта работа была возложена на клиентские машины.

Однако очень скоро стало ясно, что полный отказ от централизованного контроля данных таит в себе ряд серьезных проблем. Проблемы эти состоят уже не в отказе отдельных компонентов системы, а в логике их совместной работы. Каждое из приложений, работающее с общими данными должно придерживаться ряда весьма жестких ограничений и соглашений, обеспечивающих целостность информации при ее модификации различными модулями системы. На контроль целостности данных приходилось весьма существенная доля программного кода системы, вычислительного ресурса клиентских машин и сетевого трафика, и, тем не менее, оставались проблемы, например сбой на клиентской машине в середине выполнения операции мог привести к рассогласованию данных.

Выходом из создавшейся ситуации стала разработка концепции клиент-серверных вычислений, сочетающей в себе преимущества централизованной обработки данных унитарных систем с преимуществами распределенных вычислений систем типа файл-сервер. Ключевым отличием архитектуры клиент-сервер от архитектуры файл-сервер является абстрагирование от внутреннего представления данных (физической схемы данных). Теперь клиентские программы манипулируют данными на уровне логической схемы. Они уже не заботятся о построении индексов для ускорения выборки данных, о распределении данных по файлам, выставлении семафоров на обрабатываемые записи и т.д.

Все рутинные функции по хранению, обработке и защите данных на так называемом физическом уровне берет на себя система управления базой данных (СУБД). Со времени своего появления СУБД также активно эволюционировали, предлагая различные модели логического представления данных (иерархические, сетевые, реляционные, объектно- ориентированные). Но суть дела от этого не меняется - программе, выполняющей бизнес-функции ИС, уже не надо заботится о том, как и где хранятся данные, следить за их достаточностью и непротиворечивостью, обеспечивать условия по безопасному совместному пользованию данными несколькими пользователями. Она лишь запрашивает СУБД о предоставлении требуемых данных.

Еще одним преимуществом использования СУБД и архитектуры клиент-сервер по сравнению с файл-серверным подходом явилась возможность использовать транзакционный механизм манипулирования данными. Этот сервис, предоставляемый сервером данных, позволяет объединять несколько действий по изменению данных в одну неделимую операцию (транзакцию). Использование транзакций обеспечивает надежную защиту информации от программно-аппаратных сбоев как на клиентской, так и на серверной части ИС.

Помимо улучшения работоспособности уже готовых программ, архитектура клиент-сервер существенно облегчает и процесс создания ИС. Как уже было отмечено, прикладному программисту теперь не надо отвлекаться от описания логики работы системы на проблемы хранения данных, индексации таблиц и т.п. Транзакционный механизм позволяет не заботится о порядке модификации данных внутри одной транзакции и о способах восстановления их первоначального состояния при обнаружении исключительных ситуаций. А это в свою очередь сильно расширяет возможности командной иерархической разработки проекта - программисту уже не нужно знать внутреннее устройство функций написанных другими людьми - он может просто пользоваться ими без риска отказа работоспособности уже отлаженных модулей. Кроме того, использование в прикладных программах логического уровня представления данных и использование стандартизованных механизмов запроса к СУБД позволило писать платформо-независимые программы клиентской части ИС.

Итак, использование архитектуры клиент-сервер позволило создавать надежные (в смысле целостности данных) многопользовательские ИС с распределенной базой данных (РБД), поддерживающие графический интерфейс пользователя (ГИП) на клиентских станциях, связанных коммуникационной сетью. Основной составной единицей таких систем стали РБД.

#### 2 Основные принципы, правила построения и функционирования РБД

РБД состоит из набора узлов, связанных коммуникационной сетью, основной задачей которой является передача данных без ошибок и искажения. Коммуникационная сеть является ядром информационной сети, обеспечивающим передачу и некоторые виды обработки данных.

Коммуникационной сети присущи следующие свойства:

* каждый узел — это полноценная СУБД сама по себе;
* узлы взаимодействуют между собой таким образом, что пользователь любого из них может получить доступ к любым данным в сети так, как будто они находятся на его собственном узле.

Каждый узел сам по себе является СУБД. Любой пользователь может выполнить операции над данными на своём локальном узле точно так же, как если бы этот узел вовсе не входил в распределённую систему. Распределённую систему баз данных можно рассматривать как партнёрство между отдельными локальными СУБД на отдельных локальных узлах.

Дадим следующее определение: распределенная база данных — это набор файлов (отношений), хранящихся в разных узлах информационной сети и логически связанных таким образом, чтобы составлять единую совокупность данных (связь может быть функциональной или через копии одного и того же файла) [3].

Распределенная база данных предполагает хранение и выполнение функций управления данными в нескольких узлах и передачу данных между этими узлами в процессе выполнения запросов. Разбиение данных в распределенной базе данных может достигаться путем хранения различных таблиц на разных компьютерах или даже хранения разных частей и фрагментов одной таблицы на разных компьютерах. Для пользователя (или прикладной программы) не должно иметь значения, каким образом распределены данные между компьютерами. Работать с распределенной базой данных, если она действительно распределенная, следует так же, как и с централизованной, т. е. размещение базы данных должно быть прозрачно.

Несмотря на то, что распределенная база данных состоит из нескольких локальных баз данных, у пользователя должна сохраняться иллюзия работы с централизованной базой данных, что вызывает потребность в использовании некоторого общего представления о данных — глобальной концептуальной схемы. Определение данных в такой концептуальной схеме должно быть аналогичным определению в централизованной базе данных.

Отличия начинаются, когда требуется хранить данные в нескольких узлах. Чтобы произвести разбиение данных, нужно секционировать таблицы глобальной схемы на фрагменты. Существует два типа секционирования: горизонтальное и вертикальное. При секционировании таблицы по строкам выполняется горизонтальное секционирование, при разбиении по столбцам — вертикальное.

