Содержание:

ВВЕДЕНИЕ

Область применения информационных систем постоянно расширяется, а сами они становятся все более и более сложными. Некоторые системы вырастают и усложняются настолько, что приобретают глобальный характер, и от их правильного и надежного функционирования начинает зависеть деятельность десятков или даже сотен тысяч людей. В силу своей "глобальности" (нужно обеспечить доступ к системе из территориально разнесенных между собой точек), а также в силу ряда других причин такие системы часто имеют очень сложную архитектуру, предполагающую их функционирование в виде набора компонентов, каждый из которых выполняется на отдельном узле. Поскольку число таких систем постоянно возрастает, требования, предъявляемые к ним, достаточно серьезны. Сложность проектирования и разработки таких систем высока, а методы и средства, применяемые при реализации таких проектов, отличны от принятых при разработке "монолитных" систем.

Не следует думать, что распределенные системы - изобретение последних лет. Два-три десятилетия назад при построении информационных систем популярной была модель "хост-компьютер + терминалы", реализованная на базе мэйнфреймов (например, IBM-360/370 или их отечественных аналогов - компьютеров серии ЕС ЭВМ), либо на базе так называемых мини-ЭВМ (например, PDP-11, также имевших отечественный аналог - СМ-4). Характерной особенностью такой системы была полная "неинтеллектуальность" терминалов, используемых в качестве рабочих мест - их работой управлял все тот же хост-компьютер. Этот подход обладал несомненными по тем временам достоинствами. Во-первых, пользователи такой системы могли совместно использовать различные ресурсы хост- компьютера (оперативную память, процессор) и довольно дорогие для тех времен периферийные устройства (принтеры, графопостроители, устройства

ввода с магнитных лент и гибких дисков, дисковые накопители). Задействованное программное обеспечение в таком случае имело дело только с "локальными" ресурсами - с локальной файловой системой, локальной оперативной памятью и т.д.

Начавшийся бурный рост индустрии персональных компьютеров поначалу мало что изменил в идеологии построения программных систем - по-прежнему в большинстве своем программы имели дело с локальными ресурсами. Правда, часть этих ресурсов была уже "псевдолокальной", например, файлы на сетевом диске. Однако по-прежнему файл обрабатывался непосредственно самим узлом, при этом файл сначала передавался по сети (уже на этом этапе развития возникли сложности - проблемы блокировки ресурсов и предупреждения тупиков, проблемы поддержки логической целостности для вносимых изменений и т.д.). В какой-то момент стало очевидно, что традиционные подходы не работают. При увеличении объема перерабатываемых данных, а также по мере возрастания их стоимости стало очевидно, что доверять их обработку клиентским машинам нельзя. Любая ошибка на них (а чем больше клиентов, тем больше вероятность ошибки) приводит либо к потере данных, либо к их блокировкам в процессе работы, а, стало быть, к снижению общей производительности системы.

Следующим ключевым шагом стало повсеместное распространение идеологии клиент-серверной обработки. Это были "двухролевые" системы: клиент нес ответственность за отображение пользовательского интерфейса и выполнение кода приложения, а роль сервера обычно поручалась СУБД. В применении к примеру с файлом переход к клиент-серверной архитектуре может быть проиллюстрирован следующим образом: вместо того, чтобы читать файл целиком и обрабатывать его, машина-клиент передает машине- серверу запрос, в котором указывает, каким образом файл должен быть обработан. Сервер запрос клиента обрабатывает и возвращает ему результат.

Повсеместный переход на технологию "клиент-сервер" помог решить много старых проблем, но при этом создал много новых. Одной из основных трудностей было и остается определение границы между функционалом клиента и сервера. Часто решение о переносе части задач на сервер пагубно сказывается на общей производительности системы, и наоборот, перенос части нагрузки на клиента может привести к потере централизации.

По мере роста популярности систем "клиент-сервер" набирала силу и технология объектно-ориентированного программирования, которая предлагала перейти к системной архитектуре с тремя слоями: слой представления отводится пользовательскому интерфейсу, слой предметной области предназначен для описания основных функций приложения, необходимых для достижения поставленной перед ним цели, а третий слой представляет источник данных. С появлением Web[1] всем внезапно захотелось иметь системы "клиент- сервер", где

в роли клиента выступал бы Web-браузер. Появившиеся инструментальные средства конструирования Web-страниц были в меньшей степени связаны с SQL[2] и потому более подходили для реализации третьего уровня.

В настоящее время можно считать, что бум технологий, связанных с клиент-серверной архитектурой, все еще продолжается - большинство работающих в настоящее время информационных систем выполнено в этой технологии. Однако актуальными являются направления, связанные с развитием этой идеи - так называемые трехслойные и многослойные, а также децентрализованные приложения.

Опыт последних лет разработки программного обеспечения (ПО) показывает, что архитектура информационной системы должна выбираться с учетом нужд бизнеса, а не личных пристрастий разработчиков. Не секрет, что правильная и четкая организация информационных бизнес- решений является слагающим фактором успеха любой компании. Особенно важным этот фактор является для предприятий среднего и малого бизнеса, которым необходима система, которая способна предоставить весь объем бизнес-логики для решения задач компании. В то же время, такие системы для компаний со средним и малым масштабом сетей часто попадают под критерий ―цена - качество‖, то есть должны обладать максимальной производительностью и надежностью при доступной цене.

Первоначально системы такого уровня базировались на классической двухуровневой клиент-серверной архитектуре (Two-tier architecture).

ГЛАВА 1. КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА

1.1. Основные понятия

Архитектура информационной системы[3] - концепция, определяющая модель, структуру, выполняемые функции и взаимосвязь компонентов информационной системы.

Клиент-сервер (*Client-server*)[4] **—** вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг (сервисов), называемыми серверами, и заказчиками услуг, называемыми клиентами.

Сервер — это программа, представляющая какие-то услуги другим программам и обслуживающая запросы клиентов на получение ресурсов определенного вида.

Клиент — это программа, использующая услугу, представляемую программой сервера. Часто люди клиентом или сервером просто называют компьютер, на котором работает какая-либо из этих программ. Основным недостатком персональных компьютеров является их невысокая вычислительная мощность и надежность, а также необходимость в приобретении дополнительных аппаратных средств для устранения изолированности отдельных персональных компьютеров друг от друга.

Как правило, пользователям компьютеров нужны и высокая вычислительная мощность и прекрасные свойства персональных компьютеров. Поэтому там, где для выполнения сложных вычислений используются мощные изолированные центральные компьютеры с терминалами, их пользователям периодически приходится ходить на персональные компьютеры для редактирования текстов или выполнения задач, использующих электронные таблицы.

Это заставляет пользователей освоить 2 различные операционные системы (на больших машинах обычно установлены ОС MVS, VMS, VM, UNIX, а на персональных - MS DOS/MS Windows, OS/2 или Mac) и не решает задачи совместного использования данных[5].

В результате опроса представителей 300 крупнейших фирм США, использующих персональные компьютеры, выяснилось, что для 81% опрошенных необходим доступ к данным более чем одного компьютера. Чтобы решить эту задачу, персональные компьютеры начали объединять в локальные сети и устанавливать на них специальные операционные системы, например, NetWare фирмы Novell, для совместного использования компьютерами сети файлов, размещенных в различных узлах сети. Эта технология называется файл-сервер.

Однако файл-серверы[6] имеют ряд недостатков. Они не позволяют в полной мере обеспечить конфиденциальность доступа и целостность данных. По сети файлы передаются целиком, независимо от того, какая часть содержащихся в них данных нужна пользователю. Это сильно перегружает сеть и уменьшает быстродействие системы. Невысока и надежность системы на основе файл-серверов. Сбой на одной из рабочих станций в момент записи файла приводит к потере или искажению данных.

Для обеспечения непротиворечивости данных приходится блокировать файлы, что также приводит к замедлению работы. Естественным желанием специалистов в области вычислительной техники было совместить преимущества персональных компьютеров и мощных центральных компьютеров.

Первым шагом в этом направлении явилось использование персональных компьютеров в качестве интеллектуальных терминалов.

При таком подходе в персональном компьютере, соединенном с центральным компьютером, запускается специальное программное обеспечение, позволяющее этому персональному компьютеру работать в режиме эмуляции терминала.

При этом мы получаем архитектуру, реализующую все достоинства архитектуры с мощным центральным компьютером, но, кроме того, персональный компьютер может использоваться и самостоятельно, по своему прямому назначению. Теперь нет необходимости иметь на столе 2 дисплея, однако большинство недостатков, присущих архитектуре с центральным компьютером, все еще сохраняется. Кроме того, хотя персональные компьютеры, имеющие дисплеи с картой VGA, позволяют работать с графикой, однако использовать их в качестве графического терминала большой центральной машины неудобно. Задача выполняется в центральном компьютере и по проводам передаются графические образы экрана. Эти образы довольно велики и скорость смены изображения на экране может быть очень низкой.

Следующим шагом в решении описанной выше проблемы явилось использование архитектуры клиент-сервер. В такой архитектуре все компьютеры сети разделены на 2 группы: клиенты и серверы. Компьютер- сервер - это мощный компьютер с большой оперативной памятью и большим количеством дискового пространства. На нем хранится база данных и выполняется сложная обработка, требующая больших вычислительных ресурсов. На компьютерах-клиентах выполняются первичная обработка данных при вводе, форматирование данных, а также окончательная (финишная) обработка данных, извлеченных с сервера.

В качестве компьютеров-клиентов обычно используются персональные. компьютеры типа IBM PC или Macintosh. Преимущества архитектуры клиент-сервер очевидны. Каждый тип компьютера используется по своему назначению, а следовательно, обеспечивается более полное использование возможностей компьютеров.

На компьютерах-клиентах работают знакомые пользователям PC пакеты, позволяющие предоставлять результаты работы всей системы в удобном для анализа и принятия решений виде. На этих компьютерах легко можно реализовать дружественный пользовательский интерфейс приложения, использующий графику, цвет, звук, работу с окнами и мышью и т.д.

Компьютер-клиент позволяет быстро выполнять ввод и первичный контроль данных. Для финишной обработки данных могут использоваться те редакторы или пакеты электронных таблиц, которые пользователь считает наиболее удобными. В качестве компьютеров- клиентов могут одновременно использоваться компьютеры разных типов с различными операционными системами.

Архитектура клиент-сервер[7] позволяет реализовать распределенную обработку, поскольку часть работы (интерфейс с пользователем, финишная обработка) выполняется на компьютере-клиенте, а часть - на компьютере- сервере. Это позволяет снизить загрузку сервера и оптимизировать его работу, а также увеличить число клиентов, одновременно работающих с сервером.

