Содержание:

Введение

Технология «клиент-сервер» пришла на смену централизованной схеме управления вычислительным процессом на мейнфреймах еще в 80-х годах прошлого века. Благодаря высокой живучести и надежности вычислительной системы, легкости масштабирования, возможности одновременной работы пользователя с несколькими приложениями, высокой оперативности обработки информации, обеспечению пользователя высококачественным интерфейсом и другим возможностям эта весьма перспективная и далеко не исчерпавшая себя технология получила свое дальнейшее развитие.

Со временем малофункциональную модель файлового сервера для локальных сетей (FS) заменили появившиеся одна за одной модели структуры «Клиент-сервер» (RDA, DBS и AS).

Заняв нишу баз данных, технология «Клиент – сервер» стала основной технологией глобальной сети Internet. Далее, в результате перенесения идей сети Internet в среду корпоративных систем, появилась технология Intranet. В отличие от технологии «Клиент-сервер» эта технология ориентирована не на данные, а на информацию в ее окончательно готовом к потреблению виде. Вычислительные системы, построенные на основе Intranet, имеют в своем составе центральные серверы информации и распределенные компоненты представления информации конечному пользователю (программы-навигаторы, или браузеры). Взаимодействие между клиентом и сервером в Intranet происходит при помощи web – технологий.

На сегодняшний день технология «Клиент-сервер» получает все большее распространение, однако сама по себе она не предлагает универсальных рецептов. Она лишь дает общее представление о том, как должна быть организована современная распределенная информационная система. В то же время реализации этой технологии в конкретных программных продуктах и даже в видах программного обеспечения различаются весьма существенно.

Глава 1. Архитектура клиент-сервер

1.1. Клиент-сервер

Архитектура "клиент - сервер" (Рис. 1.1) - это в настоящее время наиболее распространенная архитектура, в которой выполнено, пожалуй, большинство работающих информационных систем. Существует даже мнение, что почти все остальные архитектуры могут быть представлены как большая или меньшая вариация этой, базовой.

Рис. 1.1. Архитектура "клиент - сервер"

В традиционном понимании система, выполненная в архитектуре клиент-сервер, представляет собой совокупность взаимодействующих компонент двух типов - клиентов и серверов. Обычным также является разнесение этих компонент по узлам двух типов - соответственно узлам - клиентам и узлам - серверам. Клиенты обращаются к серверам с запросами, сервера их обрабатывают и возвращают результат. Клиент, вообще говоря, может обращаться с запросами к нескольким серверам. Сервера также могут обращаться с запросами друг к другу. Таким образом, типичный протокол для одного факта взаимодействия может быть представлен в виде двух обменов - запрос на сервер и ответ сервера. Наиболее часто встречающийся класс приложений, выполненных в архитектуре клиент-

сервер - различные приложения, работающие с базами данных. В таком случае в качестве сервера выступает СУБД, обеспечивающая выполнение запросов клиента, который в свою очередь реализует интерфейс пользователя. Рассмотренные далее модели систем, по сути, являются вариациями архитектуры клиент – сервер.[2]

Появление архитектуры клиент-сервер, как и многих других новых компьютерных технологий, сопровождалось рождением новой терминологии.

1.2. Вычислительная модель клиент-сервер

Разделение приложения на отдельные задачи, размещаемые на различных платформах для большей эффективности. Как правило, это означает, что программа представления данных находится на машине пользователя (на клиенте), а программа управления данными и сами данные — на сервере. В зависимости от приложения и используемого программного обеспечения вся обработка данных может осуществляться на клиентской машине или распределяться между клиентом и сервером. Сервер соединяется со своими клиентами по сети. Серверное программное обеспечение принимает запросы от клиентского программного обеспечения и возвращает ему результаты.

Сетевое окружение, в котором управление данными осуществляется на серверном узле, а другим узлам предоставляется доступ к данным

Совместная с клиентом обработка запросов клиента сервером и возвращение результатов клиенту. В этой модели обработка данных приложением распределена (не обязательно поровну) между клиентом и сервером. Обработка данных инициируется и частично управляется клиентом, но не в режиме «ведущий-ведомый», а, скорее, в режиме сотрудничества [5]

Модель взаимодействия между одновременно выполняемыми программными процессами. Клиентские процессы посылают запросы серверному процессу, посылающему обратно результаты этих запросов. Как предполагает название, серверные процессы предоставляют услуги своим клиентам, как правило, выполняя специфическую обработку, которую могут выполнить только они. Клиентский процесс, освобожденный от выполнения сложной обработки транзакции, может выполнять другую полезную работу. Взаимодействие между клиентским и серверным процессами представляет собой совместный транзакционный обмен, в котором активность исходит от клиента, а сервер

реагирует на эту активность.[9]

1.3. Эволюция архитектуры клиент-сервер

Идея организовать вычислительные ресурсы в соответствии с архитектурой клиент-сервер возникла на уровне рабочих групп и подразделений компаний. Менеджеры отделов обнаружили, что использование централизованных приложений, работающих на мэйнфрейме, не дает им достаточно быстро реагировать на требования бизнеса. Развертывание приложений центральным отделом информационного обслуживания требовало очень много времени, а результаты не всегда соответствовали нуждам подразделений. С развитием персональных компьютеров работники получили возможность хранить и обрабатывать данные прямо на своих рабочих местах, а менеджеры отделов — быстро находить нужные приложения.

