Содержание:

ВВЕДЕНИЕ

Ни для кого не секрет, что информационные технологии являются основой современного общества.

Инфромация, вращающаяся в обществе, по праву является важнейшим ресурсом современности.

Для обработки больших объемов информации люди привыкли использовать различные приложения, установленные на компьютерах. Однако, чем больший объем требуется обработать, тем большее количество ресурсов нужно затратить.

Технология клиент-сервер позволила людям выполнять обработку данных на более мощном вычислителе – сервере. Благодаря этому даже слабые компьютеры, которые подключены к серверу смогли быстро получать результат обработки больших данных.

Все о чем выше говорилось объясняет актуальность рассматриваемой темы.

Цель данной работы – изучить разные варианты архитектуры клиент-сервер.

Для достижения поставленной цели необходимо решить некоторые задачи:

- изучить литературу по заданной теме;
- изучить историю возникновения технологии;
- описать идею технологии;
- рассмотреть варианты архитектуры;
- изучить развитие технологии.

Курсовая работа состоит из трех глав. В первой приводится общее описание технологии, во второй рассматриваются варианты архитектуры клиент-сервер, третья глава посвящена развитию технологии.

1. Технология клиент-сервер

1.1. История

Под архитектурой информационной системы (ИС) как правило понимают концепцию, определяющую модель, структуру, выполняемые функции и взаимосвязь элементов ИС.

Клиент-сервер (Client-server) — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределяются между поставщиками услуг (сервисов), которые называются серверами, и заказчиками услуг - клиентами.

Сервер представляет собой программу, реализующую какие-либо услуги другим программам и обслуживающую клиентские запросы на получение ресурсов определенного типа.

Клиент — это программа, которая пользуется услугами, представляемыми сервером .

Зачастую под терминами «клиент» и «сервер» люди понимают компьютер, на котором функционирует какая-либо из этих программ. В действиетльности, клиент и сервер — это всего лишь роли, исполняемые программами. Физически клиенты и серверы могут даже размещаться на одном компьютере. Более того, даже одна и та же программа может одновременно являться и клиентом, и сервером.

Известно, что основной недостаток персонального компьютера состоит в достаточно невысокой вычислительной мощности и надежности, а также необходимости приобретения дополнительных аппаратных средств с целью устранения изолированности отдельных персональных компьютеров друг от друга.

Как правило, пользователям компьютеров требуется высокая вычислительная мощность, поэтому там, где для выполнения сложных вычислений применяются 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.

В результате опроса представителей трехсот крупнейших компаний США, использующих персональные компьютеры, выяснилось, что для 81% опрошенных требуется 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, для совместного использования компьютерами сети файлов, размещенных в различных узлах. Такая технология получила название «файл-сервер».

Однако файл-серверы обладают рядом недостатков. Так, например, данная архитектура не позволяет в полной мере обеспечить целостность и конфиденциальность данных. По сети файлы передаются целиком, независимо от того, какая часть содержащихся в них данных требуется пользователю. Это очень сильно загружает сеть и сокращает быстродействие системы. Надежность таких систем также невысока - сбой на одной из рабочих станций в момент записи файла становится причиной потери или искажения данных.

Чтобы данные не противоречили в подобных системах необходимо блокировать файлы, что является причиной замедления работы. Естественное желание специалистов в области вычислительной техники - совместить преимущества персональных компьютеров и мощных центральных компьютеров. Первый шаг в данном направлении - использование персональных компьютеров в роли интеллектуальных терминалов. В данном случае в персональном компьютере, который соединен с центральным компьютером, запускается специальное программное обеспечение, позволяющее этому персональному компьютеру работать в режиме эмуляции терминала. Значит получается архитектура, реализующая все преимущества архитектуры с мощным центральным компьютером, но, за исключением того, что персональный компьютер может использоваться и самостоятельно. В результате отсутствует необходимость двух дисплеев, однако есть недостатки, присущих архитектуре с центральным компьютером, их большенство. Кроме того, даже персональные компьютеры, имеющие дисплеи с картой VGA, и предоставляющие возможность работы с графикой, не могут применяться в качестве графических терминалов большой центральной машины. Данная задача реализуется центральным компьютером, в результате чего графические образы экрана передаются по проводам. Эти образы достаточно велики и скорость смены изображений на экране может быть очень низкой.

Следующий шаг в решении описанной выше проблемы - использование клиент-серверной архитектуры. В данном случае все компьютеры сети делятся на две группы: клиенты и серверы. Компьютер-сервер это мощный компьютер с большой оперативной памятью и большим объемом жесткого диска. Он хранит базу данных, а также отвечает за сложную обработку, требующую больших вычислительных ресурсов. На клиентских компьютерах выполняются такие операции как первичная обработка данных при вводе, форматирование данных, а также окончательная обработка данных, полученных с сервера. В качестве клиентских компьютеров чаще всего применяются персональные компьютеры типа IBM PC или Macintosh. Достоинства такого подхода очевидны. Каждый тип компьютера используется по своему назначению, таким образом достигается более полное использование возможностей компьютеров.