Наиболее часто архитектура клиент-сервер применяется для приложений, созданных с использованием систем управления базами данных (СУБД)[8].

Дальнейшим развитием архитектуры клиент-сервер явилось использование в сети не одного, а нескольких серверов баз данных. Это позволило перейти от работы с локальной БД к работе с распределенной БД. Причем работа с распределенной базе данных (БД) "прозрачна" для пользователя, т.е. он работает с ней так же, как с локальной БД, не задумываясь о том, на каком сервере лежат его данные. Пользователь обращается к одному из серверов, тот, не найдя у себя нужных данных, автоматически обращается к другим серверам.

Многосерверная архитектура сегодня представляется очень перспективной.

Она позволяет заменить одну мощную центральную машину на несколько менее мощных и, следовательно, более дешевых, и еще больше распараллелить обработку данных.

Кроме того, такая архитектура повышает надежность системы, поскольку при выходе из строя одного из серверов все приложения, работающие с данными других серверов, могут продолжать работу. При выходе из строя части локальной или глобальной сети система может попытаться найти альтернативный путь к нужному серверу (по другим ветвям сети). Кроме того, на локальных серверах

могут храниться данные, наиболее часто используемые в данном узле, что позволяет свести к минимуму передачу данных по сети от сервера к серверу.

1.2. Клиент-серверная архитектура применительно к бд

Вообще говоря, клиент-серверная система характеризуется наличием двух взаимодействующих самостоятельных процессов - клиента и сервера, которые, в общем случае, могут выполняться на разных компьютерах, обмениваясь данными по сети. По такой схеме могут быть построены системы обработки данных на основе СУБД, почтовые и другие системы. Файл-серверная система тоже использует технологию клиент-сервер, однако с точки зрения архитектуры прикладных программ важным является то, какого рода ресурсы предоставляет клиентам сервер. В файл-серверной системе[9] данные хранятся на файловом сервере (например, Novell NetWare или Windows NT Server), а их обработка осуществляется на рабочих станциях, на которых, как правило, функционирует одна из, так называемых, "настольных СУБД" - Access, FoxPro, Paradox и т.п.

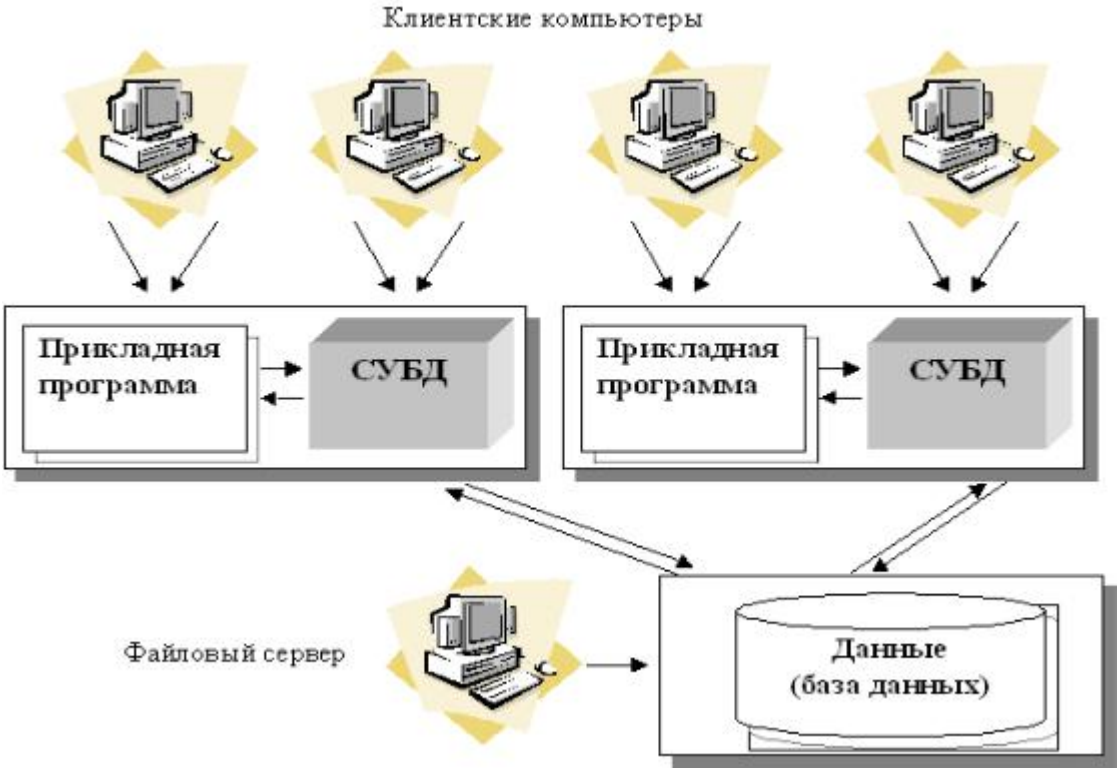

Рис. 1. Архитектура файл-сервер

Приложение на рабочей станции "отвечает за все" - за формирование пользовательского интерфейса, логическую обработку данных и за непосредственное манипулирование данными. Файловый сервер предоставляет услуги только самого низкого уровня - открытие, закрытие и модификацию

файлов, именно файлов, а не базы данных. База данных существует только в "мозгу" рабочей станции.

Таким образом, непосредственным манипулированием данными занимается несколько независимых и несогласованных между собой процессов. Кроме того, для осуществления любой обработки (поиск, модификация, суммирование и т.п.) все данные необходимо передать по сети с сервера на рабочую станцию.

Понятие архитектуры клиент-сервер в системах управления предприятием связано с делением любой прикладной программы на три основных компонента или слоя. Этими тремя компонентами являются: компонент представления (визуализации) данных; компонент прикладной логики; компонент управления базой данных.

Действительно, любая программа, компьютеризирующая выполнение той или иной прикладной задачи, должна обмениваться информацией с пользователем, осуществлять собственно обработку этой информации в рамках автоматизации того или иного бизнес-процесса, и, наконец, хранить данные, используемые в программе, на том или ином постоянном носителе.

Критерием, позволяющим отнести прикладную программу к архитектуре клиент-сервер, является то, что хотя бы один из трех ее компонентов полностью выполняется на другом компьютере, и взаимодействие между компонентами на разных компьютерах осуществляется через ту или иную сетевую среду посредством передачи запросов на получение того или иного ресурса.

Поскольку архитектура клиент-сервер является частным случаем технологии клиент-сервер, в ней обязательно есть клиент и сервер. Соответственно, выделяют клиентскую и серверную стороны приложения.

Клиентская сторона приложения функционирует на рабочем месте пользователя, в роли которого в подавляющем числе случаев выступает персональный компьютер. Серверная сторона функционирует на специализированном комплексе, включающем в себя мощные аппаратные средства, требуемый набор стандартного программного обеспечения, систему управления базами данных и собственно структуры данных.

Рис. 2. Классическое представление архитектуры "клиент-сервер"

1.3. Двухуровневая клиент-серверная архитектура

Эта архитектура получила распространение с начала 1990-х годов на фоне роста рынка персональных компьютеров и снижения спроса на мэйнфреймы.

Первоначально вышеописанные системы базировались на классической двухуровневой клиент-серверной архитектуре (Two-tier architecture)[10]. Под клиент-серверным приложением в этом случае понимается информационная система, основанная на использовании серверов баз данных.

Компанией Gartner Group, специализирующейся в области исследования информационных технологий, была предложена следующая классификация двухзвенных моделей взаимодействия клиент-сервер(двухзвенными эти модели называются потому, что три компонента приложения различным образом распределяются между двумя узлами):

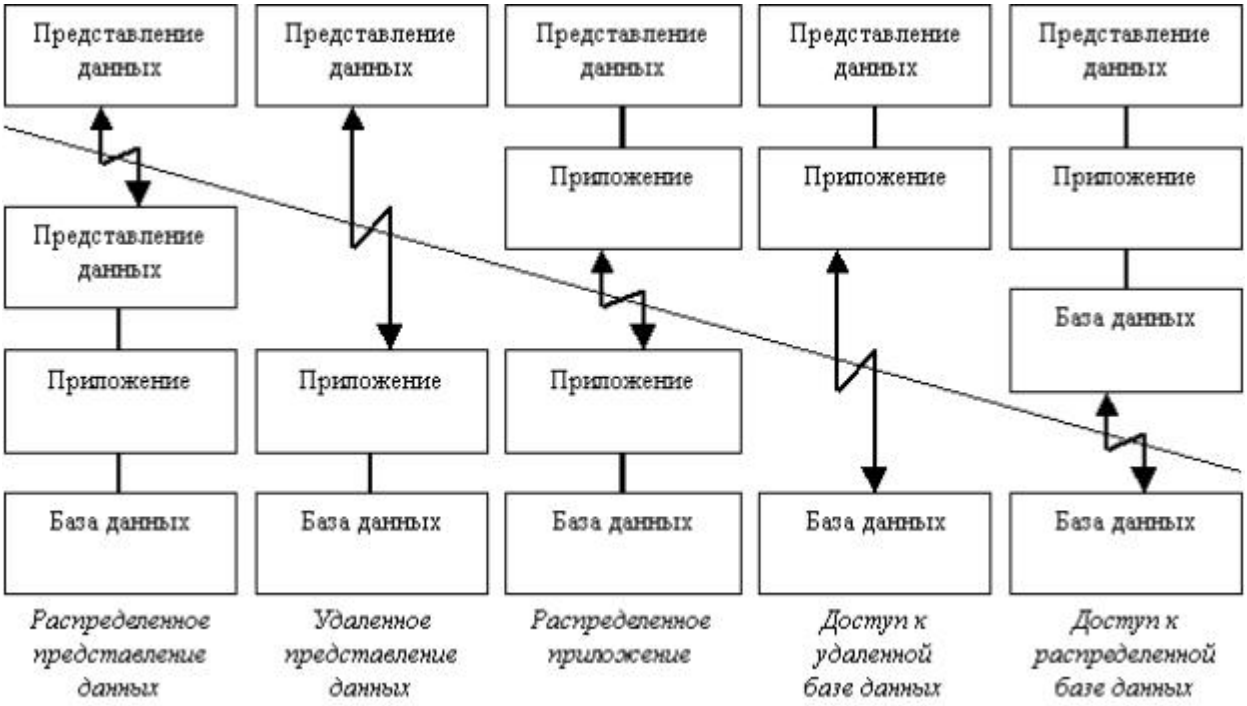

Рис. 3. Классификация двухзвенных моделей взаимодействия клиентсервер

Исторически первой появилась модель распределенного представления данных, которая реализовывалась на универсальной ЭВМ с подключенными к ней неинтеллектуальными терминалами. Управление данными и взаимодействие с пользователем при этом объединялись в одной программе, на терминал передавалась только "картинка", сформированная на центральном компьютере.