Однако в окружении, состоящем исключительно из персональных компьютеров, пользователи при совместной работе сталкивались с определенными трудностями. Даже на уровне одного подразделения компании было необходимо поддерживать базу данных, а также форматы и стандарты их использования. Решение проблем дает архитектура клиент-сервер, развернутая на уровне подразделения. Как правило, подобная архитектура включает одну локальную сеть, несколько персональных компьютеров и один или два сервера.

Успех систем клиент-сервер уровня подразделения проложил дорогу системам клиент-сервер уровня предприятия. В идеальном случае подобная архитектура позволяет интегрировать ресурсы подразделений компании и отдела информационного обслуживания, а также запускать приложения, предоставляющие пользователям контролируемый доступ к корпоративным базам данных. Важной чертой подобных архитектур является то, что центральный отдел информационного обслуживания вновь возвращает себе полный контроль над данными, но уже в контексте распределенной вычислительной системы.[1]

1.4. Приложение клиент-сервер

Любое приложение, в котором инициатор действия находится в одной системе, а исполнитель действия — в другой. Кроме того, в большинстве приложений клиент-

сервер один сервер обслуживает запросы нескольких клиентов.

Как предполагает термин, окружение клиент-сервер состоит из клиентов и серверов. Клиентские машины, как правило, представляют собой однопользовательские персональные компьютеры или рабочие станции, предоставляющие конечным пользователям дружественный интерфейс. Клиентская станция обычно имеет наиболее удобный графический интерфейс пользователя, предполагающий наличие окон и мыши. Наиболее известные примеры подобных интерфейсов — интерфейсы операционных систем MicrosoftWindowsи Macintosh. Клиентские приложения предполагают простоту использования и знакомые инструментальные средства, например, электронные таблицы.

Каждый сервер в окружении клиент-сервер предоставляет клиентам набор услуг. Наиболее распространенным типом сервера в архитектуре клиент-сервер является сервер баз данных, как правило, управляющий реляционной базой данных. Высокопроизводительный сервер обеспечивает коллективный доступ нескольких клиентов к одной и той же базе данных.

Помимо клиентов и серверов в окружение клиент-сервер входит сеть. Вычислительная модель клиент-сервер по определению является распределенной. Пользователи, приложения и ресурсы располагаются на разных компьютерах и соединены общей локальной, глобальной или составной сетью.

В чем отличие конфигурации клиент-сервер от других распределенных решений? Есть несколько характеристик, отличающих вычислительную модель клиент-сервер от обычных схем распределенных вычислений.

- В приложениях клиент-сервер большое внимание уделяется созданию на клиентской машине дружественного пользователю интерфейса. Таким образом, пользователь получает полный контроль над расписанием и режимом работы компьютера, а менеджеры уровня отделов получают возможность реагировать на локальные проблемы.
- Хотя приложения являются распределенными, в архитектуре клиент-сервер, как правило, используются централизованные корпоративные базы данных. Это позволяет руководству корпорации сохранять полный контроль над инвестициями в информационные системы, а также обеспечивать полную связность всех систем. В то же время такая конфигурация избавляет различные отделы компании от накладных расходов по управлению сложными

вычислительными системами, но позволяет им выбирать типы машин и интерфейсы, которые им необходимы для доступа к данным.
- Как корпоративные пользователи, так и производители отдают предпочтение открытым и модульным системам. Это означает, что пользователю предоставляется более широкий выбор продуктов и большая свобода в объединении оборудования от различных производителей.
- Компьютерная сеть является ключевым звеном данной архитектуры. Поэтому вопросы сетевого администрирования и сетевой безопасности при работе с информационными системами данного типа имеют приоритет.

С одной стороны, архитектура клиент-сервер представляет собой естественное решение с точки зрения производителя, так как в ней используются все более доступные микрокомпьютеры и сети. С другой стороны, архитектура клиент-сервер, возможно, является идеальным выбором для поддержки выбранного организацией направления бизнеса.

Последнее утверждение требует пояснений. Успех архитектуры клиент-сервер на рынке объясняется не новыми названиями старых решений. Вычислительная модель клиент-сервер действительно представляет собой новый технический метод распределенных вычислений. Но помимо этого, архитектура клиент-сервер создает условия для новых методов организации бизнеса. Рассмотрим две важные тенденции в промышленности, иллюстрирующие этот факт.[3]

Первая тенденция заключается в том, что компании постоянно пытаются снижать трудовые затраты и избавляться от лишних рабочих мест, что вызвано жесткой рыночной конкуренцией. Почему компаниям необходимо сокращать рабочие места, чтобы сохранить конкурентоспособность, и как им удается увеличивать производительность, добиваясь роста продаж без увеличения числа сотрудников? Стоимость каждого рабочего места стремительно растет, и этот рост сопровождается ростом заработной платы. В то же время стоимость рабочего оборудования, особенно компьютерного и сетевого, а также сетевого обслуживания увеличивается гораздо более скромными темпами. Все это привело, как и следовало ожидать, к существенному росту капиталовложений в компьютеры и информационные технологии, чтобы компенсировать сокращение штата сотрудников.