На компьютерах-клиентах функционируют знакомые пользователям PC пакеты, которые позволяют предоставлять результаты работы всей системы в виде, удобном для анализа и принятия решений. На этих компьютерах можно легко реализовать дружественный пользовательский интерфейс приложения при помощи графики, цвета, звука, работы с окнами и мышью и т.д. С помощью компьютер-клиент можно быстро выполнять ввод и первичный контроль данных. Для финальной обработки данных могут применяться те редакторы или пакеты электронных таблиц, которые наиболее удобны для пользователя. В качестве клиентских компьютеров могут одновременно применяться компьютеры разных типов с различными операционными системами.

Стоит отметить, что клиент-серверная архитектура позволяет реализовать распределенную обработку, поскольку часть работы (интерфейс с пользователем, финальная обработка) выполняется на компьютере-клиенте, а часть - на компьютере-сервере. Такой подход позволяет сократить загрузку сервера и оптимизировать его работу, а также увеличить количество клиентов, одновременно работающих с сервером.

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

1.2. Архитектура

Проблема выбора между распределенной и централизованной моделями предоставления вычислительных ресурсов сегодня считается одной из ключевых

проблем организации вычислительных систем. В качестве примера данной борьбы можно рассмотреть статью Джона Лесли Кинга «Централизованные и децентрализованные вычислительные системы: организационные соображения и варианты управления» 1983 г. До середины 70-х годов прошлого столетия по причине высокой стоимости телекоммуникационного оборудования и относительно невысокой мощности вычислительных систем преобладала централизованная модель. В конце 70-х годов в результате возникновения клиент-серверной архитектуры, обеспечивающей предоставление ресурсов мейнфреймов конечным пользователям посредством удаленного соединения, появились системы разделения времени и удаленных терминалов. Последующее развитие телекоммуникационных систем и появление персональных компьютеров послужило развитием клиент-серверной парадигмы обработки данных.

Архитектура клиент-сервер предполагает ситуацию, при которой один или несколько клиентов и один или несколько серверов совместно с базовой операционной системой и средой взаимодействия образуют единую систему, реализующую распределенные вычисления, а также анализ и представление данных. Использование такого подхода может позволить конечному пользователю персонального компьютера получить доступ к различным ресурсам удаленных серверов, например, файлам, принтерам, базам данных, процессорному времени и т.п.

В базовой модели клиент-сервер все процессы в распределенных системах принято делить на две перекрывающиеся группы. Процессы, реализующие некоторую службу, например, службу файловой системы или базы данных, принято называть серверами. Процессы, запрашивающие службы у серверов посредством посылки запроса и дальнейшего ожидания ответа от сервера, принято называть клиентами. Взаимодействие между клиентом и сервером предполагает режим работы запрос-ответ (см. рисунок 1).

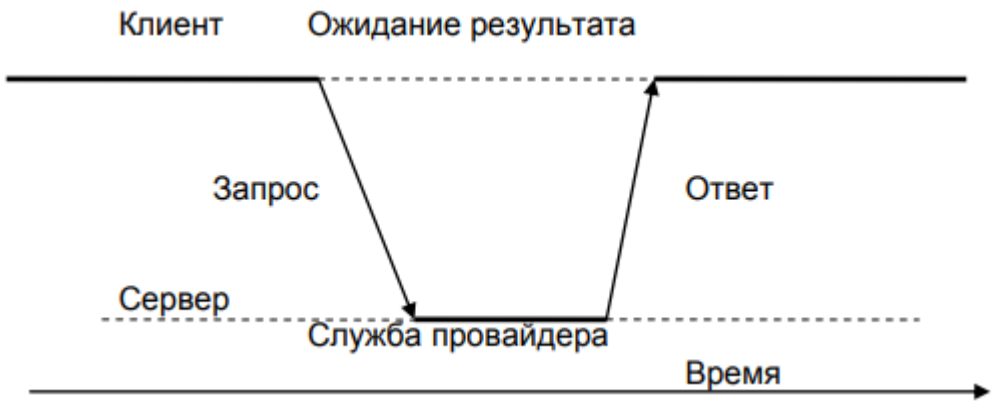

Рисунок 1 - Обобщенное взаимодействие между клиентом и сервером

Если базовая сеть так же надежна, как локальные сети, то взаимодействие между клиентом и сервером может быть реализовано без установки соединения. В таком случае клиент, запрашивая службу, формирует запрос в виде сообщения, указав в нем те службы, которыми он желает воспользоваться, а также данными, необходимыми для этого. Затем это сообщение посылается на сервер, который, в свою очередь, находится в ожидании входящего сообщения. Получив сообщение, сервер его обрабатывает, результат обработки упаковывает в ответное сообщение и отправляет его клиенту.

При использование протокола, не требующего соединения, получим значительный выигрыш в плане эффективности. Этот подход может успешно применяться до тех пор, пока сообщения не начнут повреждаться или пропадать. Однако, создать протокол, устойчивый к случайным сбоям связи, практически невозможно. Наиболее простым решением данной проблемы является предоставление клиенту возможности повторно послать запрос, на который он не получил ответ.