Затем, с появлением ПК и локальных сетей, были реализованы модели доступа к удаленной базе данных. Некоторое время базовой для сетей ПК была архитектура файлового сервера. При этом один из компьютеров является файловым сервером, на клиентах выполняются приложения, в которых совмещены компонент представления и прикладной компонент (СУБД и прикладная программа). Протокол обмена при этом представляет набор низкоуровневых вызовов операций файловой системы. Такая архитектура, реализуемая, как правило, с помощью персональных СУБД имеет очевидные недостатки - высокий сетевой трафик и отсутствие унифицированного доступа к ресурсам.

С появлением первых специализированных серверов баз данных появилась возможность другой реализации модели доступа к удаленной базе данных. В этом случае ядро СУБД функционирует на сервере, протокол обмена обеспечивается с помощью языка SQL. Такой подход по сравнению с файловым сервером ведет к уменьшению загрузки сети и унификации интерфейса "клиент-сервер"[11]. Однако,

сетевой трафик остается достаточно высоким, кроме того, по-прежнему невозможно удовлетворительное администрирование приложений, поскольку в одной программе совмещаются различные функции.

Позже была разработана концепция активного сервера, который использовал механизм хранимых процедур. Это позволило часть прикладного компонента перенести на сервер (модель распределенного приложения).

Процедуры хранятся в словаре базы данных, разделяются между несколькими клиентами и выполняются на том же компьютере, что и SQL-сервер.

Преимущества такого подхода:

-возможно централизованное администрирование прикладных функций,

-значительно снижается сетевой трафик (т.к. передаются не SQL- запросы, а вызовы хранимых процедур).

Недостаток - ограниченность средств разработки хранимых процедур по сравнению с языками общего назначения (C и Pascal). На практике обычно используется смешанный подход:

1. простейшие прикладные функции выполняются хранимыми процедурами на сервере;

2. более сложные прикладные функции реализуются на клиенте непосредственно в прикладной программе;

На стороне клиента выполняется код приложения, в который обязательно входят компоненты, поддерживающие интерфейс с конечным пользователем, производящие отчеты, выполняющие другие специфичные для приложения функции. Клиентская часть приложения взаимодействует с клиентской частью программного обеспечения управления базами данных, которая, фактически, является индивидуальным представителем СУБД для приложения. Интерфейс[12] между клиентской частью приложения и клиентской частью сервера баз данных, как правило, основан на использовании языка SQL. Поэтому такие функции, как, например, предварительная обработка форм, предназначенных для запросов к базе данных, или формирование результирующих отчетов выполняются в коде приложения. Наконец, клиентская часть сервера баз данных, используя средства сетевого доступа, обращается к серверу баз данных, передавая ему текст оператора языка SQL.

В продуктах практически всех компаний сервер получает от клиента текст оператора на языке SQL. Сервер производит компиляцию полученного оператора. Далее (если компиляция завершилась успешно) происходит выполнение оператора.

Разработчики и пользователи информационных систем, основанных на архитектуре "клиент-сервер", часто бывают неудовлетворенны постоянно существующими сетевыми накладными расходами, которые следуют из потребности обращаться от клиента к серверу с каждым очередным запросом. На практике распространена ситуация, когда для эффективной работы отдельной клиентской составляющей информационной системы в действительности требуется только небольшая часть общей базы данных.

Это приводит к идее поддержки локального кэша общей базы данных на стороне каждого клиента.

Фактически, концепция локального кэширования базы данных является частным случаем концепции реплицированных баз данных. Как и в общем случае, для поддержки локального кэша базы данных программное обеспечение рабочих станций должно содержать компонент управления базами данных – упрощенный вариант сервера баз данных, который, например, может не обеспечивать многопользовательский режим доступа. Отдельной проблемой является обеспечение согласованности (когерентности) кэшей и общей базы данных. Здесь возможны различные решения[13] – от автоматической поддержки согласованности за счет средств базового программного обеспечения управления базами данных до полного перекладывания этой задачи на прикладной уровень.

Преимуществами данной архитектуры являются[14]:

- возможность, в большинстве случаев, распределить функции вычислительной системы между несколькими независимыми компьютерами в сети; - все данные хранятся на сервере, который, как правило, защищен гораздо лучше большинства клиентов, а также на сервере проще обеспечить контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа; поддержка многопользовательской работы; гарантия целостности данных.

Недостатками данной архитектуры являются:

- неработоспособность сервера может сделать неработоспособной всю вычислительную сеть;

- администрирование данной системы требует квалифицированного профессионала;

- высокая стоимость оборудования;

- бизнес логика приложений осталась в клиентском ПО.

При проектировании информационной системы, основанной на архитектуре "клиент-сервер", большее внимание следует обращать на грамотность общих решений. Технические средства пилотной версии могут быть минимальными (например, в качестве аппаратной основы сервера баз данных может использоваться одна из рабочих станций). После создания пилотной версии нужно провести дополнительную исследовательскую работу, чтобы выяснить узкие места системы. Только после этого необходимо принимать решение о выборе аппаратуры сервера, которая будет использоваться на практике.

Увеличение масштабов информационной системы не порождает принципиальных проблем.

Обычным решением является замена аппаратуры сервера (и, может быть, аппаратуры рабочих станций, если требуется переход к локальному кэшированию баз данных). В любом случае практически не затрагивается прикладная часть информационной системы.

Данный вид архитектуры называют еще архитектурой с "толстым"[15] клиентом. Здесь логика представления данных и бизнес-логика размещаются на клиенте, который (скажем, в случае, когда сервером является СУБД) общается с логикой хранения и накопления данных на сервере, используя язык структурированных запросов SQL. Однако необходимость установки "толстых клиентов", требующих значительного количества специальных библиотек и специальной настройки окружения, на большое число пользовательских компьютеров с различными операционными средами, как правило, вызывает массу проблем.

Как альтернатива возникла также двухзвенная архитектура "с тонким клиентом". При этом в идеале программа-клиент реализует лишь графический интерфейс пользователя (GUI) и передает/принимает запросы, а вся бизнес-логика выполняется сервером. В идеале клиентом является просто интернет-браузер,

который имеется в стандартной операционной среде любого пользовательского компьютера и не требует специальной настройки, установки специализированного ПО и т.п. К сожалению, такая схема тоже не свободна от недостатков, хотя бы уже потому, что серверу иногда приходится брать на себя несвойственные для него функции реализации бизнес логики приложения (например, серверу СУБД приходится выполнять расчеты).

1.4 Многоуровневая архитектура клиент-сервер (multitier architecture)

Multitier architecture[16] - 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.

Среди многоуровневой архитектуры клиент-сервер наиболее распространена трехуровневая архитектура (трехзвенная архитектура, threetier), предполагающая 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.

Терминал – это интерфейсный (обычно графический) компонент, который представляет первый уровень, собственно приложение для конечного пользователя. Первый уровень не должен иметь прямых связей с базой данных (по требованиям безопасности), быть нагруженным основной бизнес-логикой (по требованиям масштабируемости) 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 большая 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 совмещен с сервером базы данных на одном компьютере, к которому по сети подключается один или несколько терминалов. В"правильной" (с точки зрения безопасности, надежности, 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.

Плюсами данной архитектуры являются:

- клиентское ПО не нуждается в администрировании; масштабируемость; конфигурируемость – изолированность уровней друг от друга позволяет быстро и простыми средствами переконфигурировать систему при возникновении сбоев или при плановом обслуживании на одном из уровней; высокая безопасность; высокая надежность; низкие требования к скорости канала (сети) между терминалами и сервером приложений; низкие требования к производительности и техническим характеристикам терминалов, как следствие снижение их стоимости.

Минусами данной архитектуры являются:

- растет сложность серверной части и, как следствие, затраты на администрирование и обслуживание; более высокая сложность создания приложений; сложнее в разворачивании и администрировании; высокие требования к производительности серверов приложений и сервера базы данных, а, значит, и высокая стоимость серверного оборудования; высокие требования к скорости канала (сети) между сервером базы данных и серверами приложений.

Начало процессу развития корпоративного программного обеспечения в многозвенной архитектуре было положено еще в рамках технологии "клиент-сервер". В них наряду с клиентской частью приложения и сервером баз данных появились серверы приложений (*Application Servers*).

В идеале: программа-клиент реализует GUI, передает запросы серверу приложений и принимает от него ответ, сервер приложений реализует бизнес-логику и

обращается с запросами к серверу "третьего уровня" (например, серверу базы данных за данными), сервер третьего уровня обслуживает запросы сервера приложений.

Программа-клиент, таким образом, может быть "тонкой".

Преимущества такой архитектуры очевидны:

- изменения на каждом из звеньев можно осуществлять независимо; снижаются нагрузки на сеть, поскольку звенья не обмениваются между собой большими объемами информации; обеспечивается масштабирование и простая модернизация оборудования и программного обеспечения, поддерживающего каждое из звеньев, в том числе обновление серверного парка и терминального оборудования, СУБД и т.д.; приложения могут создаваться на стандартных языках третьего или четвертого поколения (Java, C/C++).

Следующий логический шаг - дальнейшее увеличение числа звеньев, причем возрастет не только за счет разбиения, когда "утоньшается" каждое из известных технических звеньев, но вся бизнес-модель строится как многозвенная. Современные корпоративные программные системы представляют собой, как правило, сложные системы взаимодействующих между собой на

разных уровнях компонентов, каждые из которых могут являться клиентами для одних компонентов и серверами для других. Основной проблемой систем, основанных на двухзвенной архитектуре "клиент-сервер", или тем более на многозвенной архитектуре, является то, что от них требуется мобильность в как можно более широком классе аппаратно-программных сред. Даже если ограничиться UNIX- ориентированными локальными сетями, в разных сетях применяется разная аппаратура и протоколы связи. Попытки создания систем, поддерживающих все возможные протоколы, приводит к их перегрузке сетевыми деталями в ущерб функциональности. Еще более сложный аспект этой проблемы связан с возможностью использования разных представлений данных в разных узлах неоднородной локальной сети. В разных компьютерах может существовать различная адресация, представление чисел, кодировка символов и т.д. Это особенно существенно для серверов высокого уровня: телекоммуникационных, вычислительных, баз данных.