Эта тенденция просматривается как в малом, так и в крупном бизнесе и затрагивает управленцев среднего звена, а также конторских служащих. Архитектура клиент-сервер предоставляет средство автоматизации задач и

устранения барьеров для информации, что позволяет компаниям удалять лишние управленческие звенья и увеличивать производительность, не раздувая штаты.

Другой тенденцией, иллюстрирующей эффективность архитектуры клиент-сервер, является так называемое движение внутреннего рынка. Это движение затрагивает, в первую очередь, крупный бизнес, пытающийся сочетать предпринимательское рвение с корпоративной мощью, чтобы получить лучшее и от того, и от другого: экономию, обусловленную крупными масштабами большого бизнеса, и гибкость малого бизнеса. В эпоху быстрых технологических и рыночных изменений многие крупные компании отказались от традиционной функциональной иерархии, заменив ее набором относительно независимых организационных единиц. Эти организационные единицы должны конкурировать с внешними компаниями. На внутреннем рынке каждая организационная единица функционирует как независимая компания. Каждая организационная единица сама решает, что и у кого ей покупать (независимо от того, является поставщик подразделением той же корпорации или же внешней компанией). Даже такие традиционные непроизводительные подразделения, как бухгалтерия, информационные системы и юридический отдел, должны продавать свои услуги другим подразделениям и конкурировать с внешними поставщиками.

Внутренняя конкуренция призвана исправить недостатки традиционного метода ведения бизнеса. В [9] отмечается, что «Американские корпорации являются одними из самых крупных социалистических бюрократий в мире. Они характеризуются централизованным планированием, централизованным владением капитала, централизованным распределением ресурсов, субъективной оценкой труда, отсутствием внутренней конкуренции и склонностью принимать решения в ответ на политическое давление».

Движение внутренних рынков уже трансформировало некоторые компании, и обещает оказать существенное влияние на другие. Однако до недавних пор на пути реализации подобной схемы было одно труднопреодолимое препятствие. В крупной компании наличие внутреннего рынка может привести к тому, что тысячам отделов придется постоянно договариваться друг с другом и с внешними организациями. Проанализировав эту ситуацию, можно предположить, что стоимость и сложность бухгалтерского учета всех транзакций превысит пользу от введения внутреннего рынка. Это препятствие было преодолено благодаря развитию вычислительных технологий. Сегодня ряд транснациональных корпораций пользуются новейшими системами управления базами данных, работающими в сетях с архитектурой клиент-сервер, что позволяет им внедрять

идею внутреннего рынка [3].

Таблица 1 - Достоинства и недостатки архитектуры клиент-сервер

Системная характеристика	Значение
Достоинства	
Сеть небольших мощных машин	Если одна машина выйдет из строя, ваша компания все равно сможет продолжать работу
Мощные объединения компьютеров	Система предоставляет мощность, позволяющую выполнять работу без монополизации ресурсов. У конечных пользователей достаточно мощностей для локальной работы
Некоторые рабочие станции столь же мощны, как мэйнфреймы, но их стоимость на порядок ниже	Предоставляя вычислительные мощности за меньшие деньги, система позволяет вам тратить сэкономленные средства на другие приобретения или на увеличение ваших доходов
Открытые системы	Аппаратуру, программы и услуги можно приобретать у разных поставщиков
Легкость наращивания системы	Вашу систему нетрудно модернизировать, как только ваши потребности изменятся
Индивидуальная рабочая среда клиента	Вы можете объединять компьютерные платформы, подбирая их под конкретные нужды подразделений и пользователей
Недостатки	

Слабая поддержка	Отдельные части системы не всегда корректно работают вместе. Бывает довольно трудно найти причину неисправности
Недостаток инструментальных средств обслуживания	При использовании архитектуры клиент-сервер часто приходится искать инструментальные средства на рынке или разрабатывать их самостоятельно
Необходимость переобучения	Философия программирования для Mae или Windows существенно отличается от философии программирования на языке COBOL или C

Эти и другие тенденции в мире бизнеса послужили стимулом к увеличению инвестиций в технологию клиент-сервер. Разумеется, как и любое кардинальное изменение компьютерной конфигурации, переход на архитектуру клиент-сервер не является ни безопасным, ни безболезненным. В табл. 1 показано, что пользователи при переходе на архитектуру клиент-сервер, помимо получаемых преимуществ, сообщают о множестве проблем. Тем не менее благодаря снижению стоимости и росте популярности персональных компьютеров, а также благодаря растущей конкуренции в промышленности, архитектура клиент-сервер в обозримом будущем будет, скорее всего, доминировать в бизнесе.