При этом возникает еще одна проблема - клиент не способен определить, действительно ли первоначальное сообщение с запросом было потеряно или ошибка произошла в процессе передачи ответа. Если потерялся ответ, повторная посылка запроса может привести к повторному выполнению операции. Если операция представляла собой что-то важное, например, «снять тысячу рублей с моего банковского счета», понятно, что было бы гораздо лучше, если бы вместо повторного выполнения операции клиент получал уведомление о произошедшей ошибке. С другой стороны, если операция была «сообщите мне, сколько денег у меня осталось», запрос прекрасно можно было бы послать повторно. Нетрудно заметить, что у этой проблемы нет однозначного решения.

В качестве альтернативы во многих системах клиент-сервер применяется надежный протокол с установкой соединения. Хотя данное решение по причине его относительно низкой производительности не слишком хорошо подходит для локальных сетей, оно прекрасно зарекомендовало себя в глобальных системах, для которых ненадежность является «врожденным» свойством соединений. Так, практически все прикладные протоколы сети Интернет основаны на надежных соединениях по протоколу TCP/IP. В этих случаях всякий раз, когда клиент запрашивает службу, он сперва должен установить соединение с сервером. Сервер обычно использует для посылки ответного сообщения то же самое соединение, после чего оно разрывается. В данном случае проблема заключается в том, что

установка и разрыв соединения в смысле затрачиваемого времени и ресурсов относительно дороги, особенно если сообщения с запросом и ответом невелики.

1.3. Разделение приложений по уровням

Модель клиент-сервер изначально была предметом множества споров и дебатов. Один из главных вопросов заключался в том, как именно разделить клиента и сервера. Рассматривая множество приложений типа клиент-сервер, предназначенных для организации доступа пользователей к базам данных, многие рекомендовали разделять их на три уровня:

- уровень представления (пользовательского интерфейса) - содержит все необходимое для непосредственного общения с пользователем, например, для управления дисплеем. Данный уровень обычно реализуется на стороне клиента. Он включает в себя программы, при помощи которых пользователь может взаимодействовать с приложением. Сложность программ, входящих в пользовательский интерфейс, весьма различна. Простейший вариант программы пользовательского интерфейса представляет собой символьный дисплей. Чаще всего подобные интерфейсы применяются при работе с мэйнфреймами. В том случае, когда мэйнфрейм контролирует все взаимодействия (в том числе работу с клавиатурой и монитором), нельзя говорить о модели клиент-сервер. Однако в большинстве случаев пользовательские терминалы производят некоторую локальную обработку, осуществляя, например, эхо-печать вводимых строк или предоставляя интерфейс форм, в котором можно отредактировать введенные данные до их пересылки на главный компьютер. Стоит отметить, что современные пользовательские интерфейсы гораздо более функциональны;
- уровень бизнес-логики (обработки) - содержит непосредственно приложения. Под бизнес-логикой здесь понимается совокупность правил, принципов и зависимостей поведения объектов предметной области системы. В качестве синонима данного понятия может использоваться термин "логика предметной области" (Domain Logic). Бизнес-логика представляет собой реализацию предметной области (например, бухгалтерского учета, методов управления предприятием и т.п.) в рамках информационной системы. Сюда относятся, например, формулы расчета ежемесячных выплат по ссудам (в финансовой индустрии), автоматизированная отправка сообщений электронной почты руководителю проекта по окончании выполнения частей задания всеми

подчиненными (в системах управления проектами), отказ от отеля при отмене рейса авиакомпанией (в туристическом бизнесе) и т. д.

- уровень данных - данные, с которыми происходит работа. Уровень данных содержит программы, которые отвечают за предоставление данных обрабатывающим их приложениям. Характерной чертой этого уровня является требование сохранности. Данное требование оворит о том, что когда приложение не работает, данные должны сохраняться в определенном месте в расчете на последующее использование. В простейшем варианте уровень данных реализуется за счет файловой системы, но чаще для его реализации применяется база данных. В модели клиент-сервер уровень данных обычно находится на стороне сервера. Кроме простого хранения информации уровень данных также отвечает за поддержание целостности данных для различных приложений. Для базы данных поддержание целостности означает, что метаданные, например, описания таблиц, ограничения и специфические метаданные приложений, также должны храниться на этом уровне. Обычно в деловой среде уровень данных организуется в форме реляционной базы данных. Ключевым моментом здесь является независимость данных. Данные организуются независимо от приложений таким образом, чтобы изменения в их организации не влияли на приложения, а приложения не влияли на организацию данных. Применение реляционных баз данных в модели клиент-сервер помогает отделить уровень обработки от уровня данных, рассматривая обработку и данные независимо друг от друга. Стоит отметить, что существует широкий класс приложений, для которых реляционные базы данных не являются лучшим выбором. Отличительной чертой этих приложений является работа со сложными типами данных, которые проще моделировать в понятиях объектов, а не отношений. Примеры таких типов данных – от простых наборов прямоугольников и окружностей до проекта самолета в случае систем автоматизированного проектирования. Также и мультимедийным системам гораздо проще работать с видео- и аудиопотоками, используя специфичные для них операции, чем с моделями этих потоков в виде реляционных таблиц. В тех случаях, когда операции с данными гораздо проще выразить в понятиях работы с объектами, имеет смысл реализовать уровень данных средствами объектно-ориентированных баз данных. Подобные базы данных не только поддерживают организацию сложных данных в форме объектов, но и хранят реализации операций над этими объектами. Таким образом, часть функциональности, приходившейся на уровень обработки, в этом случае переносится на уровень данных [10].