Общим решением проблемы мобильности такого рода систем является использование технологий[17], реализующие протоколы удаленного вызова процедур (RPC - Remote Procedure Call) стандартизованным и платформо-

независимым способом. При использовании таких технологий обращение к сервису в удаленном узле выглядит как обычный вызов процедуры (методов удаленных объектов). Средства RPC, в которых, естественно, содержится вся информация о специфике аппаратуры локальной сети и сетевых протоколов, переводит вызов в последовательность сетевых взаимодействий. Тем самым, специфика сетевой среды и протоколов скрыта от прикладного программиста.

При вызове удаленной процедуры, программы RPC производят преобразование форматов данных клиента в промежуточные машинно- независимые форматы, и затем преобразование в форматы данных сервера. При передаче ответных параметров производятся обратные преобразования.

Таким образом, если система реализована на основе стандартного пакета RPC, она может быть легко перенесена в любую открытую среду.

Некоторые авторы представляют многозвенную архитектуру (трехзвенную) в виде пяти уровней .

1.Представление;

2.Уровень представления;

3.Уровень логики;

4.Уровень данных;

5.Данные.

Рис. 4. Пять уровней многозвенной архитектуры "клиент-сервер"

К представлению относится вся информация, непосредственно отображаемая пользователю: сгенерированные html-страницы[18], таблицы стилей, изображения.

Уровень представления охватывает все, что имеет отношение к общению пользователя с системой.

К главным функциям слоя представления относятся отображение информации и интерпретация вводимых пользователем команд с преобразованием их в соответствующие операции в контексте логики и данных.

Уровень логики содержит основные функции системы, предназначенные для достижения поставленной перед ним цели. К таким функциям относятся вычисления на основе вводимых и хранимых данных, проверка всех элементов данных и обработка команд, поступающих от слоя представления, а также передача информации уровню данных. Уровень доступа к данным – это подмножество функций, обеспечивающих взаимодействие со сторонними системами, которые выполняют задания в интересах приложения.

Данные системы обычно хранятся в базе данных.

1.5. Модели клиент-сервер

Существует, по меньшей мере, три модели клиент-сервер:

1. модель доступа к удаленным данным (RDA-модель);

2. модель сервера базы данных (DBS-модель);

3. модель сервера приложений (AS-модель).

Первые две модели являются двухзвенными и не могут рассматриваться в качестве базовой модели распределенной системы. Третья модель — трехзвенная. Она (как и все многозвенные модели) хороша тем, что в ней интерфейс работы с пользователем полностью независим от компонента обработки данных. Собственно, трехзвенной ее можно считать постольку, поскольку в ней явно выделены: компонент интерфейса с пользователем; программное обеспечение промежуточного слоя (middleware); компонент управления данными.

Middleware[19] — это главный компонент трехзвенных распределенных систем. Он выполняет функции управления транзакциями и коммуникациями, транспортировки запросов, управления именами и иные функции. Существует фундаментальное различие между технологией типа "сервер запросов — клиент запросов" и трехзвенными технологиями. В первом случае клиент явным образом запрашивает данные, зная структуру базы данных (имеет место так называемая "поставка данных" клиенту). Клиент передает СУБД, например, SQL-запрос, а в ответ получает данные.

Осуществляется жесткая связь типов, для реализации которой все СУБД используют закрытый SQL-канал[20]. Он строится двумя процессами: SQL/Net на компьютере-клиенте и SQL/Net на компьютере-сервере и порождается по инициативе клиента оператором connect.

Канал называется закрытым в том смысле, что невозможно, например, написать программу, которая будет шифровать SQL-запросы по специальному алгоритму или другим образом будет вмешиваться в процесс передачи данных между клиентским и серверным приложением.

В случае трехзвенной схемы клиент явно запрашивает один из сервисов (предоставляемых прикладным компонентом), например, передавая ему некоторое сообщение, и получает ответ также в виде сообщения. Клиент направляет запрос во внешнюю среду, ничего не зная о месте расположения сервиса. Имеет место так называемая "поставка функций" клиенту.

Для клиента сама база данных видна исключительно посредством набора сервисов. Более того, он вообще ничего не знает о ее существовании, т. к. все операции над базой данных выполняются внутри сервисов. Таким образом, речь идет о двух принципиально разных подходах к построению информационных систем клиент-сервер. Двухзвенная архитектура на сегодняшний день может считаться достаточно устаревшей и, в связи с развитием распределенных информационных систем, постепенно отходит на второй план. И если для быстрого создания несложных приложений с небольшим числом пользователей этот метод подходит как нельзя лучше, то при построении корпоративных распределенных информационных систем он абсолютно непригоден в силу вышеперечисленных причин.

1.6. Клиент-серверная архитектура применительно к ИС

Термин "клиент-сервер"[21] 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.

Любая информационная система должна иметь, как минимум, три основные функциональные части - модули хранения данных, их обработки и 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 клиент-серверной архитектуре три основные части приложения приходится распределять по двум физическим модулям.

Обычно, ПО хранения данных располагается на сервере (например, сервере базы данных), интерфейс с пользователем - на стороне клиента, а 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.

А именно, 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.

ГЛАВА 2. ТОЛСТЫЙ И ТОНКИЙ КЛИЕНТЫ

Как значится в словаре Free Online Dictionary of Computing, тонкий клиент - это клиентское устройство (или программа), передающее большую часть исполняемых

им функций серверу. Толстый клиент определить намного проще - это все клиенты, не являющиеся тонкими.

Тонкий клиент (thin client)[22] — терминал сети без жестких дисков, вычислительная мощность которого и объем памяти определяются задачами пользователя. Все программы и приложения, хранящиеся на сервере, становятся доступными для пользователя при включении его устройства и выполнении процедуры регистрации на сервере. Тонким клиентом называют также ПК (в том числе и мобильный) с минимизированной мощностью процессора, оперативной и внешней памятью, позволяющий пользователю осуществлять ввод и отображение данных за счет выполнения вычислений и хранения данных на более мощном ПК или сервере, с которыми он может осуществлять связь при помощи каналов средней пропускной способности. К тонкому клиенту могут подключаться внешние устройства ввода/вывода данных (сканеры, мониторы, принтеры и проекторы). Клиент называется тонким, если он не содержит вовсе или содержит лишь малую часть бизнес-логики, т. е. представляет собой исключительно презентационный слой. К толстым относятся клиенты со значительной долей бизнес-логики. Лучший пример тонкого клиента — Web-браузер, настолько универсальный, что способен подключаться к абсолютно разным прикладным программам, о которых "не знает" ничего, и, тем не менее обеспечивать приемлемый интерфейс пользователя. Вся концепция сетевого компьютера строится на идее создания дешевого небольшого устройства, на котором будет работать Web-браузер. Централизация администрирования настольных устройств.

За счет централизованного оперирования приложениями и их модификациями, выполняемыми на сервере, они становятся доступными для всех пользователей сразу, не требуется контакт с отдельными пользователями.

Рис. 5. Преимуществ тонких клиентов

Технология «тонкий клиент-сервер» базируется на трех основных составляющих:

1)стопроцентное выполнение прикладных задач на терминальном сервере;

2)многопользовательская операционная система,

3)технология распределенного отображения пользовательского интерфейса приложений.

Пользователи имеют возможность одновременно заходить в систему и выполнять приложения на сервере в разных, защищенных друг от друга сессиях сервера.

В системе с использованием тонкого клиента по сети или коммутируемой телефонной линии на сервер передаются сигналы, отражающие нажатие на ту или иную клавишу либо то или иное движение мыши.

А сервер производит соответствующие действия и формирует изменения экрана пользователя и передаѐт эти изменения тонкому клиенту. В роли клиента может выступать любой ПК, но, поскольку на нем почти не выполняются операции по обработке данных, в качестве тонких клиентов можно применять и недорогие терминалы, имеющие низкую производительность. При работе в терминальной системе все прикладные программы, данные и параметры настроек хранятся на терминальном сервере. Это даѐт много преимуществ в плане начального развѐртывания рабочих мест (нет необходимости устанавливать программное обеспечение на каждом терминале), более удобного проведения резервного копирования данных (надо копировать только содержимое сервера), восстановления сессий после сбоев (все пользовательские сессии автоматически сохраняются на сервере). При необходимости повысить вычислитель системы можно заменой всего лишь одного устройства – терминального сервера, все рабочие места автоматически переходят на более высокий уровень производительности без необходимости замены каких-либо устройств. Толстый или Rich-клиент - это приложение, обеспечивающее (в противовес тонкому клиенту) расширенную функциональность независимо от центрального сервера. Часто сервер в этом случае является лишь хранилищем данных, а вся работа по обработке и представлению этих данных переносится на машину клиента.

Достоинства:

Толстый клиент обладает широким функционалом в отличие от тонкого. Режим многопользовательской работы. Предоставляет возможность работы даже при обрывах связи с сервером. Имеет возможность подключения к банкам без использования сети Интернет. высокое быстродействие.

Недостатки:

Большой размер дистрибутива. Многое в работе клиента зависит от того, для какой платформы он разрабатывался. При работе с ним возникают проблемы с удаленным доступом к данным.

Довольно сложный процесс установки и настройки. Сложность обновления и связанная с ней неактуальность данных. Большинство современных средств быстрой разработки приложений (RAD), которые работают с различными базами данных, реализует стратегию: "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.

Итак, рассмотренные выше модели имеют следующие недостатки**.**

1. "Толстый" клиент[23]:

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 вследствие передачи по ней необработанных данных; слабая защита данных, поскольку сложно правильно распределить полномочия.

2. "Толстый" сервер[24]:

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. Рассмотрим следующие компоненты презентационная логика (Presentation Layer - PL); бизнес-логика (Business Layer -

BL); логика доступа к ресурсам (Access Layer - AL).

Таким образом, можно придти к нескольким моделям клиент-серверного взаимодействия :

"*Толстый*" *клиент*[25]. Наиболее часто встречающийся вариант реализации архитектуры клиент-сервер в уже внедренных и активно используемых системах. Такая модель подразумевает объединение в клиентском приложении как PL, так и BL. Серверная часть, при описанном подходе, представляет собой сервер баз данных 2)., реализующий AL. К описанной модели часто применяют аббревиатуру RDA - Remote Data Access.

"*Тонкий*" *клиент.* Модель , начинающая активно использоваться в корпоративной среде в связи с распространением Internet-технологий и, в первую очередь, Web-браузеров. В этом случае клиентское приложение обеспечивает реализацию PL, а сервер объединяет BL и AL.

Сервер бизнес-логики. Модель с физически выделенным в отдельное приложение блоком BL.