Таким образом, архитектура распределенной СУБД должна содержать информацию о секционировании исходных таблиц базы данных, что предполагает создание дополнительного уровня — фрагментного.

Самый высший уровень архитектуры распределенной СУБД — это интерфейс прикладной программы и интерфейс процессора запросов.

Взгляд на базу данных отдельных пользователей представлен в архитектуре отдельным 1-м уровнем, что аналогично внешнему уровню в классической архитектуре СУБД.

Для реализации и объяснения распределенной природы базы данных выделяются два уровня: фрагментный (см. выше) и уровень распределенного представления. Последний показывает географическое распределение данных по рабочим станциям, расположение экземпляра каждого фрагмента.

1—4 уровни архитектуры распределенной СУБД относятся к сетевой СУБД.

Однако выделяют еще локальные СУБД, где определяют представление данных на каждой рабочей станции.

Каждый уровень поддерживает различные представления базы данных; каждый уровень взаимодействует только со смежными уровнями представления.

Для управления распределенной базой данных создается программный комплекс – система управления распределенной базой данных (СУРБД).

C. J. Date в 1987 году сформулировал один основной принцип и двенадцать правил, которым, по его мнению, должны следовать распределенные базы данных [4].

Основной принцип заключается в "прозрачности распределенности". С точки зрения пользователя информации распределенная БД не должна отличаться от локальной БД. Здесь имеется в виду пользователь, формулирующий запросы к базе данных и получающий результаты на своем экране (или принтере). Технология его работы не должна зависеть от того, в какой конкретной базе находятся нужные ему данные: в его локальной базе, в удаленной базе, все необходимые данные в одной и той же базе или в различных базах данных.

Фундаментальный принцип имеет следствием определённые дополнительные правила или цели. Таких целей всего двенадцать:

1. Локальная независимость. Узлы в распределённой системе должны быть независимы, или автономны. Локальная независимость означает, что все операции на узле контролируются этим узлом.
2. Отсутствие опоры на центральный узел. Локальная независимость предполагает, что все узлы в распределённой системе должны рассматриваться как равные. Поэтому не должно быть никаких обращений к «центральному» или «главному» узлу с целью получения некоторого централизованного сервиса.
3. Непрерывное функционирование. Распределённые системы должны предоставлять более высокую степень надёжности и доступности.
4. Независимость от расположения. Пользователи не должны знать, где именно данные хранятся физически и должны поступать так, как если бы все данные хранились на их собственном локальном узле.
5. Независимость от фрагментации (дробления данных на множество мелких разрозненных фрагментов). Система поддерживает независимость от фрагментации, если данная переменная-отношение может быть разделена на части или фрагменты при организации её физического хранения. В этом случае данные могут храниться в том месте, где они чаще всего используются, что позволяет достичь локализации большинства операций и уменьшения сетевого трафика.
6. Независимость от репликации. Система поддерживает репликацию данных, если данная хранимая переменная-отношение — или в общем случае данный фрагмент данной хранимой переменной-отношения — может быть представлена несколькими отдельными копиями или репликами, которые хранятся на нескольких отдельных узлах.
7. Обработка распределённых запросов. Суть в том, что для запроса может потребоваться обращение к нескольким узлам. В такой системе может быть много возможных способов пересылки данных, позволяющих выполнить рассматриваемый запрос.
8. Управление распределёнными транзакциями (последовательность операций, представляющая собой логическую единицу работы с данными). Существует 2 главных аспекта управления транзакциями: управление восстановлением и управление параллельностью обработки. Что касается управления восстановлением, то чтобы обеспечить атомарность транзакции в распределённой среде, система должна гарантировать, что все множество относящихся к данной транзакции агентов (агент — процесс, который выполняется для данной транзакции на отдельном узле) или зафиксировало свои результаты, или выполнило откат. Что касается управления параллельностью, то оно в большинстве распределённых систем базируется на механизме блокирования, точно так, как и в нераспределённых системах.
9. Аппаратная независимость. Желательно иметь возможность запускать одну и ту же СУБД на различных аппаратных платформах и, более того, добиться, чтобы различные машины участвовали в работе распределённой системы как равноправные партнёры.
10. Независимость от операционной системы. Возможность функционирования СУБД под различными операционными системами.
11. Независимость от сети. Возможность поддерживать много принципиально различных узлов, отличающихся оборудованием и операционными системами, а также ряд типов различных коммуникационных сетей.
12. Независимость от типа СУБД. Необходимо, чтобы экземпляры СУБД на различных узлах все вместе поддерживали один и тот же интерфейс, и совсем необязательно, чтобы это были копии одной и той же версии СУБД.

Локальная автономия 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; управлен ие ею выполняется локально и независимо от других узлов системы.

Независимость от центрального узла означает, что 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.

Непрерывные операции можно трактовать как возможность непрерывного доступа к данным (известное 24 часа в течение суток, 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.

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

Прозрачная фрагментация 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 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.

Прозрачность тиражирования (асинхронного в общем случае процесса переноса изменений объектов исходной базы данных в базы, расположенные на других узлах распределенной системы) означает, что тиражирование возможно и достигается внутрисистемными средствами.

Обработка распределенных запросов DDB трактуется как возможность выполнения операций выборки над распределенной базой данных, сформулированных в рамках обычного запроса на языке SQL. To есть операцию выборки из распределенной базы данных можно сформулировать с помощью тех же языковых средств, что и операцию над локальной базой данных. Оптимизатор распределенных запросов учитывает такие параметры, как 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 алгоритмов работы оптимизатора распределенных запросов впрямую зависит скорость работы базы данных с такими запросами.

Обработка распределенных транзакций 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 глобальной транзакции.

Независимость от оборудования означает, что в качестве узлов распределенной системы могут выступать компьютеры любых моделей и производителей — от мэйнфреймов до персоналок.

Независимость от операционных систем как качество вытекает из предыдущего и означает многообразие операционных систем, управляющих узлами распределенной системы.

