4. Модель «Клиент-Сервер»

Согласно парадигме *клиент-серверной архитектуры* несколько клиентов и несколько серверов совместно с промежуточным программным обеспечением и средой взаимодействия образуют единую систему, обеспечивающую распределенные вычисления, анализ и представление данных. Использование клиент-серверного подхода позволило пользователю персонального компьютера получить доступ к различным ресурсам удаленных серверов, таких как базы данных, файлы, принтеры, процессорное время и др.

В базовой модели клиент-сервер все процессы в распределенных системах делятся на две возможно перекрывающиеся группы. Процессы, реализующие некоторый сервис, например, сервис файловой системы или базы данных, называются *серверами*. Процессы, запрашивающие сервисы у серверов путем посылки запроса и последующего ожидания ответа от сервера, называются *клиентами*.

Если базовая сеть так же надежна, как локальные сети, взаимодействие между клиентом и сервером может быть реализовано посредством простого протокола, не требующего установления соединения (например, протокол UDP). В этом случае клиент, запрашивая сервис, облекает свой запрос в форму сообщения с указанием в нем сервиса, которым он желает воспользоваться, и необходимых для этого исходных данных. Затем сообщение посылается серверу. Последний, в свою очередь, постоянно ожидает входящего сообщения, получив его, обрабатывает, упаковывает результат обработки в ответное сообщение и отправляет его клиенту.

Использование не требующего соединения протокола дает существенный выигрыш в эффективности. До тех пор пока сообщения не начнут пропадать или повреждаться, можно вполне успешно применять протокол типа запрос-ответ. К сожалению, создать протокол, устойчивый к случайным сбоям связи, – нетривиальная задача. Все, что мы можем сделать – это дать клиенту возможность повторно послать запрос, на который не был получен ответ. Проблема, однако, состоит в том, что клиент не может определить, действительно ли первоначальное сообщение с запросом было потеряно или ошибка произошла при передаче ответа. Если потерялся ответ, повторная посылка запроса может привести к повторному выполнению операции. Если операция представляла собой что-то вроде «снять 10 000 долларов с моего банковского счета», понятно, что было бы гораздо лучше, если бы вместо повторного выполнения операции вас

просто уведомили 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 единого решения.

В качестве альтернативы во многих системах клиент-сервер используется надежный протокол с установкой соединения (например, протокол TCP). Хотя это решение в связи с его относительно низкой производительностью не слишком хорошо подходит для локальных сетей, оно великолепно работает в глобальных системах, для которых ненадежность является «врожденным» свойством соединений. Так, практически все прикладные протоколы Интернета основаны на надежных соединениях на основе стека TCP/IP. В этом случае всякий раз, когда клиент запрашивает сервис, до посылки запроса серверу он должен установить с ним соединение. Сервер обычно использует для посылки ответного сообщения то же самое соединение, после чего оно разрывается. Проблема состоит в том, что установка и разрыв соединения в смысле затрачиваемого времени и ресурсов относительно дороги, особенно если сообщения с запросом и ответом невелики. Мы обсудим альтернативные решения, в которых управление соединением объединяется с передачей данных, в следующей главе.

4.1 Разделение приложений по уровням

Модель клиент-сервер была предметом множества дебатов и споров. Один из главных вопросов состоял в том, как точно 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);
- уровень бизнес-логики (обработки);
- уровень данных.

Уровень пользовательского интерфейса содержит все необходимое для непосредственного общения с пользователем, например, для управления дисплеем. Уровень обработки обычно содержит приложения, а уровень данных – собственно данные, с которыми происходит работа. В следующих пунктах мы обсудим каждый из этих уровней.

4.1.1 Уровень представления

Уровень пользовательского интерфейса обычно реализуется на клиентах. Этот уровень содержит программы, посредством которых пользователь может взаимодействовать с приложением.