1) Хотя, рассматриваемые в этой части варианты разделения функциональности между клиентом и сервером являются "классическими", далее будет использоваться не только устоявшаяся традиционная, но и более новая терминология, возникшая вследствие распространения в корпоративных средах Internet/intranet-технологий и стандартов.

2) Хотя в качестве серверной части, в общем случае, выступает менеджер многопользовательского доступа к информационным ресурсам, в этой статье будет сохраняться ориентация на серверы баз данных, как окончное серверное звено.

Модели, основанные на Internet-технологиях и применяемые для построения внутрикорпоративных систем получили название intranet. Хотя intranet-системами сегодня называют все, что так или иначе использует стек протоколов TCP/IP, с ними скорее следует связать использование Web- браузеров в качестве клиентских приложений. При этом важно отметить тот факт, что браузер не обязательно является HTML-"окном", но, в не меньшей степени, представляет собой универсальную среду загрузки объектных приложений/компонент -Java или ActiveX.

Описанные три модели организации клиент-серверных систем в определенной степени являются ориентирами в задании жесткости связей между различными функциональными компонентами, чем строго описываемыми программами в реальных проектах. Жесткость связей в схеме взаимодействия компонент системы часто определяется отсутствием (или наличием) транспортного или сетевого уровня (Transport Layer - TL), обеспечивающего обмен информацией между различными компонентами.

Серверы приложений. Посмотрим на то, что же происходит в реальной жизни. С точки зрения применения описанных моделей, при проектировании прикладных систем разработчик часто сталкивается с правилом 20/80. Суть этого правила заключается в том, что 80% пользователей обращаются к 20% функциональности, заложенной в систему, но оставшиеся 20% задействуют основную бизнес-логику - 80%. В первую группу пользователей попадают операторы информационных систем (ввод и редактирование информации), а также рядовые сотрудники и менеджеры, обращающиеся к поисковым и справочным механизмам (поиск и чтение данных). Во вторую группу пользователей попадают эксперты, аналитики и менеджеры управляющего звена, которым требуются как специфические возможности отбора информации, так и развитые средства ее анализа и представления.

С точки зрения реализации моделей необходимо обеспечить прозрачность взаимодействия между различными компонентами системы, а, следовательно, обратиться к существующим стандартам такого взаимодействия.

Любая прикладная система, вне зависимости от выбранной модели взаимодействия, требует такой инструментарий, который смог бы существенно ускорить сам процесс создания системы и, одновременно с этим, обеспечить прозрачность и наращиваемость кода. На фоне разработки и внедрения систем корпоративного масштаба явно присутствует тенденция использования объектно-ориентированных компонентных средств разработки. Соответственно, полноценное применение объектов в распределенной клиент-серверной среде требует и распределенного объектно-ориентированного взаимодействия, то есть возможности обращения к удаленным объектам.

Таким образом, мы приходим к анализу существующих распределенных объектных моделей. На настоящий момент наибольшей проработанностью отличаются COM/DCOM/ ActiveX и CORBA/DCE/Java.

Если в первом случае требуемые механизмы поддержки модели являются неотъемлемой частью операционной платформы Win32 (Windows 95/NT/CE), то во втором случае предусмотрена действительная кроссплатформенность (например, везде, где есть виртуальная машина Java). Если попытаться объективно оценить (хотя любая такая попытка во многом субъективна) перспективы применения этих моделей, то для этого необходимо понять требования к операционным платформам, выдвигаемые различными функциональными компонентами системы. При построении реальных систем корпоративного масштаба уже мало обходиться их разделением на три базовых фрагмента PL, BL, AL. Так как бизнес-логика является блоком, наиболее емким и специфичным для каждого проекта, именно ее приходится разделять на более мелкие составляющие. Такими составляющими могут быть, например, функциональные компоненты обработки транзакций (Transaction Process Monitoring), обеспечения безопасности (Security) при наличии разграничения

прав доступа и выходе в Internet (Fire-wall), публикование информации в Internet (Web-access), подготовки отчетов (Reporting), отбора и анализа данных в процессе принятия решений (Decision Support), асинхронного уведомления о событиях (Event Alerts), тиражирования данных (Replication), почтового обмена (Mailing) и др.

Вследствие наличия такого огромного количества функций, закладываемых в блоки поддержки бизнес-логики, появляется понятие сервера приложений (Application Server - AS). Причем, сервер приложений не просто является неким единым универсальным средним BL-звеном между клиентской и серверной частью системы, но AS существует во множественном варианте, как частично изолированные приложения, выполняющие специальные функции, обладающие открытыми интерфейсами управления и поддерживающие стандарты объектного взаимодействия.

Проникновение информационных технологий в сферу бизнеса в качестве неотъемлемого условия успешного управления приводит к тому, что системы корпоративных масштабов требуют сочетания различных клиент- серверных моделей в зависимости от задач, решаемых на различных конкретных направлениях деятельности предприятия. Вспомнив, снова, о правиле 20/80 можно придти к выводу, что наиболее оптимальным выбором, с точки зрения управляемости и надежности системы, является сочетание различных моделей взаимодействия клиентской и серверной части. По сути, мы приходим даже не к трехуровневой, а многоуровневой (N-tier) модели, объединяющей различных по "толщине" клиентов, серверы баз данных и множество специализированных

серверов приложений, взаимодействующих на базе открытых объектных стандартов. Существенным облегчением в реализации многоуровневых гетерогенных систем является активная работа ряда производителей программного обеспечения, направленная на создание переходного ПО. В отличие от продуктов middleware, обеспечивающих верхний транспортный уровень (универсальные интерфейсы доступа к данным ODBC, JDBC, BDE;

Message Oriented Middleware - MOM; Object Request Broker - ORB;), переходное ПО отвечает за трансляцию вызовов в рамках одного стандарта обмена в вызовы другого - мосты ODBC/JDBC и BDE/ODBC, COM/CORBA, Java/ActiveX и т.п.

В общем случае, многоуровневая модель клиент-серверной системы может быть представлена, например, следующим образом:

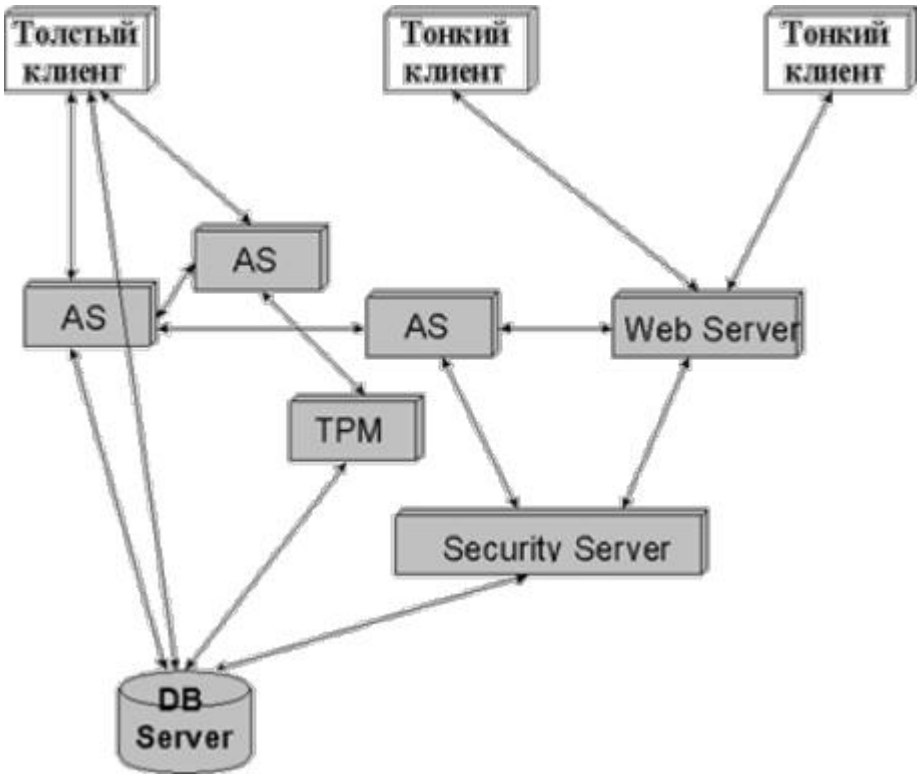

Рис. 6. Многоуровневая клиент-серверная модель

Причем, различные стандарты взаимодействия могут применяться в различных связках узлов системы, а мосты встраиваться в любой узел или выделяться в своеобразные серверы приложений, с физическим выделением в узлах сети. Двигаясь между клиентами слева-направо на диаграмме, мы можем наблюдать переход между различными моделями распределенных вычислений - через intranet к Internet.

2.1. Клиент-серверные вычисления

В 1970-х и 1980-х годов была эпоха централизованных вычислений[26] на мэйнфреймах IBM занимающих более 70% в компьютерном бизнесе мира. Бизнес транзакции, деятельности и базы данных, запросы и техническое обслуживание - все исполнялось на мэйнфреймах IBM. Этап перехода к клиент-серверным вычислениям представляет совершенно новую концепцию и технологию реорганизации всего делового мира. Вычислительные парадигмы 1990-х годов называли «волной будущего».

Машина-клиент обычно управляет интерфейсами процессов, таких как графический интерфейс (графический пользовательский интерфейс), отправкой запросов на сервер программ, проверкой данных, введенных пользователем, а также управляет местными ресурсами, а пользователь взаимодействует с такими, как монитор, клавиатура, рабочие станции, процессора и других периферийных устройств. С другой стороны, сервер выполняет запрос клиента, с помощью службы. После того как сервер получает запросы от клиентов, он выполняет поиск базы данных, обновления и управляет целостностью данных и отправляет ответы на запросы клиентов.

Цель клиент-серверных вычислений - позволить каждой сетевой рабочей станции (клиента) и принимающей (сервера) быть доступными, по мере необходимости приложения, а так же обеспечивать доступ к существующему программному обеспечение и аппаратным компонентам от различных поставщиков для совместной работы. Когда эти два условия соединены, становятся очевидными преимущества клиент-серверной архитектуры, такие как экономия средств, повышение производительности, гибкости и использования ресурсов. Клиент-серверные вычисления состоят из трех компонентов, клиентского процесса, запрашивающего обслуживание и серверного процесса предоставления запрашиваемых услуг, с Middleware между ними для их взаимодействия.

Клиент[27]

Машина клиент обычно управляет пользовательским интерфейсом частей приложения, проверкой данных, введенных пользователем, отправкой запросов на сервер программы. Кроме того, клиентский процесс также управляет местными ресурсами, что позволяет пользователю взаимодействовать с монитором, клавиатурой, рабочими станциями, процессорами и другими периферийными устройствами.