Важнейшей особенностью вычислительной модели клиент-сервер является распределение прикладных задач между клиентами и серверами. Иллюстрация общего случая приведена на рис. 2. Как на клиенте, так и на сервере базовым программным обеспечением является, разумеется, операционная система. Аппаратные платформы и операционные системы клиентов и серверов могут отличаться. В самом деле, в едином окружении могут использоваться разные типы клиентских и серверных платформ и операционных систем. Однако эти различия не имеют значения, если сервер и клиент используют одни и те же коммуникационные протоколы и поддерживают одинаковые приложения.

Взаимодействие клиента и сервера обеспечивается коммуникационным программным обеспечением. Примерами такого программного обеспечения являются набор протоколов TCP/IP,- протоколы OSI, а также различные фирменные

архитектуры, вроде SNA.

Разумеется, назначение всего этого программного обеспечения поддержки (протоколов и операционной системы) заключается в предоставлении базы для распределенных приложений. В идеальном случае выполняемая приложением функция должна быть распределена между клиентом и сервером таким образом, чтобы вычислительные и сетевые ресурсы использовались оптимально, а пользователи получили оптимальные возможности для выполнения различных задач и совместной работы. В некоторых случаях для этого может быть необходимо, чтобы большая часть программного обеспечения выполнялась на сервере, тогда как в других случаях большая часть логики может быть реализована на клиенте

часть логики может быть реализована на клиенте.

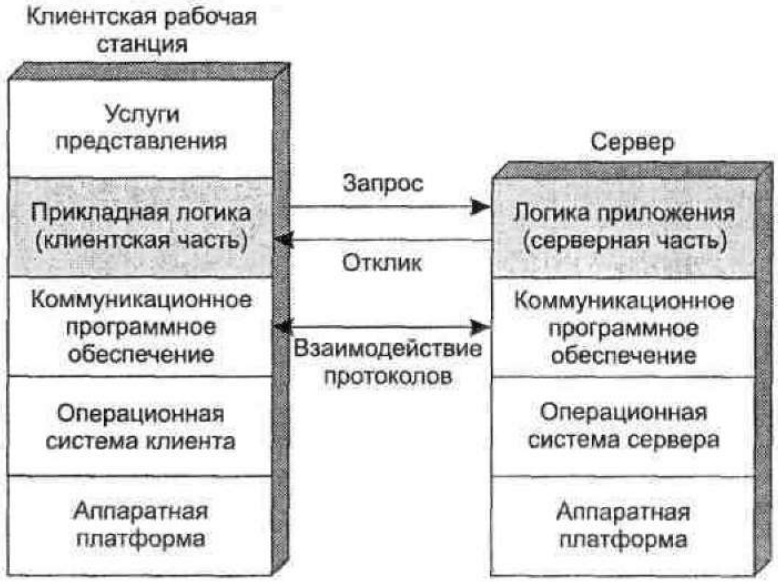

Рис. 17.3. Общая архитектура клиент-сервер

Рис. 2 – Общая архитектура клиент – сервер

Наконец, существенным фактором успеха является метод взаимодействия пользователя с системой, то есть большое значение имеет пользовательский интерфейс клиентской машины. В большинстве систем клиент-сервер *графическому интерфейсу пользователя*(GraphicalUserInterface, GUI) уделяется очень серьезное внимание — он должен быть простым и удобным, но одновременно мощным и гибким. Таким образом, модуль услуг представления на рабочей станции можно считать ответственным за дружественный интерфейс с распределенными приложениями.

Модуль услуг представления не следует путать с уровнем представления эталонной модели OSI. Уровень представления занимается форматированием данных для их корректной интерпретации каждым из двух взаимодействующих компьютеров. Модуль услуг представления имеет дело с взаимодействием пользователя и приложения, а также с функциональностью графического интерфейса пользователя.[6]

1.5. Базы данных

Рассмотрим концепцию распределенной между клиентом и сервером логики приложения на примере реляционной базы данных. В этой среде сервер является сервером баз данных. Взаимодействие между клиентом и сервером осуществляется в форме транзакций, в которых клиент посылает серверу запрос и получает ответ на него.

Архитектуру этой системы иллюстрирует рис. 3. Сервер отвечает за управление базой данных. На клиентских машинах могут располагаться различные приложения, пользующиеся базой данных. Специальное программное обеспечение связывает клиента и сервера, позволяя клиенту выполнять запросы и получать доступ к базе данных. Популярным примером такой логики является язык структурированных запросов (StructuredQueryLanguage, SQL).