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

#### распределённая база данные компьютерный

#### 3 Проблемы распределенных баз данных

Исходя из определения Дэйта, распределенную базу данных в общем случае можно рассматривать как слабосвязанную сетевую структуру, узлы которой представляют собой локальные базы данных. Локальные базы данных автономны, независимы и самоопределены; доступ к ним обеспечивается 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.

Рассмотрим теперь проблемы реальных распределенных баз данных [5]. Проблемы централизованных СУБД существуют и здесь, однако децентрализация добавляет новые:

а) Какова общая модель данных распределенной системы? Мы должны иметь единую концептуальную схему всей сети. Это обеспечит логическую прозрачность данных для пользователя, в результате чего он сможет формировать запрос ко всей базе, находясь за отдельным терминалом (т. е. как бы работая с централизованной базой данных).

б) Необходима схема, определяющая местонахождение данных в сети. Это обеспечит прозрачность размещения данных, благодаря которой пользователь может не указывать, куда переслать запрос, чтобы получить требуемые данные.

в) Распределенные базы данных могут быть однородными или неоднородными по аппаратным и программным средствам. Проблему неоднородности сравнительно легко решить, если распределенная база является неоднородной по аппаратным средствам, но однородной по программным средствам (одинаковые СУБД в узлах). Если же в узлах распределенной системы используются разные СУБД, необходимы средства преобразования структур данных и языков. Это должно обеспечить прозрачность преобразования в узлах распределенной базы данных.

г) Управление словарями. Для обеспечения всех видов прозрачности в распределенной базе данных нужны программы, управляющие многочисленными справочниками или словарями.

д) Методы выполнения запросов в распределенной базе данных отличаются от аналогичных методов централизованных СУБД, так как отдельные части запроса нужно выполнять в месторасположении соответствующих данных и передавать частичные результаты на другие узлы; при этом должна быть обеспечена координация всех процессов.

е) В распределенной базе данных нужен сложный механизм управления одновременной обработкой, который, в частности, должен обеспечивать синхронизацию при обновлениях информации, это гарантирует непротиворечивость данных.

ж) Развитая методология распределения и размещения данных, включая разбиение, является одним из основных требований к распределенной базе данных.

База данных физически распределяется по узлам компьютерной информационной системы при помощи фрагментации и репликации (тиражирования) данных.

#### 4 Особенности распределенных баз данных

В сегодняшнем быстро меняющемся компьютерном мире сосуществуют по крайней мере три основные идеологии: клиент - сервер, Web и распределенные объекты (DCOM, CORBA). Внутри каждого направления также существует большое количество решений и стандартов от разных производителей. Сегодняшняя ситуация вызывает очень большую озабоченность независимых разработчиков и потребителей: Какую технологию выбрать и что будет со мной и моим бизнесом, если я приму неправильное решение? При этом очевидно, что цена ошибки будет весьма высока, кроме того большие средства уже вложены в разработку и эксплуатацию уже существующих систем.

#### Клиент-сервер

Термин "клиент-сервер" означает такую архитектуру программного комплекса, в которой его функциональные части взаимодействуют по схеме "запрос-ответ". Если рассмотреть две взаимодействующие части этого комплекса, то одна из них (клиент) выполняет активную функцию, т. е. инициирует запросы, а другая (сервер) пассивно на них отвечает. По мере развития системы роли могут меняться, например некоторый программный блок будет одновременно выполнять функции сервера по отношению к одному блоку и клиента по отношению к другому [6].

Любая информационная система должна иметь минимум три основные функциональные части - модули хранения данных, модули обработки и модули интерфейса с пользователем. Каждая из этих частей может быть реализована независимо от двух других. Например, не изменяя программ, используемых для хранения и обработки данных, можно изменить интерфейс с пользователем таким образом, что одни и те же данные будут отображаться в виде таблиц, графиков или гистограмм. Не меняя программ представления данных и их хранения, можно изменить программы обработки, например изменив алгоритм полнотекстового поиска. И наконец, не меняя программ представления и обработки данных, можно изменить программное обеспечение для хранения данных, перейдя, например, на другую файловую систему.

В классической архитектуре клиент-сервер приходится распределять три основные части приложения по двум физическим модулям. Обычно ПО хранения данных располагается на сервере (например, сервере базы данных), интерфейс с пользователем - на стороне клиента, а вот обработку данных приходится распределять между клиентской и серверной частями. В этом-то и заключается основной недостаток двухуровневой архитектуры, из которого следуют несколько неприятных особенностей, сильно усложняющих разработку клиент-серверных систем.

При разбиении алгоритмов обработки данных необходимо синхронизировать поведение обеих частей системы. Все разработчики должны иметь полную информацию о последних изменениях, внесенных в систему, и понимать эти изменения. Это создает большие сложности при разработке клиент-серверных систем, их установке и сопровождении, поскольку необходимо тратить значительные усилия на координацию действий разных групп специалистов. В действиях разработчиков часто возникают противоречия, а это тормозит развитие системы и вынуждает изменять уже готовые и проверенные элементы.

Чтобы избежать несогласованности различных элементов архитектуры, пытаются выполнять обработку данных на одной из двух физических частей - либо на стороне клиента ("толстый" клиент), либо на сервере ("тонкий" клиент, или архитектура, называемая "2,5- уровневый клиент-сервер"). Каждый подход имеет свои недостатки. В первом случае неоправданно перегружается сеть, поскольку по ней передаются необработанные, а значит, избыточные данные. Кроме того, усложняется поддержка системы и ее изменение, так как замена алгоритма вычислений или исправление ошибки требует одновременной полной замены всех интерфейсных программ, а иначе могут возникнуть ошибки или несогласованность данных. Если же вся обработка информации выполняется на сервере (когда такое вообще возможно), то возникает проблема описания встроенных процедур и их отладки. Дело в том, что язык описания встроенных процедур обычно является декларативным и, следовательно, в принципе не допускает пошаговой отладки. Кроме того, систему с обработкой информации на сервере абсолютно невозможно перенести на другую платформу, что является серьезным недостатком.