Сервер

Машина Сервер выполняет служебные запросы клиента. После того как сервер получает запросы от клиентов, он производит поиск базы данных, обновления, управляет целостностью данных и отправляет ответы на запросы клиентов. Серверных процесс может работать на другой машине в сети; тогда сервер используется как файловая система услуг и приложений сервисов. Или в некоторых случаях, другой рабочий стол машины обеспечивает применение услуг. Сервер выступает в качестве программного обеспечения двигателя, который управляет общим ресурсам, таким как базы данных, принтеры, линии связи, или процессоров высокой мощности.

Основная цель серверного процесса - выполнение фоновых задач, которые являются общими для приложений.

Простейшая форма серверов - это дисковый сервер и файл-сервер. Если клиент передает запросы на файл или группы файлов по сети на файловый сервер, эта форма обслуживания данных требует большой пропускной способности и может замедлить сеть с большим количеством пользователей.

Более продвинутые формы серверов - это серверы баз данных, сервер транзакций и серверов приложений. Middleware Middleware позволяет приложениям прозрачно контактировать с другими программами или процессами независимо от местоположения.

Ключевым элементом Middleware является NOS (Network Operating System), которая предоставляет такие услуги, как маршрутизация, распределение, обмен сообщениями и управления сервисной сети. NOS полагается на коммуникацию протоколов предоставления конкретных услуг. Прежде чем пользователь может получить доступ к услугам сети, клиент-серверный протокол требует установку физического соединения и выбор транспортных протоколов. Клиент-серверный протокол диктует, каким образом клиенты запрашивают информацию и услуги от сервера, а также как сервер отвечает на эту просьбу.

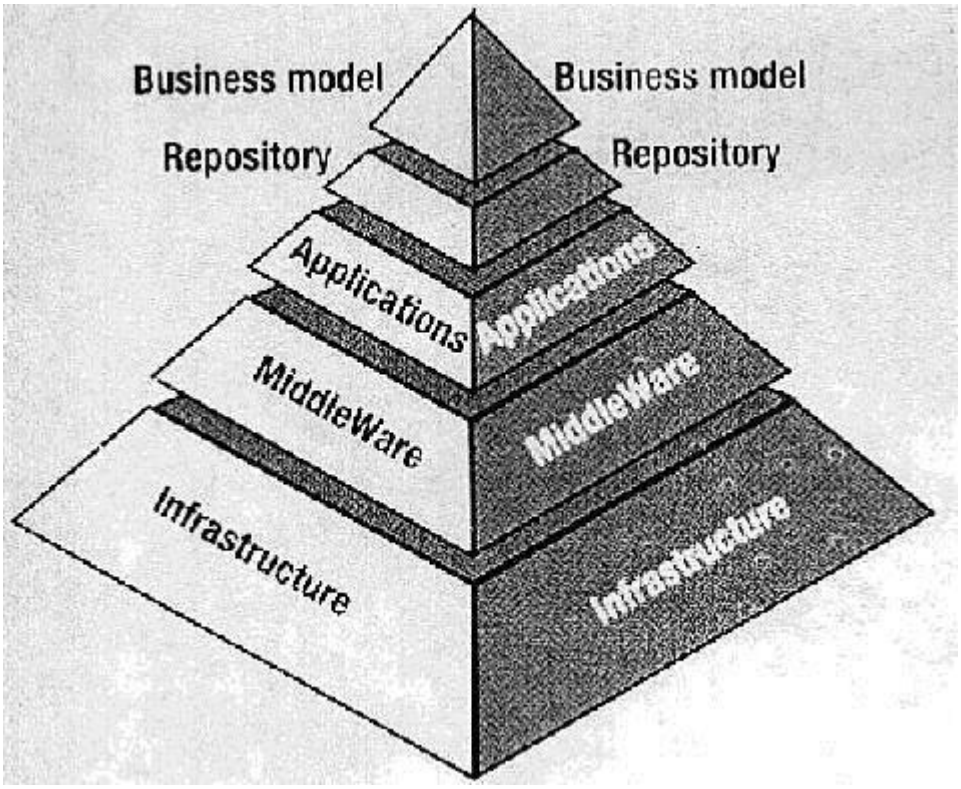

Рис. 7. Пирамида модели «клиент-сервер»

Мартин Батлер, председатель Butler Group предложил новые рамки для реализации клиент-серверной стратегии. Это пятислойная модель под названием VAL (Value Added Layers) Модель. Основная структура напоминает по форме пирамиду, со слоями Инфраструктура и Middleware в нижней части

пирамиды, а приложения, хранилища и бизнес модели на вершине. Характеристики каждого слоя[28]:

Уровень 1 - Инфраструктура слоя

Слой инфраструктура состоит из всех тех компонентов, которые являются пассивными и не выполняет бизнес-функции. Примеры относятся к этой категории компьютерных операционных систем, сетей, пользовательских интерфейсов и системы управления базой данных.

Уровень 2 - Middleware

Middleware позволяет приложениям прозрачно коммуникатировать с другими программами или процессами независимо от их местоположения.

Это средство отображения приложений используемых ими ресурсов. Middleware является ключом к интеграции гетерогенных аппаратных и программных сред, обеспечивая тот уровень интеграции, который необходим многие организациям. Типичные Middleware проявляются после сетевых соединений, соединений с базой данных и осуществляют взаимодействие между базой данных и приложениями.

Уровень 3 - Программы

Приложения являются активными компонентами, которые выполняют работы по организации, и именно сюда, многие компании инвестируют большое количество усилий, времени и денег. Приложения, которые не имеют ключевого значения. Они все чаще приобретаются как готовый пакет. Приложения же, которые являются жизненно важными для увеличения конкурентоспособности компании в отрасли, разрабатываются на месте.

Уровень 4 - Хранилище

Роль хранилища - изоляция бизнес-модели / спецификации от технологических инструментов, которые используются для ее осуществления.

Уровень 5 - Бизнес-модели

Бизнес-модель должна быть независимой, чтобы все технологии, которые используются для ее осуществления были применимы к аппаратной и программной среде в зависимости от того, что является наиболее подходящим. Это будет все больше и больше опираться на объектно- ориентированные методы, и уже существует поколение инструментов, которые поддерживают объект моделирования.

2.2. Важность сети

Для того чтобы связать клиентов и серверов вместе и в полной мере использовать ресурсы, содержащиеся в каждой машине, мы должны разработать сетевую систему, которая на это способна. Сети должны быть прозрачны для пользователей. Сети и приложения должны работать вместе так же хорошо, как если бы они работали на одном компьютере. Кроме того, сеть должна обеспечить возможности самовосстановления, чтобы можно было перенаправить сетевой трафик вокруг испорченного кабеля и быть достаточно гибкой, чтобы реагировать на бизнес-изменения в окружающей среде.

Для простоты используются локальные сети LAN[29]. Сейчас существует три различные LAN топологии: звезда, кольцо и шина и, по крайней мере, пять конкурирующих стандартов для передач, и два стандарта для информации, необходимой для управления сетью. Локальные сети стали настолько сложными, что они требуют собственную операционную систему.

Сеть продолжает быть одним из наименее изученных и наиболее важным из компонентов в информационной структуре организации.

Большинство организаций, приверженных клиент-серверной архитектуре согласны, что связь локальных сетей не место, чтобы сэкономить деньги. Мы не должны пытаться связать несовместимые локальные сети с различными платформами. Программное обеспечение, аппаратное обеспечение и операционная система, используемая в сети, должны быть тщательно протестированы.

2.3. Открытые системы и стандарты

Одна из наиболее важных особенностей клиент-серверной архитектуры - это открытость системы. Открытой системе соответствует широкий набор формальных стандартов и поддержка платформы от различных производителей. Открытые системы требует принятия стандартов в рамках всей организации. Открытую систему делает успешной открытые стандарты системы, которые должны быть приемлемыми для обеих сторон: пользователей и производителей систем и будут приняты на всех уровнях организации.

Для того чтобы все эти компоненты работали вместе как сложная система, мы должны придерживаться нескольких видов стандартов[30]. Имеются ввиду стандарты в четырех областях клиент-серверных вычислений, таких как платформы (программного и аппаратного обеспечения), сети, Middleware и приложения. Спецификации стандартов должны быть разработаны на основе консенсуса и быть общедоступными Стандарты должны быть всеобъемлющими и последовательными и определять интерфейсы, услуги и поддержку форматов для достижения совместимости. В настоящее время есть несколько консорциумов, работающих в разработке стандартов для открытых систем. -OSF (Open Software Foundation) - некоммерческий консорциум компьютерных производителей, разработчиков программного обеспечения и поставщиков чипов для разработок, основанных на стандартах программного обеспечения.

UNIX-International - консорциум производителей компьютеров, разработчиков программного обеспечения, целью которых является создание UNIX и связанных с ними стандартов разработки и лицензирования продуктов UNIX. -OMG - международные организации системы поставщиков, разработчиков программного обеспечения и пользователей, сторонники развертывания объекта управления технологиями в разработке программного обеспечения. Применяя общую основу для всех объектно-ориентированных

приложений, организации смогут управлять гетерогенными средами. CORBA (Common Object Request Broker Architecture), разработанная OMG, DEC, NCR, HP и SUN является новым механизмом, который позволяет объектам (приложениям) называть друг друга по сети.

SQL Access Group - это промышленный консорциум работает над определением и осуществления технических условий для гетерогенного доступа SQL данных с использованием принятых международных стандартов.

2.4. Модель клиент - сервер в интернете

Взаимодействие клиента и сервера в Интернете осуществляется с помощью запросов, посылаемых клиентом серверу, и ответов сервера на запрос клиента: Суть распределенных систем - связь между процессами, реализующими не только взаимодействие компьютеров, но и частей (уровней) приложений. Взаимодействие частей приложений реализуется с помощью протоколов, описывающих состав и формат данных, пересылаемых соответствующими частями клиентских и серверных приложений друг другу для решения поставленной задачи. В Интернете разбиение приложений на части осуществляется на базе стека протоколов TCP/IP.

В этой модели разработчики имеют большую свободу в определении того, какие части клиент-серверного приложения[31] будут на клиентском компьютере и какие на сервере. Логика пользовательского интерфейса существовала почти исключительно на сервере. Мы можем вновь приступить к использованию более эффективных и хорошо структурированных клиент- серверных моделей. Есть, конечно, еще технические вопросы, но мы в состоянии лучше строить истинные клиент-серверные приложений в настоящее время. Клиент-серверную модель можно разделить на три части : User Interface - Пользовательский интерфейс Business or Application Logic - Бизнес и логика приложения Data Management - Управление данными Традиционное развитие веб-приложений распространило реализации пользовательского интерфейса в сети, причем большая часть логики

интерфейса пользователя и код выполняется на сервере (тонкий клиент, толстый сервер).