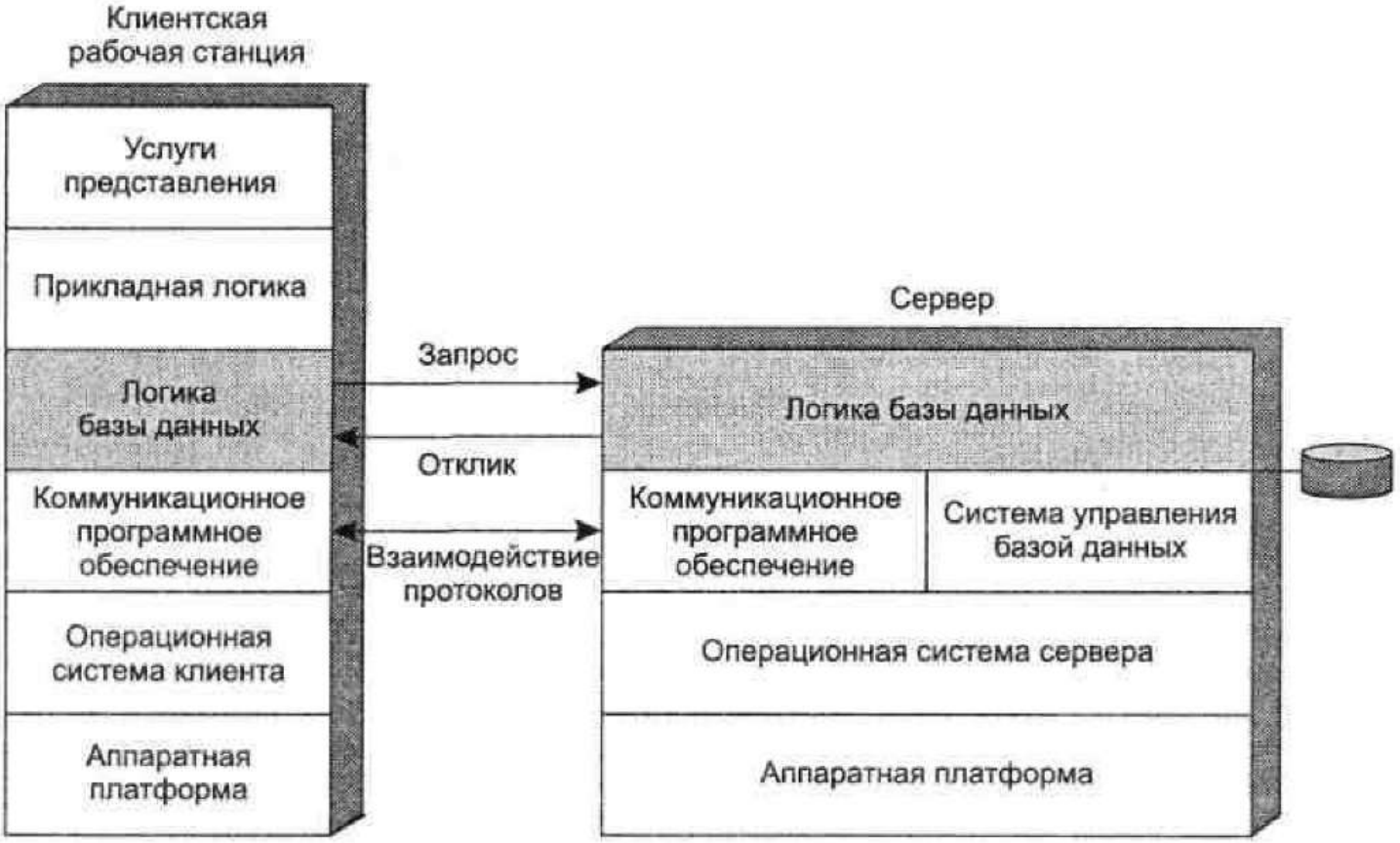

Рис. 3 - Архитектура клиент-сервер для баз данных

На рис. 3 предполагается, что вся прикладная логика — программы для обработки и анализа данных — располагается на клиентской стороне, тогда как сервер занимается только управлением базой данных. Приемлемость такой конфигурации зависит от стиля и задачи конкретного приложения. Предположим, например, что основная цель приложения заключается в обеспечении доступа для поиска записей в режиме подключения (on-line). Пример работы такой схемы показан на рис. 4, *а.* Предположим, что сервер управляет базой данных, содержащей один миллион записей (на жаргоне реляционных баз данных называемых строками), и пользователь хочет выполнить операцию поиска, результатом которой может быть нуль записей, одна запись или небольшое количество записей. Пользователь может искать эти записи по нескольким критериям поиска (например, записи, сделанные ранее 1992 года; записи, касающиеся жителей штата Огайо; записи, относящиеся к специфическим событиям или характеристикам, и т. д.). Начальный запрос клиента вызывает ответ сервера о том, что в базе данных содержится 100 000 записей, удовлетворяющих критериям поиска.

При этом пользователь задает дополнительные критерии, и посылает новый запрос. На этот раз сервер отвечает, что в базе данных содержится 1000 записей,

удовлетворяющих критериям поиска. Наконец, клиент посылает третий запрос с дополнительными критериями. Результатом поиска на этот раз является одна запись, которая возвращается клиенту.[8]