Многие средства быстрой разработки приложений (RAD), которые работают с различными базами данных, реализует первую стратегию, т. е. "толстый" клиент обеспечивает интерфейс с сервером базы данных через встроенный SQL. Такой вариант реализации системы с "толстым" клиентом, кроме перечисленных выше недостатков, обычно обеспечивает недопустимо низкий уровень безопасности. Например, в банковских системах приходится всем операционистам давать права на запись в основную таблицу учетной системы. Кроме того, данную систему почти невозможно перевести на Web-технологию, так как для доступа к серверу базы данных используется специализированное клиентское ПО.

Рассмотренные выше модели имеют следующие недостатки.

1. "Толстый" клиент:

* сложность администрирования;
* усложняется обновление ПО, поскольку его замену нужно производить одновременно по всей системе;
* усложняется распределение полномочий, так как разграничение доступа происходит не по действиям, а по таблицам;
* перегружается сеть вследствие передачи по ней необработанных данных;
* слабая защита данных, поскольку сложно правильно распределить полномочия.

2. "Толстый" сервер:

* усложняется реализация, так как языки типа PL/SQL не приспособлены для разработки подобного ПО и нет хороших средств отладки;
* производительность программ, написанных на языках типа PL/SQL, значительно ниже, чем созданных на других языках, что имеет важное значение для сложных систем;
* программы, написанные на СУБД-языках, обычно работают недостаточно надежно; ошибка в них может привести к выходу из строя всего сервера баз данных;
* получившиеся таким образом программы полностью непереносимы на другие системы и платформы.

Для решения перечисленных проблем используются многоуровневые (три и более уровней) архитектуры клиент-сервер.

Многоуровневая архитектура клиент-сервер — разновидность архитектуры клиент-сервер, в которой функция обработки данных вынесена на один или несколько отдельных серверов. Это позволяет разделить функции хранения, обработки и представления данных для более эффективного использования возможностей серверов и клиентов.

Частный случаи многоуровневой архитектуры - трёхуровневая (трехзвенная) архитектура.

Трёхзвенная архитектура предполагает наличие следующих компонентов приложения: клиентское приложение (обычно говорят «тонкий клиент» или терминал), подключенное к серверу приложений, который в свою очередь подключен к серверу базы данных.

«Тонкий клиент» или Терминал — это интерфейсный (обычно графический) компонент, который представляет собственно приложение для конечного пользователя. Первый уровень не должен иметь прямых связей с базой данных (по требованиям безопасности), быть нагруженным основной бизнес-логикой и хранить состояние приложения (по требованиям надежности). На первый уровень может быть вынесена и обычно выносится простейшая бизнес-логика: интерфейс авторизации, алгоритмы шифрования, проверка вводимых значений на допустимость и соответствие формату, несложные операции (сортировка, группировка, подсчет значений) с данными, уже загруженными на терминал.

Сервер приложений располагается на втором уровне. На втором уровне сосредоточена бо́льшая часть бизнес-логики. Вне его остаются фрагменты, экспортируемые на терминалы, а также погруженные в третий уровень хранимые процедуры и триггеры.

Сервер базы данных обеспечивает хранение данных и выносится на третий уровень. Обычно это стандартная реляционная или объектно-ориентированная СУБД. Если третий уровень представляет собой базу данных вместе с хранимыми процедурами, триггерами и схемой, описывающей приложение в терминах реляционной модели, то второй уровень строится как программный интерфейс, связывающий клиентские компоненты с прикладной логикой базы данных.

В простейшей конфигурации физически сервер приложений может быть совмещён с сервером базы данных на одном компьютере, к которому по сети подключается один или несколько терминалов.

В «правильной» (с точки зрения безопасности, надёжности, масштабирования) конфигурации сервер базы данных находится на выделенном компьютере (или кластере), к которому по сети подключены один или несколько серверов приложений, к которым, в свою очередь, по сети подключаются терминалы.

Технология клиент-сервер по праву считается одним из "китов", на которых держится современный мир компьютерных сетей. Но те задачи, для решения которых она была разработана, постепенно уходят в прошлое, и на сцену выходят новые задачи и технологии, требующие переосмысления принципов клиент-серверных систем. Одна из таких технологий - World Wide Web.

Многоуровневые клиент-серверные системы достаточно легко можно перевести на Web-технологию - для этого достаточно заменить клиентскую часть универсальным или специализированным браузером, а сервер приложений дополнить Web-сервером и небольшими программами вызова процедур сервера. Реализация этого принципа основана на использовании либо HTTP-SQL, либо API (организация динамических приложений на стороне сервера), либо Java (JDBC - организация динамических приложений на стороне клиента) интерфейсов для формирования запросов пользователя к базам данных или к другим информационным источникам на получение и обработку информации.

Следует отметить и тот факт, что в трехуровневой системе по каналу связи между сервером приложений и базой данных передается достаточно много информации. Однако это не замедляет вычислений, так как для связи указанных элементов можно использовать более скоростные линии. Это потребует минимальных затрат, поскольку оба сервера обычно находятся в одном помещении. Таким образом, увеличивается суммарная производительность системы - над одной задачей теперь работают два различных сервера, а связь между ними можно осуществлять по наиболее скоростным линиям с минимальными затратами средств. Правда, возникает проблема согласованности совместных вычислений, которую призваны решать менеджеры транзакций - новые элементы многоуровневых систем.