Неудовлетворительное распределение обработки - с большим числом клиентов, делает все обработки на сервере неэффективно.

Высокая латентность ответа пользователя - традиционные веб-приложения не реагируют достаточно быстро.

Высокое качество взаимодействия с пользователем является очень чувствительным к задержкам, и очень быстрая реакция имеет важное значение.

Трудная модель программирования – программирование пользовательского интерфейса через клиент-сервер достаточно сложно. Если правила доступа распространяются через пользовательский код интерфейса, то при росте кода пользовательского интерфейса возникают новые векторы атаки. Сложное управления на серверах. Offline трудности. Код пользовательского интерфейса должен выполняться на клиенте и в автономном ситуациях. Снижение возможностей для взаимодействия. Когда клиент-сервер состоит из передачи внутренней части пользовательского интерфейса в браузере, бывает очень трудно понять эту связь и использовать ее для других приложений.

Необходимо решить, какой код должен работать на клиенте, а какой на сервере. Коду пользовательского интерфейса лучше работать на браузере, а бизнес-логике и управлению данными лучше работать на стороне сервера.

Хороший дизайн включает в себя создание объектов, которые инкапсулируют большую часть своего поведения и минимальной площади поверхности. Он должен быть интуитивно понятным и легко взаимодействовать с хорошо разработанным интерфейсом объекта. Кроме того, клиент-серверное взаимодействие должно быть построено на хорошо продуманном интерфейсе. Проектирование модульных удаленных интерфейсов часто называют сервис-ориентированной архитектурой (SOA).

Клиент-серверная реализация высокого качества должна иметь простой интерфейс между клиентом и сервером. На стороне клиента должен инкапсулировать презентации и код взаимодействия с пользователем. Код на стороне сервера должен инкапсулировать правила поведения и взаимодействия данных. Веб-приложения должны быть разделены на два основных элемента, пользовательский интерфейс и веб-сервис. Преимущества чистой модели клиент-

сервер:

Масштабируемость. Чем больше клиентов, которые используют приложения, тем больше клиентских машин, которые доступны, в то время как сервер остается постоянным.

Немедленный ответ пользователя. Клиентский код может немедленно реагировать на действия пользователя, а не ждать для передачи данных по сети.

Организованная модель программирования[32]. Такая модель обеспечивает более чистый подход к безопасности. Когда все запросы идут через код пользовательского интерфейса, данные могут передаваться через различные интерфейсы до проверки безопасности. Это может сделать анализ безопасности более сложным. С другой стороны, с понятным интерфейсом веб- службы есть четко определенные шлюз безопасности для работы и анализа безопасности.

Клиентская часть управления - сохранение информации о состоянии сессии на клиенте уменьшает нагрузку на сервер. Это также позволяет клиентам использовать более «спокойное» взаимодействие, которое может еще больше повысить масштабируемость и возможности кэширования.

Автономные приложения. Если большая часть кода для приложений уже построена, чтобы работать на клиенте, создание автономной версии приложения почти наверняка будет легче.

Взаимодействие. К структурированным данным с минимальными API-интерфейсами для взаимодействия намного проще подключать дополнительных потребителей и производителей и взаимодействовать с существующими системами.

2.5. Развитие

Технология перехода к клиент-серверным вычислениям проявляется, главным образом, за счет более сложной ситуации в бизнесе в последние годы, такие как глобальный маркетинг, дистанционные он-лайн продажи распределения, децентрализованной корпоративной стратегии и т.д. Все это требует быстрое реагирование, легкий доступ к информации данных, и более эффективной координации между людьми на всех уровнях как внутри, так и вне организации. Клиент-серверные вычисления позволяют решать все эти проблемы и, следовательно, являются одним из приоритетных вопросов в умах управления ИТ.

Ясно, что клиент-серверная технология приносит много преимуществ для бизнеса, а также расширение возможностей для расширения и конкурирования[33]. К сожалению, все это может быть достигнуто только за счет более высокой стоимости сооружений и более сложной совместимости систем.

Клиент-сервер не единственный способ решения бизнес-проблем, он также получил свои ограничения, один из которых, дороговизна. Но пока клиент-сервер осуществляется мудро, он может принести конкурентные преимущества. В настоящее время клиент-серверная архитектура имеет гибкую модульную архитектуру. Она может быть изменена или дополнена.

Различные подходы могут быть скомбинированы в различных комбинаторных последовательностях, удовлетворяющих практически любые вычислительным потребностям. Поскольку Интернет становится важным фактором в вычислительных средах клиент-серверных приложений, работающих через Интернет станет важным нового типа распределенных вычислений.

Интернет расширит охват и мощь клиент-серверной архитектуры. С помощью общепринятых стандартов, это позволит облегчить и расширить клиент-серверную архитектуру как внутри, так и между компаниями.

Так же из-за интернета будут происходить изменения в языках программирования к технологии распределенных объектов.

Клиент-серверная архитектура по-прежнему остается единственной и лучшей архитектурой с точки зрения использования Интернета и других новых технологий. Но, независимо от развитий других архитектурных подходов, клиент-серверная архитектура, вероятно, останется основой для большинства вычислительных событий в течение следующего десятилетия. Конечно же, с появлением большого количества людей, соединенных посредством компьютерной сети, сразу пошло бурное развитие клиент- серверных приложений. Но сервер в этом случае используется крайне экономно – это средство для хранения данных и выполнения несложных операций. Основную вычислительную роль берет на себя именно клиент, присоединенных к этой сети. Довольно удачная архитектура дает возможность использовать максимально ресурсы компьютера, богатый пользовательский интерфейс, важные данные пользователя хранятся на компьютере дома, а не сервере неизвестно какой страны.

Проблем несколько – заставить человека установить какое-либо приложение можно лишь действительно предоставив ему его как средство решения проблемы.

А как же быть тому количеству предпринимателей, которые любыми способами хотят привлечь человека своим выгодным товаром? Пока что клиент-серверная архитектура не сильно им в этом помогает. Да и приложения пишутся для определенной платформы и версии операционной системы, что напрочь лишает взаимодействия пользователей разных операционных систем.

Но прогресс не стоит на месте, и в 1989 году была предложена концепция всемирной паутины. А уже через четыре года появляется первый браузер Mosaic. Сразу же в мире приложений начинает вырисоваться два класса приложений – тонкие клиенты, примером которого может служить тот же браузер Mosaic и толстые клиенты, которые по прежнему берут на себя

значительную часть обработки информации и используют сервер для хранения и взаимодействия.

Клиентские приложения не собирается сдавать позиций, потому что все, что может сейчас браузер - это отображать информацию. Динамика отсутствует. Одно лишь значительное преимущество – сайты становятся визитной картой, имеющие свой оригинальный дизайн, расположение меню, цветовую гамму. Для клиентских приложений это не было распространено – все создавалось на основе стандартных вариантов, стандартное место расположения меню, стандартные цвета и шрифты. С одной стороны простота и функциональность, с другой стороны – красота и изящество.

Конечно же, красота и изящество не могла не найти своих сторонников. Также пользователи разных операционных систем могли спокойно просматривать сайты. Возможны некоторые несоответствия в отображении информации, но все же лучше чем ничего.

С появлением JavaScript в 1996 году html-страницы[34] получают небывалую динамику и немного начинают походить на обычные клиентские приложения. Конечно же, до полноценных клиентских приложений им, скорее всего, не дойти никогда, но страницы оживают. Живые страницы делают переворот в интернете, и идет бурное совершенствование скриптовой технологии, что приводит к рождению в 2005 году AJAX, а именно идеи асинхронного обращения к серверу для получения лишь необходимой части страницы, а не всей страницы. Инновационные решения, основанные на AJAX, типа карт Windows Live Local, приблизили веб-приложения к уровню удобства обычных клиентских программ.

Вот тут веб-страницу можно было уже спутать с обычным клиентским приложением. Но выглядеть как в клиентском приложении, и работать как клиентское приложение – это разные вещи.

Все же доступ к ресурсам ограничен, соединение между клиентом и сервером одноразовое – уже просмотренные страницы при следующем обращении необходимо загружать опять.

Но это проблемы, которые могут быть решены с появлением нового http-протокола.

В свое время, толстые клиенты, работающие на компьютере пользователя, при грандиозном развитии веб-технологий становятся в прямом смысле толстыми – они сложны для клиента при обновлении версии, они прихотливы к семействам операционных систем, плохо взаимодействуют с веб-серверами, потому что зачастую построены с использованием клиент- серверной структуры. Здесь надо было срочно искать способы исправить ситуацию.

Корпорация Microsoft не задержалась с предложением и выпустила на рынок программную технологию Microsoft .NET Framework, призванную объединить множество различных служб, написанных на разных языках, для общей совместимости. Эта виртуальная машина может быть установлена на разных семействах Windows, а также на других операционных системах, что позволяет использовать любой из языков NET-семейства для написания работоспособных приложений для всех операционных систем, на которых установлен framework. Одна из проблем, которая так долго преследовала клиентские приложения, частично была решена.

Итак, гонка клиентских и веб-приложений находится на той стадии, когда веб просто физически не может проникнуть глубже в ресурсы пользователя, чтобы увеличить быстродействие[35]. Возможности в реализации отдельных техник все же еще не доступны по сравнению с клиентскими приложениями и есть проблема постоянного обращения к серверу, потому что все еще мы работаем с обычным html-кодом.

Клиентские же приложение со своим богатством и простотой реализации сложнейших техник веба слишком неохотно потребляются пользователями, привыкшими к этому времени с помощью браузера решать самые сложные свои задачи. К тому же, клиентские приложения по прежнему привязаны к определенным протоколам передачи данных для обмена информацией. А это

влечет за собой дублирование определенных сервисов.

Так мы плавно перешли от истории к дням сегодняшним и видим, что бесспорного лидера нет, и каждая из технологий имеет свои достоинства и недостатки.

После долгих блужданий возле клиентского компьютера интернет- гиганты все же готовы смирится с тем, что для увеличения быстродействия отдельных сервисов, для дальнейшего усложнения систем придется рассчитывать на мощности своих серверов, а не пытаться максимально глубоко влезть в ресурсы пользователей. В связи с этим последнее время очень интенсивно пошло развитие сервисов, построенных для использования облачных вычислений (cloud computing) – технология обработки данных, в которой программное обеспечение предоставляется пользователю как Интернет-сервис. При этом пользователь не заботится об архитектуре облака, а лишь получает необходимые ему мощности от целых кластеров.