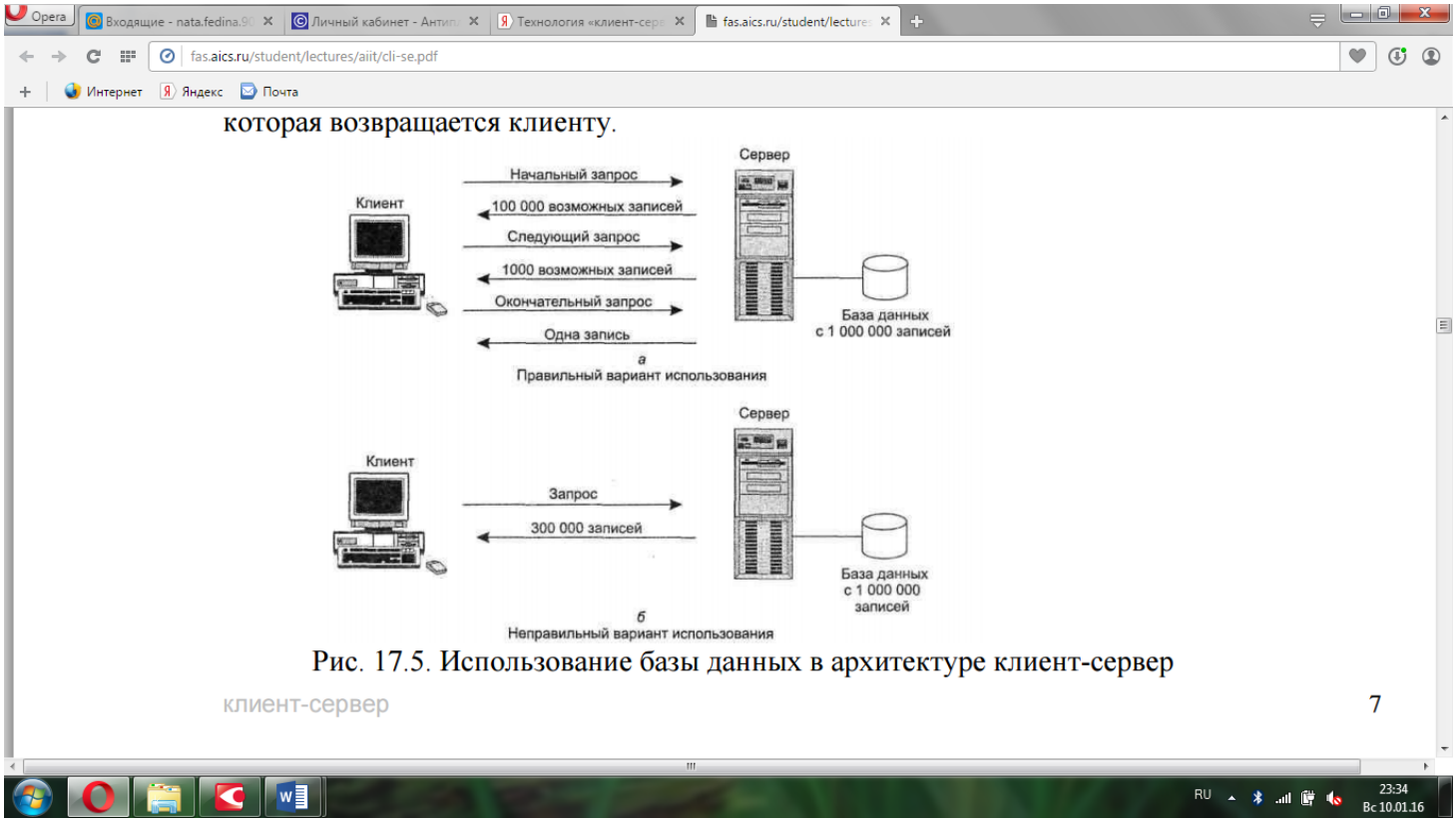

Рис. 4 - Использование базы данных в архитектуре клиент-сервер

Данное приложение хорошо соответствует архитектуре клиент-сервер по двум причинам.

- С базой данных производится большой объем работ по сортировке и поиску данных. Для этого необходим большой диск или массив дисков, высокоскоростной центральный процессор и высокоскоростная архитектура ввода-вывода. Такие мощности не нужны на однопользовательской рабочей станции или на персональном компьютере.
- Перемещение на клиентскую машину файла с миллионом записей для поиска явилось бы слишком тяжелым бременем для сети. Таким образом, серверу недостаточно просто получать доступ к записям от имени клиента. Сервер должен обладать логикой базы данных, позволяющей ему выполнять операции поиска от имени клиента.

Рассмотрим теперь рис. 4, *б,* который иллюстрирует сценарий работы с той же самой базой данных, содержащей миллион записей. В этом случае в результате

одного запроса пользователю по сети пересылается 300 000 записей. Такое может случиться, если, например, пользователь захочет определить сумму или среднее значение определенного поля в большом множестве записей или даже во всей базе данных.[7]

Разумеется, подобные действия являются недопустимыми. Одно из возможных решений данной проблемы состоит в том, чтобы переместить часть логики приложения на сервер. То есть помимо функций доступа к записям базы данных и функций поиска записей, сервер может быть оснащен прикладной логикой анализа данных.

1.6. Классы приложений клиент-сервер

В рамках общей структуры приложений клиент-сервер выполняемая работа может быть разделена между клиентом и сервером по-разному. Точная доля исполняемых операций и объем передаваемых по сети данных зависят от природы информации, содержащейся в базе данных, поддерживаемых типов приложений, доступности оборудования, которое может работать совместно, а также от характера использования данных в организации.