Сложность программ, входящих в пользовательский интерфейс, весьма различна. Простейший вариант программы пользовательского интерфейса не содержит ничего, кроме символьного (не графического) дисплея. Такие интерфейсы обычно используются при работе с мейнфреймами. В том случае, когда мейнфрейм контролирует все взаимодействия, включая работу с клавиатурой и монитором, мы вряд ли можем говорить о модели клиент-сервер («однозвенная» архитектура). Однако во многих случаях терминалы пользователей производят некоторую локальную обработку, осуществляя, например, эхопечать вводимых строк или предоставляя интерфейс форм, в котором можно отредактировать введенные данные до их пересылки на главный компьютер. Современные пользовательские интерфейсы значительно более функциональны.

4.1.2 Уровень бизнес-логики

Бизнес-логика – это совокупность правил, принципов и зависимостей поведения объектов предметной области системы. Синонимом данного понятия является *логика предметной области* (анг. Domain Logic).

Бизнес-логика – это реализация предметной области (например, бухгалтерского учета, обучения студентов, методов управления предприятием и др.) в информационной системе. К ней относятся, например, формулы расчёта 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.

4.1.3 Уровень данных

Уровень данных в модели клиент-сервер содержит программы, которые предоставляют данные обрабатывающим их приложениям. Специфическим свойством этого уровня является *требование сохранности.* Это означает, что когда приложение не работает, данные должны сохраняться в определенном месте в расчете на дальнейшее использование. В простейшем варианте уровень данных реализуется файловой системой, но чаще для его реализации задействуется полномасштабная база данных. В модели клиент-сервер уровень данных обычно находится на стороне сервера.

Кроме простого хранения информации уровень данных обычно также отвечает за *поддержание целостности* данных для различных приложений. Для базы данных поддержание целостности означает, что метаданные, такие как описания таблиц, ограничения и специфические метаданные приложений, также хранятся на этом уровне.

Обычно в деловой среде уровень данных организуется в форме реляционной базы данных. Ключевым здесь является независимость данных. Данные организуются независимо от приложений так, чтобы изменения в организации данных не влияли на приложения, а приложения не оказывали влияния на организацию данных. Использование реляционных баз данных в модели клиент-сервер помогает нам отделить уровень обработки от уровня данных, рассматривая обработку и данные независимо друг от друга.

Однако существует обширный класс приложений, для которых реляционные базы данных не являются наилучшим выбором. Характерной чертой таких 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 значительно проще работать с видео- и аудио потоками, используя специфичные для них операции, чем с моделями этих потоков в виде реляционных таблиц.

В тех случаях, когда операции с данными значительно 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, мигрирует в этом случае на уровень данных.

4.2 Типы клиент-серверной архитектуры

Исторически, первым вариантом клиент-серверной архитектуры являлась так называемая «Однозвенная» архитектура, характеризующаяся тем, что клиент не нес никакой функциональной нагрузки, кроме отображения информации, предоставляемой мейнфреймом (сервером). Примером такой архитектуры может являться терминальный доступ к удаленному серверу или удаленный рабочий стол. В этом случае весь объем вычислительной нагрузки приходился на сервер.

Следующим шагом в развитии клиент-серверной архитектуры было появление так называемой двухзвенной архитектуры (рис. 6).