Такие сервисы на данный момент уже предоставляют Microsoft, Amazon (Elastic Compute Cloud).

Тем самым вся необходимая пользователю функциональность перемещается на сервера тех фирм, которые ее предоставляют. Доступ осуществляет через браузер, а, значит, отсутствует привязанность к разным семействам операционных систем.

Ярким примером может служить Gmail - почтовый клиент Google, предоставляющий богатый инструментарий для работы с почтой прямо из браузера.

Тем же временем продолжается совершенствование способов приблизить веб в клиентским приложениям. В 2006 году корпорация Microsoft выпустила плагин к IE – Silverlight, который позволяет запускать приложения, содержащие анимацию, векторную графику и аудио-видео ролики, что характерно для RIA (Rich Internet application).

Софтверные же компании имеют другую позицию – а именно видят будущее в smart client-ах – локальных приложениях, которые всецело ориентированы на потребление всевозможных сервисов из вне.

Smart Client — это легко устанавливаемое и управляемое клиентское приложение, предоставляющее пользователю адаптивный, отзывчивый и богатый пользовательский интерфейс, полностью использующее возможности локальных

ресурсов компьютера и интеллектуально управляющее взаимодействием с распределенными источниками данных.

Ключевыми особенностями, отличающими Smart Client, являются:

Богатый пользовательский интерфейс. Чтобы называться «умным», клиентское приложение должно иметь удобный пользовательский интерфейс, подстраиваясь под нужды пользователя, допуская персонализацию и предоставляя все современные способы управления (drag’n’drop, контекстные меню, дочерние окна, нотификации и т. д.)

Простая установка, не требующая участия пользователя. Приложение должно предлагать пользователю автоматическую установку, не требующую перезагрузки, долгого ожидания или большого объема закачиваемых файлов.

Автоматическая установка обновлений. Появление новых версий приложения должно автоматически проверяться, их установка так же должна происходить в автоматическом режиме. Возможность работы при отсутствии соединения с сервером. Если приложение в своей работе взаимодействует с удаленными источниками данных, оно также должно работать и предоставлять максимум возможной функциональности и при «отсоединенной» (оффлайн) работе. Примерами существующих смарт-клиентов могут быть:

IssueVision - help desk management application TaskVision[36] – клиентское приложение, которое позволяет подключенным пользователям создавать задачи, проекты и распределять их между другими пользователями. Взаимодействие между пользователями построено с использованием веб-сервисов.

Поскольку обмен структурированными данными между клиентом с сервисом производится с помощью стандартного языка XML – то приложение может взаимодействовать с большинством существующих сервисов, не зависимо от языка реализации. Однако даже с этими решениями у «smart» клиентов в случае прерывания связи с Internet только один выбор — отключаться, поэтому для устранения этого неудобства в Microsoft предложили технологию Live Mesh, позволяющую локально запускать Web- приложения. Звучит немного парадоксально – имеется в виду, что приложение может работать с данными и при следующем подключении уже синхронизировать их с сервером.

Такая возможность (работать оффлайн) также будет включена в последний Silverlight, что позволит даже с веб-страницами работать в оффлайн режиме.

В ближайшее время, как и последние много лет, основной средой обмена информацией останется интернет. Судя по тенденциям, клиентские и веб приложения будут развиваться параллельно, только немного другим путем – теперь это будут не монолитные порталы, написанные одной командой и использующие ресурсы одной эко-системы. Это будут наряженные ёлки – один костяк и множество подключенных сервисов, возможно даже разработанные разными фирмами. Это приведет к тому, что основное внимание и львиная доля времени будет расходоваться на разработку сложных сервисов, но потом они легко будут подключаться к всевозможным порталам,

приложениях и другим сервисам. Тем самым в скором будущем мы будем находиться не только во всемирной паутине, но еще и каждая ниточка этой паутины будет состоять из такого же сложного смешения различных сервисов, потребляемых различными устройствами. Но это огромное разнообразие сервисов будет полезно после окончательного внедрения нового протокола IPV6, что позволит подключать к интернету даже микроволновые печи, холодильники и т.д. Именно управление таким огромным количеством устройств в сети приведет к созданию множества сервисов и порталов, которые в онлайн режиме помогут управлять вашими электроприборами.

ЗАКЛЮЧЕНИЕ

Изучив и проанализировав архитектуру информационной системы, в структуру которой входят файл-сервер и клиент-сервер мною был сделан вывод, что файл-сервер во многом уступает клиент-серверу. Вся тяжесть вычислительной нагрузки при доступе к базе данных ложится на приложение клиента, что является следствием принципа обработки информации в системах файл-сервер при выдаче запроса на выборку информации из таблицы вся Рис. базы данных копируется на клиентское место, и выборка осуществляется на клиентском месте.

При этом возникают следующие ограничения:

- невозможность организации равноправного одновременного доступа пользователей к одному и тому же участку базы данных;

- количество одновременно работающих с системой пользователей не превышает пяти человек для ЛВС;

- невысокая скорость обработки и представления информации;

- высокие требования к ресурсам компьютеров.

При всем этом система обладает одним очень важным преимуществом - низкой стоимостью. Недостатки архитектуры файл-сервер решаются при переводе приложений в архитектуру клиент-сервер, достоинствами которой, является то, что вся вычислительная' нагрузка переносится на сервер базы данных, осуществляется высокая защита данных, поддерживается большое количество пользователей и сложных приложений. Рассмотрев языки запросов SQL и QBE, был сделан вывод, что SQL является наиболее гибким, динамичным, а также он поддерживает высокий уровень безопасности данных, их централизованное хранение и он ориентирован на конечный результат обработки данных.

В принципы разработки приложений архитектуры клиент-сервер входит обеспечение безопасности данных, организация взаимодействия клиента и сервера, все это достигается при использовании языка SQL.

Из сравнения сред программирования Microsoft SQL Server и Oracle, я сделала вывод, что Microsoft SQL Server отличается быстродействием, надежностью от Oracle, позволяет удовлетворить более широкие потребности клиентов по развертыванию крупномасштабных распределенных систем информации. SQL Server 6.0 обеспечивает мощные инструментальные средства для предприятий - широкой администрации, копирования данных, параллельного DBMS исполнения, и поиск в очень больших базах данных. Microsoft SQL Server 6.0 также обеспечивает плотную интеграцию OLE технологии. SQL Server 6.0 продолжает придерживаться промышленных стандартов, с улучшенной ANSI SQL поддержкой и языковыми расширениями, которые включают декларативную справочную целостность, и мощную поддержку сервер курсора, что значительно превышает стандарт ANSI.

СПИСОК ЛИТЕРАТУРЫ

1. http://www.intuit.ru/department/pl/distrsysjava/1/4.html

2. http://www.cio-world.ru/infrastructure/29164/

3. http://www.glossary.ru

4. http://ru.wikipedia.org

5. http://www.nadprof.ru/school/client-server.shtml

6. http://mrivkin.narod.ru/Publ/CLSERV1.htm

7. http://www.ci.ru/inform2_97/astr1.htm

8. http://belani.narod.ru/1/Lklser2.htm

9. http://www.mstu.edu.ru/education/materials/zelenkov/ch_7_1.html

10. http://www.intuit.ru/department/se/crosspl/1/2.html

11. http://www.intuit.ru/department/internet/mwebtech/5/2.html

12. http://www.feip.ru/2008/12/27/arkhitektura-klient-server.html

13. http://www.osp.ru/text/print/302/142618.html

14. http://mf.grsu.by/other/lib/klients/tonk_kl.htm

15. http://www.pcmag.ru/issues/detail.php?ID=8196

16. http://ru.wikipedia.org/wiki/Толстый

17. http://ydobno.net/article/future_of_client_app.aspx

1. http://ru.wikipedia.org ↑

2. http://www.nadprof.ru/school/client-server.shtml ↑

3. http://www.intuit.ru/department/pl/distrsysjava/1/4.html ↑

4. http://www.glossary.ru ↑

5. http://www.cio-world.ru/infrastructure/29164/ ↑

6. http://mrivkin.narod.ru/Publ/CLSERV1.htm ↑

7. http://belani.narod.ru/1/Lklser2.htm ↑

8. http://mrivkin.narod.ru/Publ/CLSERV1.htm ↑
9. http://www.intuit.ru/department/se/crosspl/1/2.html ↑
10. http://www.osp.ru/text/print/302/142618.html ↑
11. http://mf.grsu.by/other/lib/klients/tonk_kl.htm ↑
12. http://ydobno.net/article/future_of_client_app.aspx ↑
13. http://www.pcmag.ru/issues/detail.php?ID=8196 ↑
14. http://www.ci.ru/inform2_97/astr1.htm ↑
15. http://ru.wikipedia.org/wiki/Толстый ↑
16. http://ru.wikipedia.org ↑
17. http://ru.wikipedia.org ↑
18. http://belani.narod.ru/1/Lklser2.htm ↑
19. http://www.cio-world.ru/infrastructure/29164 ↑
20. http://www.mstu.edu.ru/education/materials/zelenkov/ch_7_1.html ↑
21. http://www.feip.ru/2008/12/27/arkhitektura-klient-server.html ↑
22. http://mf.grsu.by/other/lib/klients/tonk_kl.htm ↑
23. http://mf.grsu.by/other/lib/klients/tonk_kl.htm ↑

24. http://ru.wikipedia.org/wiki/Толстый ↑

25. http://ru.wikipedia.org/wiki/Толстый ↑

26. http://belani.narod.ru/1/Lklser2.htm ↑

27. http://mf.grsu.by/other/lib/klients/tonk_kl.htm ↑

28. http://www.feip.ru/2008/12/27/arkhitektura-klient-server.html ↑

29. http://www.intuit.ru/department/internet/mwebtech/5/2.html ↑

30. http://belani.narod.ru/1/Lklser2.htm ↑

31. http://mf.grsu.by/other/lib/klients/tonk_kl.htm ↑

32. http://www.feip.ru/2008/12/27/arkhitektura-klient-server.html ↑

33. http://mf.grsu.by/other/lib/klients/tonk_kl.htm ↑

34. http://www.feip.ru/2008/12/27/arkhitektura-klient-server.html ↑

35. http://www.intuit.ru/department/internet/mwebtech/5/2.html ↑

36. http://www.cio-world.ru/infrastructure/29164/ ↑