2. Варианты архитектуры

2.1. Двухуровневая

Первый вариантом архитектуры клиент-сервер принято называть одноуровневой (однозвенной) архитектурой. В данном случае клиент отвечал исключительно за отображение информации, предоставляемой со стороны сервера. В качестве примера такой архитектуры можно назвать удаленный рабочий стол или терминальный доступ к удаленному серверу. При этом весь объем вычислительной нагрузки приходится на сервер.

Следующим шагом в развитии клиент-серверной архитектуры стало появление двухуровневой архитектуры, представленной на рисунке 2.

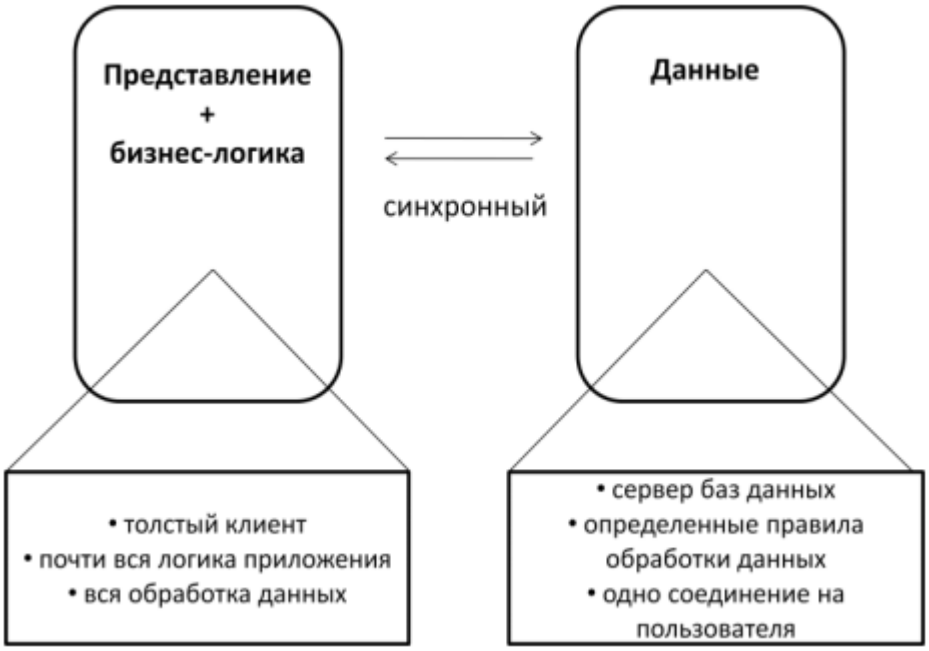

Рисунок 2 - Двухуровневая клиент-серверная архитектура

Особенностью этого подхода стало использование «толстых» клиентов, которые отвечали за реализацию основных задач по отображению информации пользователю и обработке всех данных. Центральный сервер выполняли лишь функции хранения и предоставления данных.

Данный подход считался наиболее популярным решением для корпоративных распределенных вычислительных систем вплоть до начала 2000-х годов.

В силу того, что большинство логики клиент-серверного приложения располагается в клиентской части, клиентская рабочая станция несет ответственность за большую часть обработки. Для оценки разделения объемов работ часто применяется соотношение 80/20: на сервер базы данных приходится порядка двадцати процентов всей работы. Несмотря на это, база данных зачастую является узким местом производительности в таких средах.

Двухуровневая клиент-серверная система требует, чтобы каждый клиент устанавливал свои собственные соединения с базой. Постоянная поддержка подобного множества соединений очень дорога. Кроме того, потребность в ресурсах иногда может стать причиной перегрузки сервера баз данных и задержки обработки пользовательских запросов.

Одной из ключевых причин отказа от двухуровнего подхода был постоянный рост расходов, необходимых на поддержку логики работы приложения на пользовательских рабочих станциях. Поскольку код приложения реализуется в каждом клиенте, каждое обновление приложения требует переустановки клиентского программного обеспечения на всех рабочих станциях. В больших сетях это приводит к высокой сложности администрирования.

Кроме того, возникает ряд вопросов по поддержанию работоспособности клиентских частей, поскольку рабочие станции могут иметь различный набор установленного ПО или, возможно, были приобретены у различных поставщиков оборудования. Также важно отметить, что при расширении возможностей клиентской программы, устаревший парк пользовательских рабочих станций может стать препятствием для обновления до новой версии системы [1].

2.2. Трехуровневая

Кроме описанного ранее варианта разделения двухуровневой архитектуры, также существует множество других подходов распределения программ, находящихся на уровне приложений по различным машинам, как показано на рисунке 3 [4].