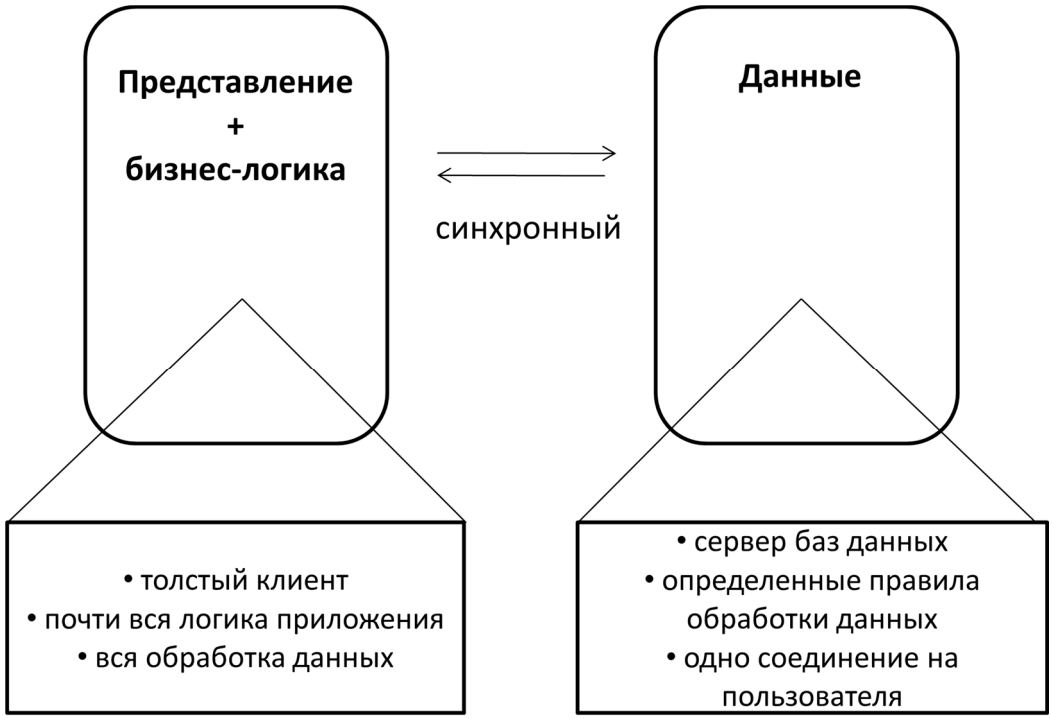

Рис. 6. Двухзвенная клиент-серверная архитектура

Особенностью данного подхода является использование «толстых» клиентов, на которые возлагались основные задачи по отображению информации пользователю и обработке всех данных. Центральный сервер реализовывал лишь функции хранения и предоставления данных. Этот подход являлся наиболее распространенным решением для корпоративных распределенных вычислительных систем вплоть до начала 2000-х годов.

Поскольку большинство логики клиент-серверного приложения находится в клиентской части, клиентская рабочая станция несет ответственность за большую часть обработки. Для оценки разделения объемов работ часто используется соотношение 80/20: сервер базы данных обычно выполняет порядка двадцати процентов работы. Несмотря на это, база данных часто становится узким местом производительности в этих средах.

Двухуровневая клиент-серверная система часто требует, чтобы каждый клиент устанавливал свои собственные соединения с базой данных. Постоянная поддержка такого множества соединений с базой данных стоят дорого, и потребность в ресурсах иногда может привести к перегрузке сервера баз данных и задержки обработки запросов всех пользователей.

Одной из основных причин отказа от двухзвенного клиент-серверного подхода было постоянное увеличение расходов на поддержание логики работы

приложения на рабочих станциях пользователей. Поскольку код приложения реализуется в каждом клиенте, каждое обновление для приложения требует переустановки клиентского программного обеспечения на всех рабочих станциях. В больших средах это приводит к высокой сложности администрирования.

Также, поскольку рабочие станции могут иметь различный набор установленного ПО или, возможно, были приобретены у различных поставщиков оборудования, возникает ряд вопросов по поддержанию работоспособности клиентских частей. Кроме того, при расширении возможностей клиентской части приложения, устаревший парк рабочих станций может стать препятствием для обновления до новой версии системы в связи с ограниченными возможностями старых клиентских машин.

В ответ на затраты и ограничения, связанные с двухуровневой клиент-серверной архитектурой, зародилась концепция многоуровневой клиент-серверной архитектуры (рис. 7).