#### Менеджеры транзакций

Особенностью многоуровневых архитектур является использование менеджеров транзакций (МТ), которые позволяют одному серверу приложений одновременно обмениваться данными с несколькими серверами баз данных. Хотя серверы Oracle имеют механизм выполнения распределенных транзакций, но если пользователь хранит часть информации в БД Oracle, часть в БД Informix, а часть в текстовых файлах, то без менеджера транзакций не обойтись. МТ используется для управления распределенными разнородными операциями и согласования действий различных компонентов информационной системы. Следует отметить, что практически любое сложное ПО требует использования менеджера транзакций. Например, банковские системы должны осуществлять различные преобразования представлений документов, т. е. работать одновременно с данными, хранящимися как в базах данных, так и в обычных файлах, - именно эти функции и помогает выполнять МТ.

Менеджер транзакций - это программа или комплекс программ, с помощью которых можно согласовать работу различных компонентов информационной системы [7]. Логически MT делится на несколько частей:

* коммуникационный менеджер (Communication Manager) контролирует обмен сообщениями между компонентами информационной системы;
* менеджер авторизации (Authorisation Manager) обеспечивает аутентификацию пользователей и проверку их прав доступа;
* менеджер транзакций (Transaction Manager) управляет распределенными операциями;
* менеджер ведения журнальных записей (Log Manager) следит за восстановлением и откатом распределенных операций;
* менеджер блокировок (Lock Manager) обеспечивает правильный доступ к совместно используемым данным.

Обычно коммуникационный менеджер объединен с авторизационным, а менеджер транзакций работает совместно с менеджерами блокировок и системных записей. Причем такой менеджер редко входит в комплект поставки, поскольку его функции (ведение записей, распределение ресурсов и контроль операций), как правило, выполняет сама база данных (например, Oracle).

Первые менеджеры транзакций появились в начале 70-х гг. (например, CICS); с тех пор они незначительно изменились идеологически, но весьма существенно - технологически. Наибольшие идеологические изменения произошли в коммуникационном менеджере, так как в этой области появились новые объектно-ориентированные технологии (CORBA, DCOM и т.д.). Из-за бурного развития коммуникационных средств в будущем следует ожидать широкого использования различных типов менеджеров транзакций.

Таким образом, многоуровневая архитектура клиент-сервер позволяет существенно упростить распределенные вычисления, делая их не только более надежными, но и более доступными. Появление таких средств, как Java, упрощает связь сервера приложений с клиентами, а объектно-ориентированные менеджеры транзакций обеспечивают согласованную работу сервера приложений с базами данных. В результате создаются все предпосылки для создания сложных распределенных информационных систем, которые эффективно используют все преимущества современных технологий.

Одним из основных требований к распределенной базе данных остается требование наличия развитой методологии распределения и размещения данных, включая разбиение.

#### Фрагментация данных

База данных физически распределяется по узлам компьютерной информационной системы при помощи фрагментации и репликации (тиражирования) данных.

Отношения, принадлежащие реляционной базе данных, могут быть фрагментированы на горизонтальные или вертикальные разделы.

Горизонтальная фрагментация реализуется при помощи операции селекции, которая направляет каждый кортеж отношения в один из разделов, руководствуясь предикатом фрагментации. Например, для отношения Employee (Сотрудник) возможна фрагментация в соответствии с территориальным распределением рабочих мест сотрудников.

Тогда запрос "получить информацию о сотрудниках компании" может быть сформулирован так:

SELECT \* FROM employee@donetsk,

employee@kiev

На рисунке 1 изображен принцип разделения данных при горизонтальной фрагментации.

На рисунке 2 приведен пример горизонтальной фрагментации.

**Рис. 1 Горизонтальная фрагментация**

**Рис. 2 Пример горизонтальной фрагментации**

При вертикальной фрагментации отношение делится на разделы при помощи операции проекции. Например, один раздел отношения Employee может содержать поля Номер\_сотрудника (emp\_id), ФИО\_сотрудника (emp\_name), Адрес\_сотрудника (emp\_adress), а другой – поля Номер\_сотрудника (emp\_id), Оклад (salary), Руководитель (emp\_chief).

Тогда запрос "получить информацию о заработной плате сотрудников компании" будет выглядеть следующим образом:

SELECT employee.emp\_id,

emp\_name,

salary

FROM employee@donetsk,

employee@kiev

ORDER BY emp\_id

На рисунках 3 и 4 изображены сущность и пример вертикальной фрагментации.

**Рис. 3 Вертикальная фрагментация**

**Рис. 5. Пример вертикальной фрагментации**

За счет фрагментации данные приближаются к месту их наиболее интенсивного использования, что потенциально снижает затраты на пересылки; уменьшаются также размеры отношений, участвующих в пользовательских запросах. Однако практически добиться ускорения выполнения запросов, затрагивающих фрагментированные отношения, очень трудно. Основная проблема состоит в резком расширении пространства поиска вариантов выполнения запросов, с которым должен работать оптимизатор запросов.

#### Репликация данных

Второй способ распределения данных – репликация (рис.6). Репликация (или тиражирование) означает создание дубликатов данных. Репликаты – это множество различных физических копий некоторого объекта базы данных (обычно таблицы), для которых поддерживается синхронизация (идентичность) с некоторой "главной" копией.

**Рис. 6. Репликация**

Теоретически значения всех данных в тиражированных объектах должны автоматически и незамедлительно синхронизироваться друг с другом. (На практике это правило обычно несколько ослабляется.) В некоторых системах копии используются исключительно в режиме чтения и обновляются в соответствии с заданным расписанием. В других средах допускается модификация отдельных значений в копиях, и эти изменения распространяются в соответствии с процедурами планирования и координации. На рисунках 7, 8, 9 показаны различные модели тиражирования.

При репликации фрагменты данных тиражируются с учетом спроса на доступ к ним. Это полезно, если доступ к одним и тем же данным нужен из приложений, выполняющихся на разных узлах. В таком случае, с точки зрения экономии затрат, более эффективно будет поддерживать копии данных на всех узлах, чем непрерывно пересылать данные между узлами.