Схемы некоторых основных вариантов приложений баз данных показаны на рис 5. Возможны также другие варианты распределения задач между сервером и клиентом. В любом случае стоит изучить этот рисунок, чтобы получить представление о возможных компромиссах.

На рисунке изображены четыре класса приложений с разными вариантами распределения задач между сервером и клиентом.

- *Обработка данных на базе хоста.* Данная схема не является настоящим приложением клиент-сервер, а относится к традиционному окружению мэйнфрейма, когда вся или практически вся обработка данных осуществляется на главной вычислительной машине. Зачастую в подобной вычислительной среде интерфейс пользователя предоставляет примитивный терминал. Но даже если пользователь сидит за персональным компьютером, роль персонального компьютера в этом случае ограничивается эмуляцией терминала.
- *Обработка данных на базе сервера.* Простейшим классом конфигурации клиент-сервер является схема, в которой клиент отвечает лишь за

предоставление графического интерфейса пользователя[1], тогда как практически вся обработка данных осуществляется на сервере.

- *Обработка данных па базе клиента.* Данная схема представляет собой другую крайность. Практически вся обработка данных осуществляется на клиенте, за исключением процедур проверки целостности данных и прочей логики, относящейся к обслуживанию базы данных, которые лучше исполнять на сервере. Как правило, наиболее клиент-серверсложные функции для работы с базой данных располагаются на клиентской стороне.
- *Совместная обработка данных.* В данной конфигурации обработка данных оптимизирована таким образом, чтобы использовать сильные стороны как клиента, так и сервера, а также самого факта распределения данных. Подобные конфигурации гораздо сложнее в установке и обслуживании, но в долговременной перспективе они позволяют обеспечить лучшие показатели производительности и эффективности использования сетевых ресурсов, чем другие методы реализации архитектуры клиент-сервер.

1.7. Трехзвенная архитектура клиент-сервер

Традиционная архитектура клиент-сервер состоит из двух уровней, или звеньев: клиентского и серверного. В последние годы все большее распространение получает трехзвенная архитектура клиент-сервер (рис. 5). В данной архитектуре прикладное программное обеспечение распределено между машинами трех типов: пользовательской машиной, промежуточным сервером и сервером базы данных. Машина пользователя представляет собой клиента, и в трехзвенной модели это, как правило, «тонкий» клиент. Промежуточные машины являются, по существу, шлюзами между «тонкими» клиентами и разнообразными серверами баз данных. Промежуточные машины должны уметь преобразовывать протоколы и отображать одни типы запросов к базам данных на другие. Вдобавок промежуточные машины должны объединять результаты запросов от различных источников данных. Наконец, эти машины должны играть роль шлюзов между настольными приложениями и оставшимися у организации со старых времен системами управления базами данных, таким образом, являясь посредниками между двумя «мирами». По существу, такая архитектура означает интеграцию приложений предприятия (EnterpriseApplicationIntegration, EAI).[6]

Взаимодействие между промежуточным сервером и сервером баз данных также соответствует модели клиент-сервер. Таким образом, промежуточная система функционирует одновременно и как клиент, и как сервер.

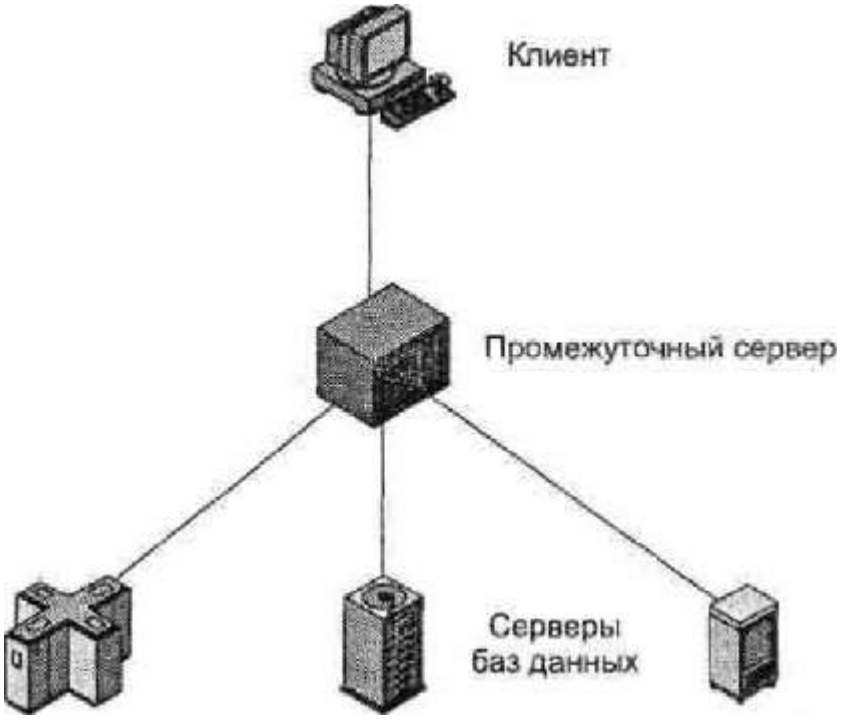

Рис. 5 - Трехзвенная архитектура клиент-сервер

1.8. Промежуточное программное обеспечение

Процессы разработки и развертывания программных продуктов, относящихся к архитектуре клиент-сервер, значительно опередили процессы стандартизации всех аспектов распределенных вычислений от физического до прикладного уровня. Отсутствие стандартов затрудняет реализацию интегрированной конфигурации, состоящей из устройств различных производителей. Поскольку основные достоинства архитектуры клиент-сервер связаны с ее модульностью, а также с возможностью объединять различные платформы и приложения, необходимо решить вопросы совместной работы этих платформ и приложений.