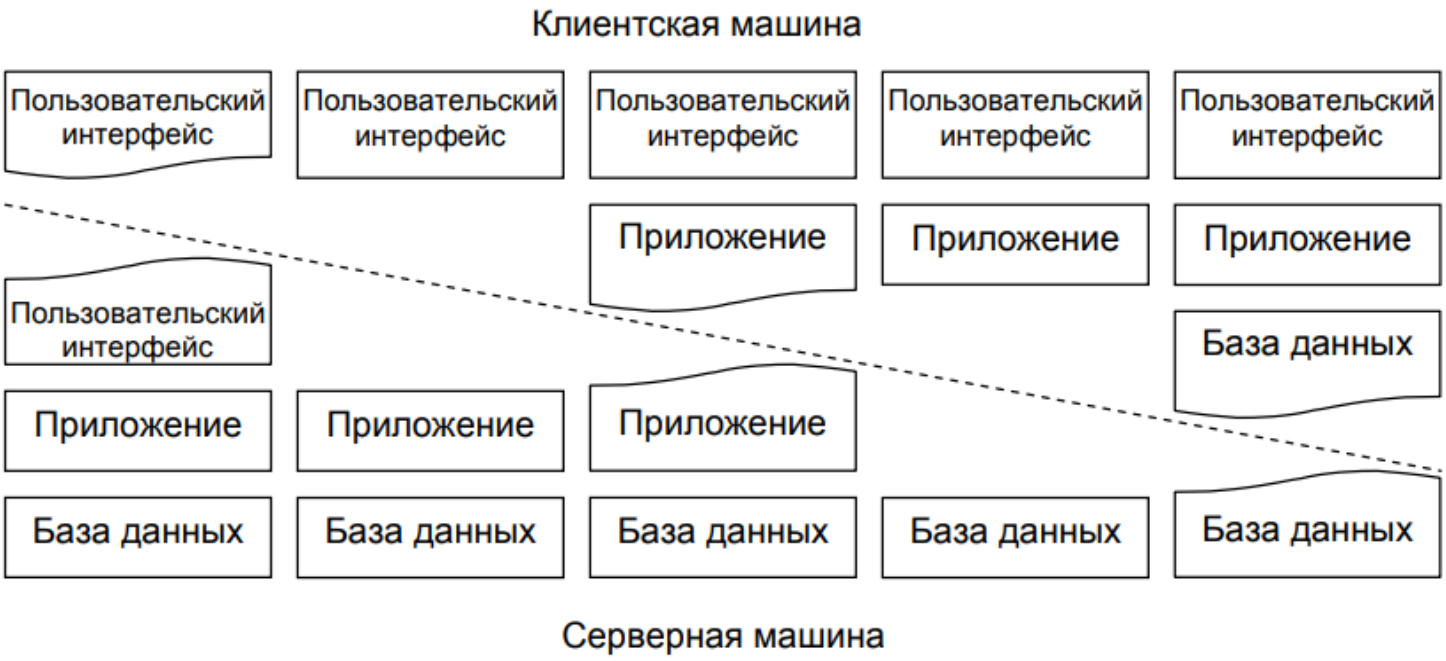

Рисунок 3 - Альтернативные формы организации двухуровневой архитектуры клиент-сервер

В ответ на затраты и ограничения, вызванные двухуровневой архитектурой, сформировалась концепция компонентно-ориентированной разработки приложений.

Многоуровневая клиент-серверная архитектура обеспечила переход к объектно-ориентированному подходу в рамках задачи построения распределенных вычислительных систем.

Использование данного подхода позволило распределить уровень бизнес-логики среди нескольких компонентов (часть – на клиенте, часть – на сервере) и уменьшить число проблем, возникающих при развертывании системы путем централизации большего количества логики на серверах. Серверные компоненты, перешедшие на выделенные сервера приложений, обеспечили возможность управления пулами соединений с базой данных. Это уменьшило количество одновременных соединений к базе, так как одно соединение в таком случае способно обеспечить работу нескольким клиентам (см. рисунок 4).

Рисунок 4 - Обобщенная организация трехуровневой архитектуры

В качестве примера описывается поисковая машина в Интернете, представленная на рисунке 5.

Пользовательский интерфейс поисковой машины очень прост: пользователь вводит строку, представляющую собой набор ключевых слов, а в ответ получает список заголовков web-страниц. Результат формируется из огромной базы просмотренных и проиндексированных web-страниц. В качестве ядра поисковой машины выступает программа, которая трансформирует введенную пользователем строку в один или несколько запросов к базе данных. Посе этого она помещает результаты запроса в список, преобразуя его в набор HTML-страниц. В рамках модели клиент-сервер часть, отвечающая за выборку информации, обычно располагается на уровне обработки [3].