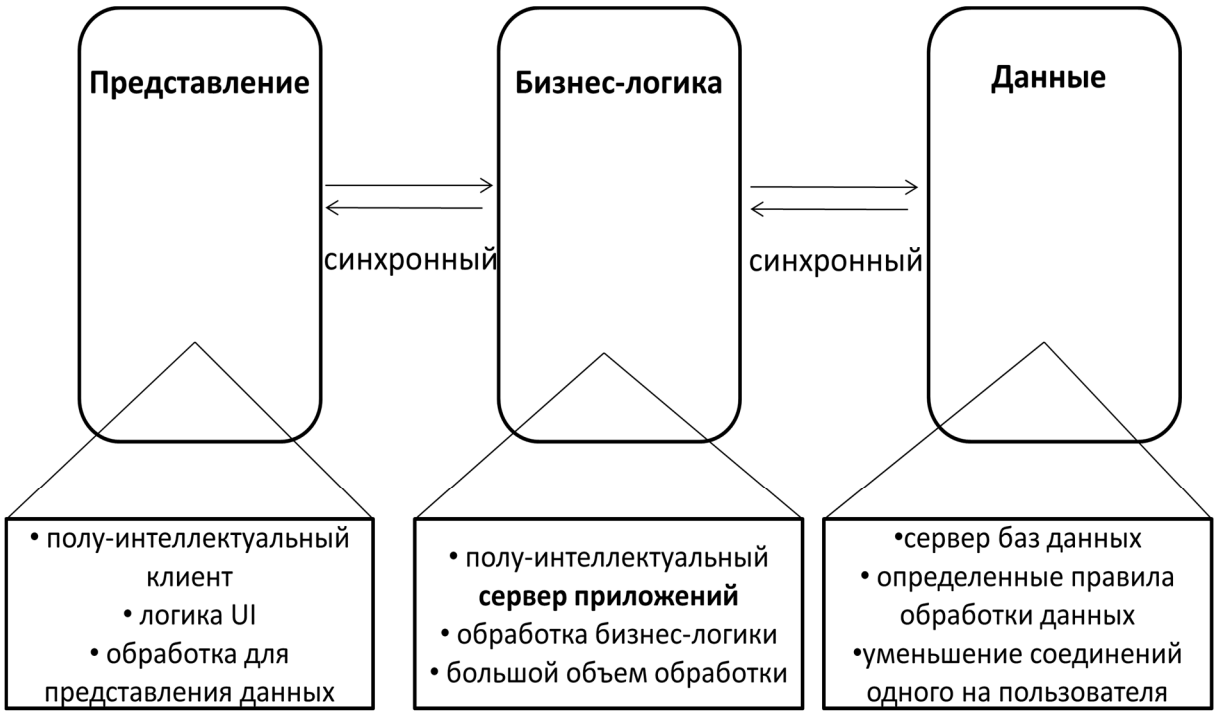

Рис. 7. Обобщенная организация трехзвенной архитектуры

Применение такого подхода позволило распределить уровень бизнес-логики среди нескольких компонентов (часть – на клиенте, часть – на сервере) и уменьшить количество проблем 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, облегчая задачу серверу баз данных

посредством значительного уменьшения одновременного количества соединений, так как одно соединение может обеспечить работу нескольким клиентам.

В качестве примера рассмотрим поисковую машину в Интернете. Пользовательский интерфейс поисковой машины очень прост: пользователь вводит строку, состоящую из ключевых слов, и получает список заголовков веб-страниц. Результат формируется из гигантской базы просмотренных и проиндексированных веб-страниц. Ядром поисковой машины является программа, трансформирующая введенную пользователем строку в один или несколько запросов к базе данных. Затем она помещает результаты запроса в список и преобразует этот список в набор HTML-страниц. В рамках модели клиент-сервер часть, которая отвечает за выборку информации, обычно находится на уровне обработки (рис. 8).