**Рис. 7. Одновременное обновление (с управлением параллелизмом)**

**Рис. 8 Распространенные обновления**

Основной проблемой репликации данных является то, что обновление любого логического объекта должно распространяться на все хранимые копии этого объекта. Трудности возникают из-за того, что некоторый узел, содержащий данный объект, может быть недоступен (например, из-за краха системы или данного узла) именно в момент обновления. В таком случае очевидная стратегия немедленного распространения обновлений на все копии может оказаться неприемлемой, поскольку предполагается, что обновление (а значит и исполнение транзакции) будет провалено, если одна из копий будет недоступна в текущий момент.

**Рис. 9. Запланированная синхронизация дубликатов только для чтения**

В современных СУБД функции репликации выполняет, как правило, специальный модуль – сервер тиражирования данных, называемый репликатором (так устроены СУБД CA – OpenIngress и Sybase). В Informix-OnLine Dynamic Server репликатор встроен в сервер, вOracle для использования репликации необходимо приобрести дополнительную опцию Replication Option.

Спецификация механизмов репликации зависит от используемой СУБД. Простейший вариант – использование “моментальных снимков” (snapshot).

#### Каталог распределенной системы

Важным компонентом структуры логического уровня РБД является сетевой каталог, который обеспечивает эффективное выполнение основных функций управления РБД и содержит всю информацию, необходимую для обеспечения независимости размещения, фрагментации и репликации. Существует несколько вариантов хранения системного каталога. Ниже перечислены некоторые из этих вариантов.

1.Централизированный каталог. Весь каталог храниться в одном м месте, т.е. на центральном узле.

2. Полностью реплицированный каталог. Весь каталог полностью хранится на каждом узле.

3. Секционированный каталог. На каждом узле содержится его с собственный каталог для объектов, хранимых на этом узле. Общий каталог я является объединением всех разъединенных локальных каталогов.

4. Комбинация первого и третьего вариантов. На каждом узле с содержится его собственный каталог (как в п.3), кроме того, на одном центральном узле хранится унифицированная копия всех этих локальных каталогов (как в п.1).

Для каждого подхода характерны определенные недостатки и проблемы. В первом подходе, очевидно, не достигается "независимость от центрального узла". Во втором утрачивается автономность функционирования, поскольку при обновлении каждого каталога это обновление придется распространять на каждый узел. В третьем выполнение не локальных операций становится весьма дорогостоящим (для поиска удаленного объекта потребуется в среднем осуществить доступ к половине имеющихся узлов). Четвертый подход более эффективен, чем третий (для поиска удаленного объекта потребуется осуществить доступ только к одному удаленному каталогу), но в нем снова не достигается "независимость от центрального узла".

#### 5 Internet/Intranet технологии

Использование Internet/Intranet технологий при создании информационных ресурсов и построении информационных систем различного назначения в последнее время стало доминирующим в мировом информационном пространстве по следующим причинам. Эти технологии [2]:

Позволяют организовать с достаточной простотой для пользователя системы поиска нужной информации.

Предъявляют минимальные требования как с технической стороны так и со стороны программного обеспечения к рабочему месту клиента (клиент работает со стандартным программным обеспечением и единственным требованием является поддержка работы WWW просмотрщика -- браузера одной из последних версий1).

Поддерживают распределенные системы хранения информации и множественные методы ее хранения.

Поддерживают работу с практически неограниченным объемом разноплановых данных (текст, графика, изображение, звук, видео, векторные карты и др.).

Предоставляют технологически простой способ администрирования информационных систем с одного рабочего места.

Поддерживают удаленные методы редактирования и пополнения информации.

**Рис 10. Взаимодействие с БД через Интернет**

Основой построения информационных систем с использованием Intranet технологии является организация системы доступа к информации через WWW сервис Internet. Internet технология позволяет оперативно управлять и актуализировать информацию, хранящуюся в базах данных через просмотрщик (браузер) WWW страниц (рис.10).

Основной принцип, заложенный в Intranet технологию создания информационных ресурсов и построения информационных систем, заключается в разделении вычислительных ресурсов как между многочисленными серверами, расположенными в различных концах сети, так и между серверами и клиентами (компьютер на котором работает конечный пользователь). Реализация этого принципа основана на использовании либо HTTP-SQL (формирование SQL запросов к БД с WWW сервера), либо API (организация динамических приложений на стороне сервера), либо Java (JDBC - организация динамических приложений на стороне клиента) интерфейсов для формирования запросов пользователя к базам данных или к другим информационным источникам на получение и обработку информации.

Internet технология позволяет удачно сочетать возможности гипертекстового оформления информации c использованием возможностей современных СУБД. Причем со стороны клиента полностью унифицируются запросы на поиск и представление информации, а также получение аналитических справок и данных из информационных систем.

Вместе с тем рассматриваемые технологии позволяют использовать в сетевом режиме уже имеющиеся базы данных, не затрачивая при этом средства на их унификацию и приведение к единому стандарту. Основные затраты здесь направлены только на соответствующие описания баз данных и запросов для HTTP-SQL интерфейса или для сервера обработки транзакций, при этом базы данных могут находится на различных машинах, расположенных на произвольном расстоянии друг от друга. Использование данной технологии позволяет решать весь спектр задач, присущий информационной системе, включая удаленный ввод и редактирование данных.

Математическое обеспечение для организации HTTP-SQL интерфейса является свободно распространяемым как для MS Windows NT систем, так и для некоммерческих UNIX платформ. СУБД можно использовать либо имеющиеся в наличии, либо приобретать сетевые (например, Informix, Oracle, MS SQL).

Любая информационная система, построенная на основе клиент-серверных Интернет технологий, должна содержать следующие серверные компоненты:

шлюз-сервер, управляющий правами доступа к информационной системе;

WWW-сервер;

сервер баз данных;

сервер приложений и(или) сервер обработки транзакций.