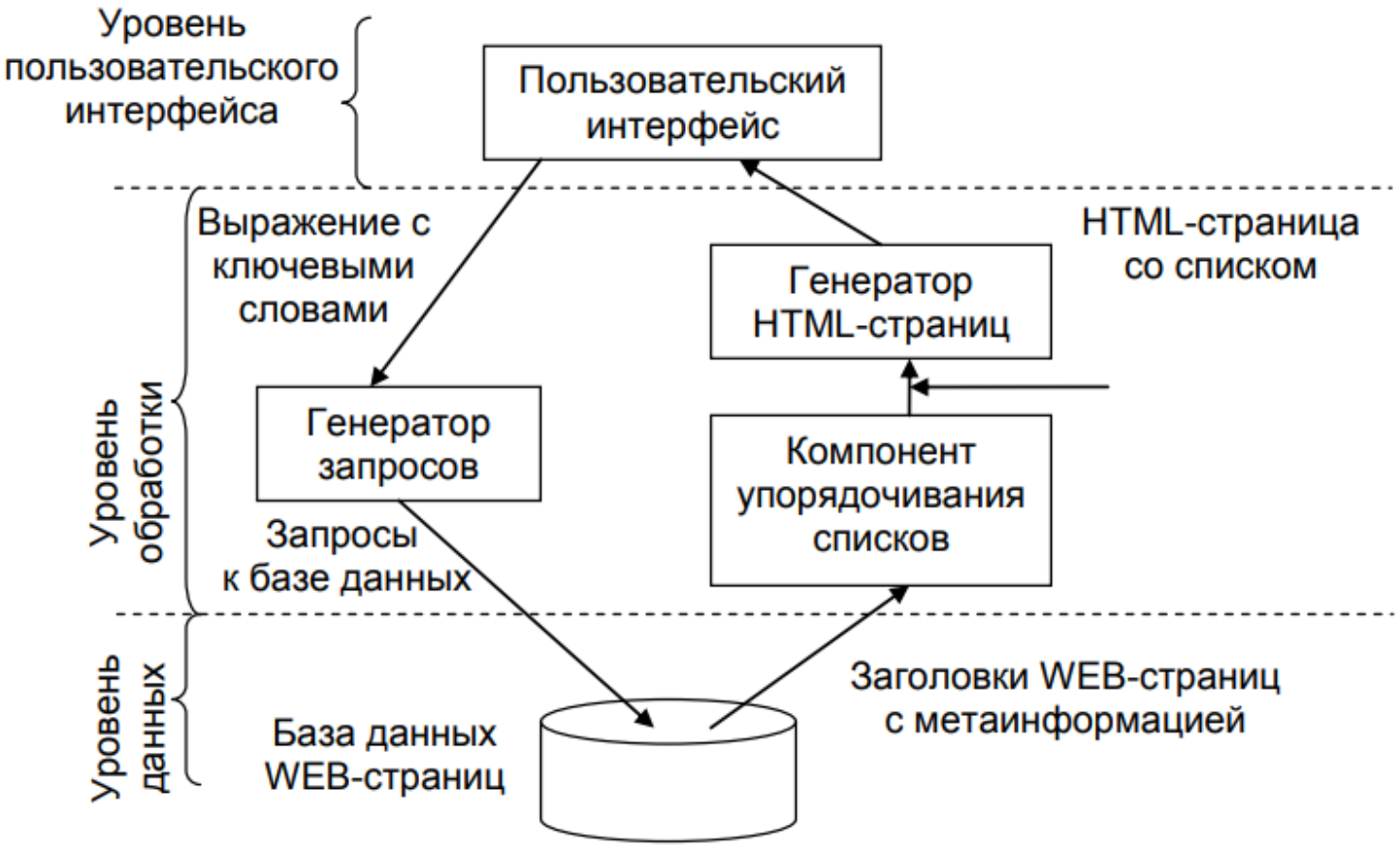

Рисунок 5 - Обобщенная организация трехуровневой поисковой машины для Интернета

2.3. Многоуровневая

Многоуровневые архитектуры являются прямым продолжением разделения приложений на уровни пользовательского интерфейса, компонентов обработки и данных.

Различные звенья взаимодействуют в соответствии с логической организацией приложения. В большинстве бизнес-приложений распределенная обработка эквивалентна организации многоуровневой архитектуры. Подобный тип распределения также называется вертикальным распределением. Отличительной чертой вертикального распределения является то, что оно достигается путем размещения логически различных компонентов на разных машинах. Данное понятие принято связывать с концепцией вертикального разбиения, применяемой в распределенных реляционных базах данных, где под этим термином понимается разбиение таблиц по столбцам для их хранения на различных машинах.

Однако вертикальное распределение – это лишь один из возможных способов организации приложений клиент-сервер. В современных архитектурах распределение на клиенты и серверы происходит способом, известным как горизонтальное распределение.В таком случае клиент или сервер могут содержать физически разделенные части логически однородного модуля, причем работа с каждой из частей может происходить независимо - это сделано для выравнивания загрузки.

В качестве распространенного примера горизонтального распределения рассмаривается web-сервер, реплицированный на несколько машин локальной сети (см. рисунок 6).