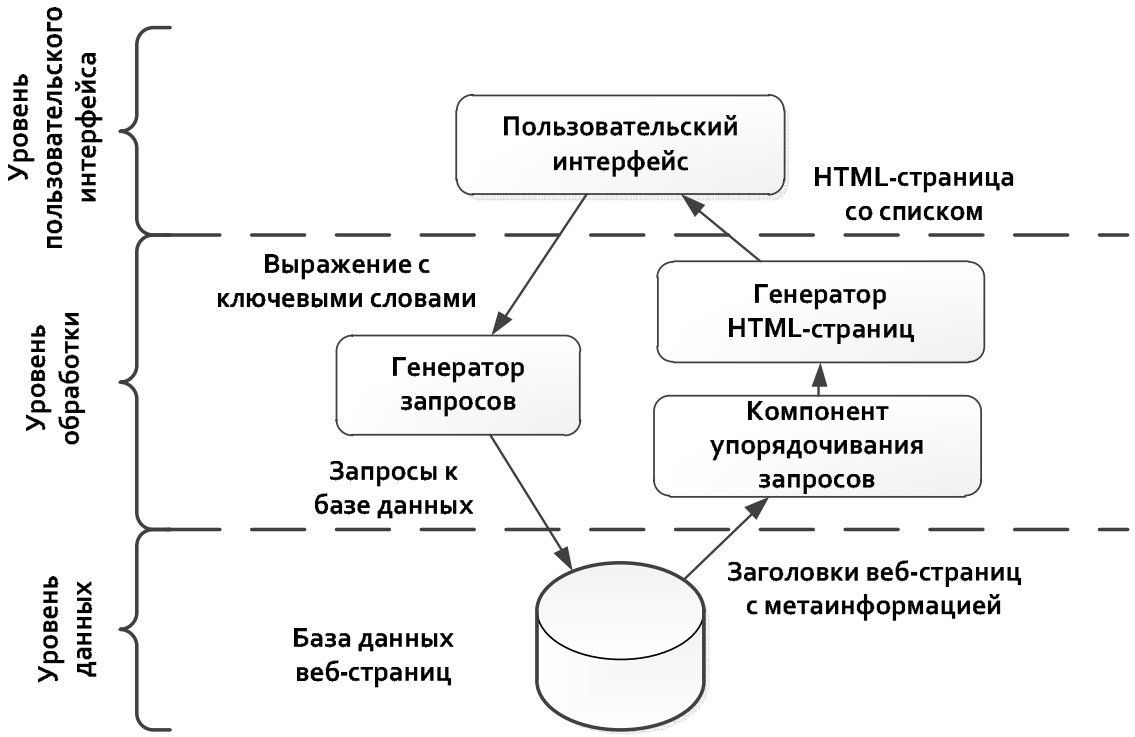

Рис. 8. Обобщенная организация трехуровневой поисковой машины для Интернета

4.2.1 Методы горизонтального распределения

Многозвенные архитектуры клиент-сервер являются прямым продолжением разделения приложений на уровни пользовательского интерфейса, компонентов обработки и данных. Различные звенья взаимодействуют в соответствии с логической организацией приложения. Во множестве бизнес-приложений распределенная обработка эквивалентна организации многозвенной архитектуры приложений клиент-сервер. Мы будем называть такой тип распределения *вертикальным распределением*.

Характеристической особенностью вертикального распределения является то, что оно достигается *размещением логически различных компонентов на разных машинах*. Это понятие связано с концепцией вертикального разбиения, используемой в распределенных реляционных базах данных, где под этим термином понимается разбиение по столбцам таблиц для их хранения на различных машинах.

Однако вертикальное распределение – это лишь один из возможных способов организации приложений клиент-сервер, причем во многих случаях наименее интересный. В современных архитектурах распределение на клиенты и серверы происходит способом, известным как *горизонтальное распределение*. При таком типе распределения клиент или сервер могут содержать физически разделенные части логически однородного модуля, причем работа с каждой из частей может происходить независимо. Это делается для выравнивания загрузки.

В качестве распространенного примера горизонтального распределения рассмотрим веб-сервер, реплицированный на несколько машин локальной сети (рис. 9).