Взаимодействие WWW сервера с базами данных может быть организовано двумя способами:

через сервер транзакций (см. рис.11);

через API интерфейс WWW сервера или сервера приложений (см. рис.12,13 ).

Использование коммерческих серверов транзакций, подразумевает организацию более менее стандартного интерфейса, а использование API приложений дает полную волю разработчикам

**Рис 11. Взаимодействие с БД с использованием сервера транзакций**

Организация взаимодействия с базами данных, использованием API, возможна по одной из приведенных на рис.12,13 схеме.

На рис.3 представлена стандартная схема формирования информационной системы, основанная как на использовании активных программ на сервере и стандартных средств доступа к БД как, например, Windows-NT ODBC интерфейс доступа к БД со стороны сервера и JDBC Java интерфейс доступа к БД со стороны клиента.

**Рис 12. Использование API интерфейса WWW сервера**

Схема, изображенная на рис.4, соответствует информационной системе использующей сервер приложений.

**Рис 13. Использование сервера приложений и API интерфейса WWW сервера**

В случае размещения базы данных на разных машинах, находящихся в различных локальных сетях, необходимо строить доверительные базы с обязательным применением шлюзов для обеспечения прав доступа (см. рис.14 ).

**Рис 14. Организация доверительных БД -- работа через машину-посредник (шлюз)**

#### Заключение

В основе любых технологических потрясений лежит простой экономический расчет: выгодно - невыгодно. В основе нынешней ситуации в развитии распределенных систем также лежит экономическое обоснование - стоимость передачи данных по сети становится меньше стоимости вычислений на клиентской машине и эта тенденция имеет устойчивый характер. Взрывной рост Internet, который многие связывают с "демократическими свободами" или развитием новой технологии имеет в своей основе все тоже простое экономическое обоснование - эта технология экономически выгодна. Отсюда проистекают и те изменения в мире технологий свидетелями которых мы являемся: стремительный рост пропускной способности каналов (Internet - 2, новые более быстрые модемы, спутниковые каналы для домашнего пользователя ), присутствие в сети большинства корпораций и масс медиа, электронная коммерция и банки … На основе этих технологий выросли новые направления бизнеса, а распространенность Internet растет темпами невиданными в отрасли (быстрее телефонии и телевидения).

Однако, если присмотреться поближе к этой технологии, то в ней нет ничего революционного, за исключением того, как уже известные решения применены в новой области. Давно известны языки разметки (TeX), протоколы передачи данных (TCP) и удаленных сервисов (NSF, POP), распределенные транзакции (мониторы транзакций), платформопереносимые языки (С, Perl) и т.д.

Весь секрет новых решений в заложенной изначально совместимости, опирающейся на открытые стандарты. Именно поэтому основная битва идет вокруг стандартов, чтобы не декларировали "участники забега". Решения, основанные на стандарте, являются экономически выгодными, т.к. гарантируют возврат инвестиций чтобы не происходило на рынке.

В тоже время сама технология пока достаточно слаба, как и любая технология в начале своего пути. Требования к системным ресурсам не уменьшились. Однако, как уже было показано в основе революции лежит общая экономия средств, которая при нынешней дешевизне компьютерных ресурсов и дороговизне человеческих получается весьма значительная.

Создание реальных прикладных систем на основе Internet технологии, в свою очередь, катализировало изменения в самой технологии. Впервые ставится под вопрос необходимость священной коровы - Операционной Системы. Чрезвычайно фетишизированная усилиями Microsoft ОС, тем не менее, всего лишь служебная функция необходимая для выполнения реальных приложений (недаром Sun и Oracle заключили кросслицензионное соглашение, позволяющее встраивать функции ОС в СУБД и СУБД в ОС).

Значительно пересмотрены и другие концепции, казавшиеся незыблемыми. К примеру, технология клиент - сервер построена на обращении клиента к серверу по частному протоколу (SQL Net в случае). Находящийся на стороне сервера listener обеспечивает соединение и обработку запроса. Возникает вопрос - а почему к СУБД можно обращаться только по одному специальному протоколу? Ведь при построении приложения в Internet приходится несколько раз проводить преобразование протоколов http в SGI (Perl, сервлеты и т.п.) и затем в SQL. Когда можно просто поручить listener'у иметь возможность обрабатывать запросы по http, POP3, IMAP4, NFS и другим. Подобная концепция реализованная в Oracle8i позволяет реально превратить реляционную СУБД в хранилище информации в Internet. Подобные решения кардинальным образом переворачивают наши представления о правильно построенной информационной системе, но это неизбежная дань за участие в очередной технологической революции.

Подход корпорации Oracle основан на полном признании сложившихся стандартов и их интеграцию (тоже на основе стандартов). Для Oracle вопрос не ставится как "или" клиент-сервер "или" Web "или" распределенные объекты, решение Oracle являются объединением лучших черт технологий клиент-сервер (мощность, устойчивость, транзакции), Web (легкость распространения и управления, тонкий клиент) и распределенных объектов (компонентное программное обеспечение, интеграция решений от разных производителей и распределение задачи по всем компьютерам в сети). Такая интеграция возможна на основе стандартов CORBA 2.0 (Архитектура Диспетчера Объектных Запросов) , HTTP/HTML, IIOP (Internet Inter Object Protocol), COM/DCOM (стандарт Microsoft) и Java.

В основе подхода Oracle лежит WRB (Диспетчер Объектных Запросов Web), связанный с Web сервером и управляющий всеми объектами в сети. Такие объекты могут располагаться на сервере приложений (любом сервере в сети), в базе данных (используя все возможности Базы Данных Oracle 7 и особенно объектные расширения Oracle 8) или на клиентской части (браузере). При этом все объекты (картриджи) предоставляют Диспетчеру Объектных Запросов свой интерфейс и после этого могут вызывать друг друга, создавать новые экземпляры объектов и т.д. Управление объектами, их установка, секретность, регулировка загрузки серверов и др. возможности обеспечиваются WRB. Такая архитектура называется Архитектура Сетевых Вычислений.