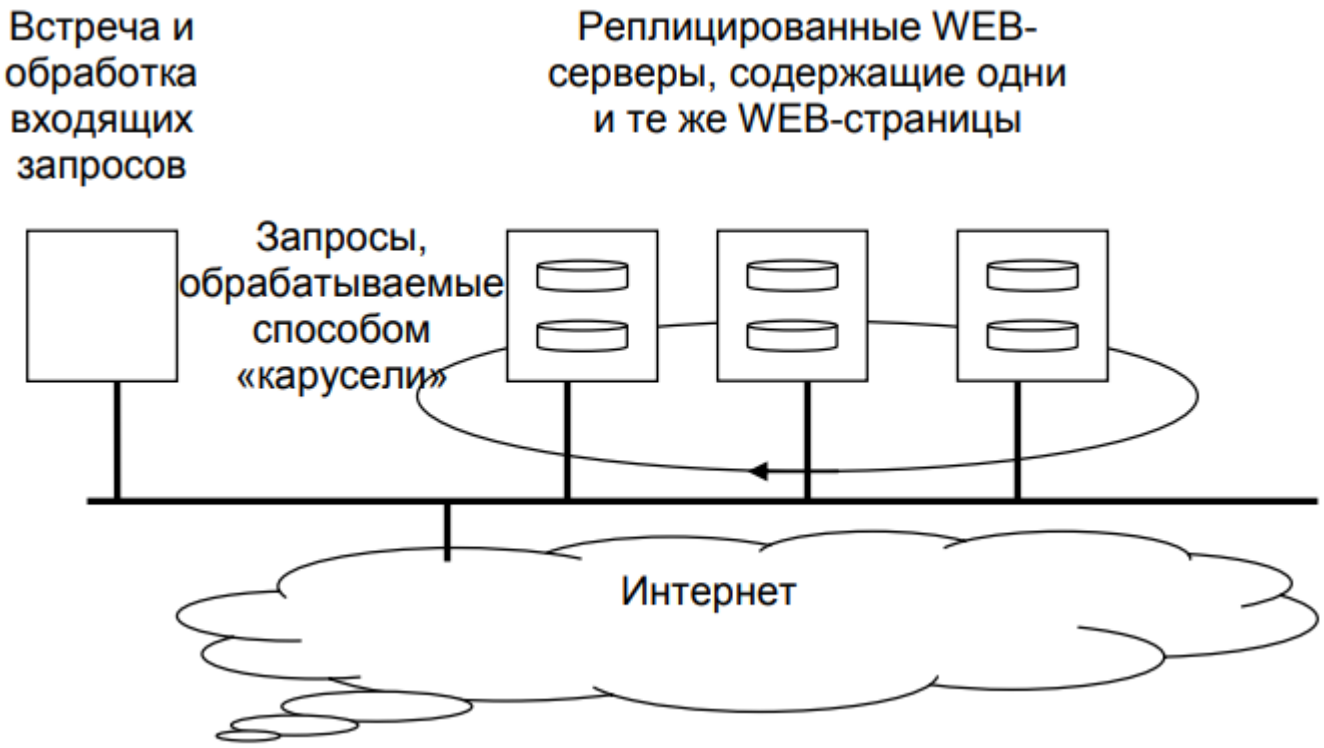

Рисунок 6 - Пример горизонтального распределения WEB-службы

Каждый из серверов содержит один и тот же набор web-страниц, и всякий раз, когда одна из web-страниц обновляется, ее копии незамедлительно рассылаются на все серверы.

Сервер, которому будет передан входящий запрос, выбирается по правилу «карусели». Эта форма горизонтального распределения достаточно успешно применяется для выравнивания нагрузки на серверы популярных web-сайтов.

Таким же образом могут распределяться и клиенты. Для несложного приложения, предназначенного для коллективной работы, можно вообще не использовать сервер - это одноранговое распределение. Подобное происходит, например, если пользователь хочет связаться с другим пользователем. Оба они должны запустить одно и то же приложение для начала сеанса. Третий клиент может общаться с одним из них или обоими, для чего ему нужно запустить то же самое приложение [9].

3. Развитие технологии

70-е и 80-е годы прошлого столетия принято считать эпохой централизованных вычислений на мэйнфреймах IBM, которые в то время составляли более 70% мирового компьютерного бизнеса.

Бизнес-транзакции, базы данных, запросы и техническое обслуживание - все реализовывалось на мэйнфреймах IBM. Этап перехода к клиент-серверным вычислениям представил принципиально новую концепцию и технологию организации всего делового мира. Эту парадигму называли «волной будущего».

Клиентская машина обычно управляет интерфейсами процессов, таких как графический интерфейс, отправкой запросов на сервер программ, проверкой данных, введенных пользователем, а также управляет местными ресурсами. При этом конечный пользователь только взаимодействует с устройствами - монитором, клавиатурой и т.п. Сервер же отвечает за выполнение клиентских запросов.

Задача клиент-серверных вычислений - позволить каждой клиентской станции и серверу быть доступными, по мере необходимости приложения, а так же обеспечивать доступ к существующему программному обеспечение и аппаратным компонентам от различных поставщиков для совместного использования. Если эти два условия соединены, преимущества клиент-серверной архитектуры очевидны - она позволяет экономить средства, а также увеличивать производительность и гибкость использования ресурсов.