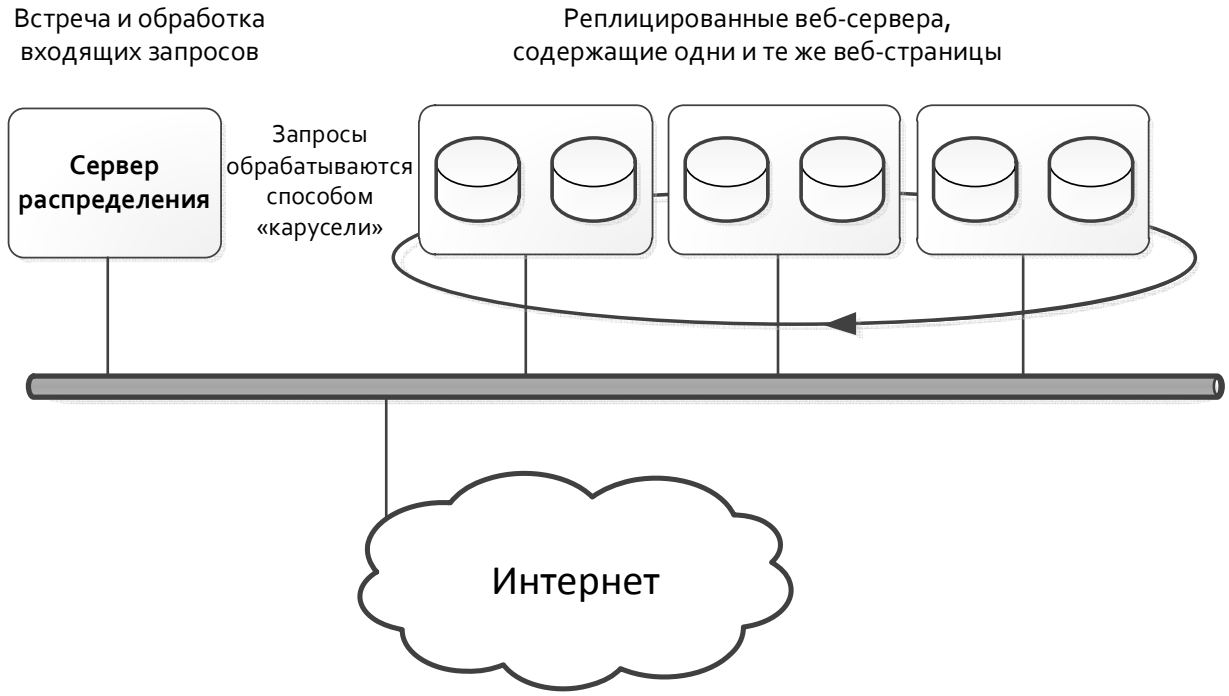

Рис. 9. Пример горизонтального распределения веб-сервера

На каждом из серверов содержится один и тот же набор веб-страниц. Всякий раз, когда одна из веб-страниц обновляется, ее копии незамедлительно рассылаются на все серверы. Сервер, которому будет передан приходящий запрос, выбирается по правилу «карусели». Эта форма горизонтального распределения

весьма успешно используется для выравнивания нагрузки на серверы популярных веб-сайтов.

Таким же образом, хотя и менее очевидно, могут быть распределены и клиенты. Для несложного приложения, предназначенного для коллективной работы, мы можем не иметь сервера вообще. В этом случае мы обычно говорим об одноранговом распределении. Подобное происходит, например, если пользователь хочет связаться с другим пользователем. Оба они должны запустить одно и то же приложение, чтобы начать сеанс. Третий клиент может общаться с одним из них или обоими, для чего ему нужно запустить то же самое приложение.

Самым ярким примером применения горизонтального клиент-серверного распределения могут служить программные системы, созданные на основе концепции облачных вычислений. Более подробно об этом мы поговорим в главе 12 «Облачные вычисления».