Таким образом любое приложение используемое в Архитектуре Сетевых Вычислений становится независимым от языка программирования (Java, C/C++, SQL, Visual Basic и др.), независимым от типа архитектуры (клиент-сервер, web) и может быть легко интегрировано с любым другим приложением (картриджем), даже разработанным другим производителем.

Подобный подход позволяет разрабатывать любые приложения для Web. При этом страницы HTML не существуют до появления запроса пользователя и генерятся в виде последовательности команд HTML по запросу пользователя и содержат только ту информацию, которая запрашивалась в запросе. Дальнейшим расширением "виртуальных" HTML страниц по запросу является использование Java applets передаваемых на браузер, создающие интерфейс аналогичный интерфейсу экранных клиент-сервер с аналогичной функциональностью и поддержкой транзакций и секретности (SSL 3.0).

Имея в своем арсенале Web сервер с функциональностью недостижимой при других подходах Oracle также предлагает средства разработки приложений уровня предприятия. Это Designer/2000 - лучшее на сегодня CASE решение, которое теперь имеет генератор не только для клиент-сервер, но и для Web, Developer/2000 - мощное средство разработки приложений, устанавливаемое как картридж на сервере приложений (под управлением WRB) и автоматически генерящее полный набор Java апплетов для работы на браузере, Sedona - мощное объектно-ориентированное средство для создания и управления распределенными объектами.

Система, построенная по технологии распределенных объектов, состоит из набора компонент (объектов), взаимодействующих друг с другом. При этом объекты, как правило, разбросаны по сети и выполняются отдельно друг от друга.

**Рисунок 1: Модель распределенных объектов**

DCOM и CORBA основываются на коммуникации типа клиент-сервер. Запрашивая сервис, клиент вызывает метод, реализуемый удаленным объектом, действующим в роли сервера. Сервис, предоставляемый объектом, инкапсулируется с помощью интерфейса, определенного на языке IDL. Именно собственный язык IDL является одной из изюминок CORBA. Вообще, существуют три различных языка описаний под одним и тем же названием: OMG IDL (очевидно, используется в CORBA), Microsoft IDL (разработан для технологии DCOM, но в силу двоичного представления объектов не играет в этой технологии ключевой роли) и OSF IDL. Однако, по сравнению с DCOM, CORBA имеет ряд существенных отличий.

Технология CORBA изначально проектировалась для создания распределенных систем. В силу этого сервер объектов и клиентские программы, в отличие от COM/DCOM, в технологии CORBA, как правило, располагаются на разных машинах. Взаимодействие между клиентом и сервером происходит следующим образом. В процессе клиента имеется объект-посредник, именуемый stub (или Client-Side Stab). Он является полномочным представителем сервера и исполняет функции, во многом сходные с функциями объекта Proxy в технологии DCOM. Именно к stub при помощи интерфейса объекта обращается программа-клиент так, как будто stub и являет собой объект. Далее stub перенаправляет запрос клиента к особому объекту, который действует также на машине клиента. Этот объект называется ORB (Object Required Broker, брокер объектных запросов). Получив запрос, ORB формирует широковещательное сообщение во внешнюю сеть. На это сообщение откликается один из объектов Smart Agent, который функционирует на одном из компьютеров сетевого окружения (локальная сеть или Интернет). Smart Agent знает, где расположены соответствующие серверы объектов (фактически это как бы виртуальный сетевой каталог, где зарегистрированы некоторые серверы), и перенаправляет запрос на нужный сервер. На сервере пакет запроса принимает еще один объект ORB, который дешифрует запрос и пересылает его следующему объекту — BOA (Basic Object Adapter, базовый адаптер объектов). Роль объекта BOA заключается в фильтрации, кэшировании запросов и, соответственно, разграничении доступа к объекту сервера. Если запрос пропущен BOA, то он попадает в объект сервера skeleton. При этом в адресном пространстве сервера создается требуемый объект, skeleton помещает аргументы вызова в стек объекта и реализует собственно вызов. Используя объект BOA, skeleton также регистрирует созданный серверный CORBA-объект с помощью Smart Agent, а также сообщает о доступности, факте создания и о готовности объекта принимать запросы клиента. Далее следует обратная связь по описанной цепочке объектов

Какой выход из этой ситуации? Каждая компания предлагает свое частное решение уверяя, что оно наилучшее. К счастью это уже не первый случай революционной ситуации в компьютерной индустрии и мы можем учесть уроки предыдущих кризисов. Опыт показывает, что **выигрывают те, кто выбирает общепризнанные стандарты**.

#### Литература

1. Е.Н. Коровин. Методология прогнозирования и оптимального управления территориально распределенными социально-экономическими системами на основе трансформации информации и многовариантного моделирования : Дис. ... д-ра техн. наук : 05.13.01, 05.13.10 Воронеж, 2005 356 с. РГБ ОД, 71:06-5/194

2. Ю.И. Шокин, А.М. Федотов  Информационные технологии. Internet // Вычислительные технологии Том 2, N 3, 1997.

3 M. Tamer Ozsu, Patrick Valduriez. Distributed and parallel database systems. //Открытые системы. # 4/1996.

4. Date C.J. 1987. What is distributed database? InfoDB, 2:7

5. Г.М. Ладыженский. Технология "клиент-сервер" и мониторы транзакций. //Открытые информационные системы

6. В.И. Коржов, Многоуровневые системы клиент-сервер // Сети, #06/1997

7. В.А. Гладцын, К.В. Кринкин, В.В. Яновский. Сервис-ориентированная архитектура, стандарты, алгоритмы, протоколы. —СПб.: Издательство СПбГЭТУ "ЛЭТИ", 2006;