В состав клиент-серверных входят три компонента:

- клиент - пользовательская станция, реализующая интерфейс приложения, проверку данных и отправку запросов на сервер. Кроме того, клиентский процесс также управляет локальными ресурсами;
- сервер - выполняет служебные запросы клиента. Сервер представляет собой программное обеспечение двигателя, управляющего общими ресурсами, таким

как базы данных, принтеры, линии связи, или процессоры высокой мощности. Основная цель серверного процесса - выполнение фоновых задач, общих для приложений. Простейшие формы серверов - это дисковый и файл-сервер. Если клиент передает запросы на файл или группы файлов по сети на файловый сервер, такая форма обслуживания данных требует большой пропускной способности и способна замедлить сеть с большим числом пользователей. Более продвинутые формы серверов - это серверы баз данных, сервер транзакций и сервер приложений;
- Middleware - промежуточный уровень, который позволяет приложениям прозрачно взаимодействовать с другими программами или процессами независимо от распоположения. Ключевой элемент Middleware - NOS (Network Operating System) - сетевая операционная система. Она предоставляет такие услуги, как маршрутизация, распределение, обмен сообщениями и управления сервисной сети. NOS полагается на коммуникацию протоколов предоставления конкретных услуг. Прежде чем пользователь получит доступ к услугам сети, клиент-серверный протокол требует от него установки физического соединения и выбора транспортных протоколов. Клиент-серверный протокол диктует, каким образом клиенты запрашивают информацию и услуги от сервера, а также как именно сервер должен отвечать на эту просьбу [8].

Мартин Батлер, председатель Butler Group, предложил новый способ реализации клиент-серверной стратегии - это пятислойная модель под названием VAL (Value Added Layers). Данная структура по форме напоминает пирамиду.

Далее представлены характеристики каждого слоя:

- Уровень 1 - Инфраструктура слоя - данный слой включает в себя все те компоненты, которые являются пассивными и не выполняют бизнес-функции;
- Уровень 2 - Middleware - позволяет приложениям взаимодействовать с другими программами или процессами. Это средство отображения приложений используемых ими ресурсов. Middleware является ключом к интеграции гетерогенных программных и аппаратных сред, реализуя тот уровень интеграции, который необходим большинству организаций;
- Уровень 3 - Программы - непосредственно приложения, являющиеся активными компонентами, выполняющими задачи по организации данных;
- Уровень 4 - Хранилище - хранилище предназначено для изоляции бизнес-модели и спецификации от технологических инструментов, которые используются для ее реализации;

- Уровень 5 - Бизнес-модели - набор независимых моделей, необходимых для того, чтобы все технологии, используемые для их реализации были применимы к программной и аппаратной среде в зависимости от того, что является наиболее подходящим [6].

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. 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 клиентсерверных 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 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 клиентсерверной 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. 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 – 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 Webприложения. 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 èлки – 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 [11].

ЗАКЛЮЧЕНИЕ

В рамках выполнения данной работы была рассмотрена тема «Варианты архитектуры клиент-сервер».

Первая глава работы обзорная. В ней дается понятие термину «клиент-сервер» - это вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределяются между поставщиками услуг (сервисов), которые называются серверами, и заказчиками услуг – это клиенты.

Сервер представляет собой программу, которая реализует какие-либо услуги другим программам и обслуживающую клиентские запросы на получение ресурсов определенного типа.

Клиент — это программа, которая пользуется услугами, которые представляет сервер.

Во второй главе представлены существующие варианты архитектуры клиент-сервер.

Первый вариантом архитектуры клиент-сервер принято называть одноуровневой (однозвенной) архитектурой. В данном случае клиент отвечал исключительно за отображение информации, которая предоставляется со стороны сервера.

Следующим шагом в развитии клиент-серверной архитектуры стало появление двухуровневой архитектуры. Особенностью этого подхода стало использование «толстых» клиентов, которые отвечали за реализацию основных задач по отображению информации пользователю и обработке всех данных. Центральный сервер выполняет лишь функции хранения и предоставления данных.

В ответ на затраты и ограничения, вызванные двухуровневой архитектурой, сформировалась концепция компонентно-ориентированной разработки приложений.

Многоуровневая клиент-серверная архитектура обеспечила переход к объектно-ориентированному подходу в рамках задачи построения распределенных вычислительных систем.

Третья глава работы посвящена описанию развития технологии, в результате которого можно сделать вывод о том, что клиент-сервер продолжит свое существование и в будущем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Вишневский В.М. Теоретические основы проектирования компьютерных сетей. – М.: Техноосфера, 2013. – 512 с.
2. Дубаков А.А. Сетевое программирование. – СПб.: НИУ ИТМО, 2015. – 248 с.
3. Елизаров И.А. Интеегрированные системы проектирования и управлени. – Тамбов: Изд-во ФГБОУ ВПО «ТГТУ» 2015. – 160 с.
4. Карпов А.Е. Архитектура распределенных систем программного обеспечения. – М.: МАКС Пресс, 2017. – 130 с.
5. Когаловский М.Р. Перспективные технологии информационных систем. – М.: ДМК Пресс, 2013. – 288 с.
6. Левин Л.И. Основы информационных технологий. – М.: Бином, 2012. – 336 с.
7. Рогозов Ю.И. Архитектура информационных систем / Ю.И. Рогозов, А.С. Свиридов, С.А. Кучеров. – Ростов-на-Дону: Изд-во ЮФУ, 2014. – 117 с.
8. Романчева Н.И. Базовые Интернет-технологии. – М.: МТУ ГА, 2014. – 96 с.
9. Рябов В.А. Современные веб-технологии. – М.: Интуит, 2010. – 475 с.
10. Советов Б.Я. Архитектура информационных систем. – М.: «Академия», 2012. – 288 с.
11. Таненбаум Э. Распределенные системы. Принципы и парадигмы. – СПб.: Питер, 2011. – 368 с.