Чтобы добиться максимальной эффективности применения архитектуры клиент-сервер, разработчики должны обладать набором инструментальных средств, предоставляющих единообразные средства и обеспечивающие единый стиль доступа к системным ресурсам на всех платформах. Это позволит программистам создавать приложения, которые не только одинаково выглядят на различных

персональных компьютерах и рабочих станциях, но и используют один и тот же метод доступа к данным, независимо от их расположения.[5]

Наиболее общий подход к удовлетворению этих требований заключается в использовании стандартных интерфейсов и протоколов между приложением, с одной стороны, и коммуникационным программным обеспечением и операционной системой, с другой. Эти стандартизованные интерфейсы и протоколы получили название *промежуточного программного обеспечения.* Наличие стандартных программных интерфейсов облегчает развертывание одного и того же приложения на серверах и рабочих станциях разных типов. Это, очевидно, является достоинством не только для потребителя, но и для производителя. Это связано с тем, что потребители приобретают приложения, а не серверы. Потребители только выбирают серверные продукты, на которых работают нужные им приложения. Стандартизованные протоколы необходимы для взаимодействия различных серверных интерфейсов с клиентами.

Существуют разные пакеты промежуточного программного обеспечения от очень простых до очень сложных. Всех их объединяет способность скрывать от пользователя сложности и несоответствия различных сетевых протоколов и операционных систем. Производители клиентов и серверов, как правило, предлагают на выбор ряд популярных пакетов промежуточного программного обеспечения. Таким образом, пользователь может выбрать определенную стратегию развертывания промежуточного программного обеспечения, а затем заняться приобретением оборудования от разных производителей, поддерживающих данную стратегию.[8]

На рис. 6 демонстрируется возможное применение промежуточного программного обеспечения в архитектуре клиент-сервер. Точная роль промежуточного программного обеспечения зависит от вычислительной модели клиент-сервер. Как было показано на рис. 6, существует несколько классов приложений клиент-сервер, зависящих от способа разделения функций приложения.

хорошее представление о данной архитектуре.

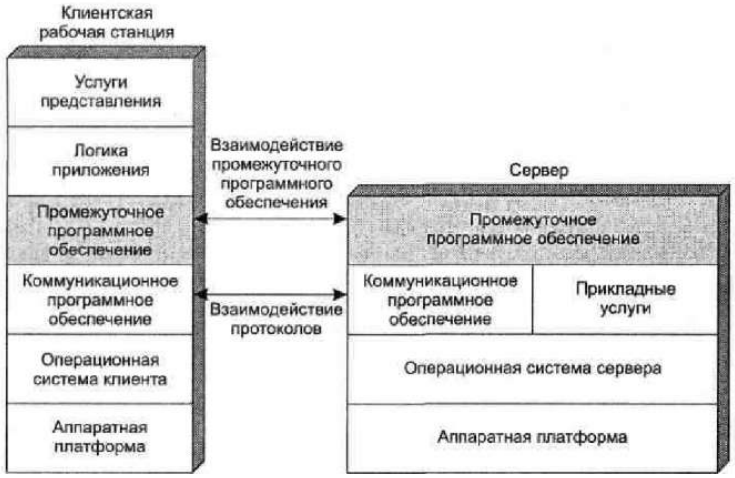

Рис. 6 - Роль промежуточного программного обеспечения в архитектуре клиент-сервер

Обратите внимание, что промежуточное программное обеспечение состоит из двух компонентов — клиентского и серверного. Основное назначение промежуточного программного обеспечения заключается в обеспечении доступа приложения или пользователя к различным услугам, предоставляемым на серверах, причем так, чтобы не беспокоиться о различиях серверов. Стандартным средством доступа к реляционной базе данных как для локального, так и для удаленного пользователя или приложения является язык структурированных запросов (SQL). Однако многие производители реляционных баз данных, хотя и поддерживают язык SQL, добавили к нему свои собственные расширения. Это, с одной стороны, позволяет производителям различать свою продукцию, но, с другойстороны, приводит к несовместимости.

Для примера рассмотрим распределенную систему, обслуживающую, помимо прочего, отдел кадров. Основные сведения о сотрудниках, такие как имена, адреса и т. д., могли бы храниться в базе данных от Gupta, тогда как информация о зарплате — в базе данных от Oracle. Когда пользователь в отделе кадров запрашивает доступ к определенным записям, он не должен думать, в которой из баз данных хранятся требуемые ему записи и кто является производителем этой базы данных. Единообразный доступ к этим системам должно предоставлять промежуточное программное обеспечение.

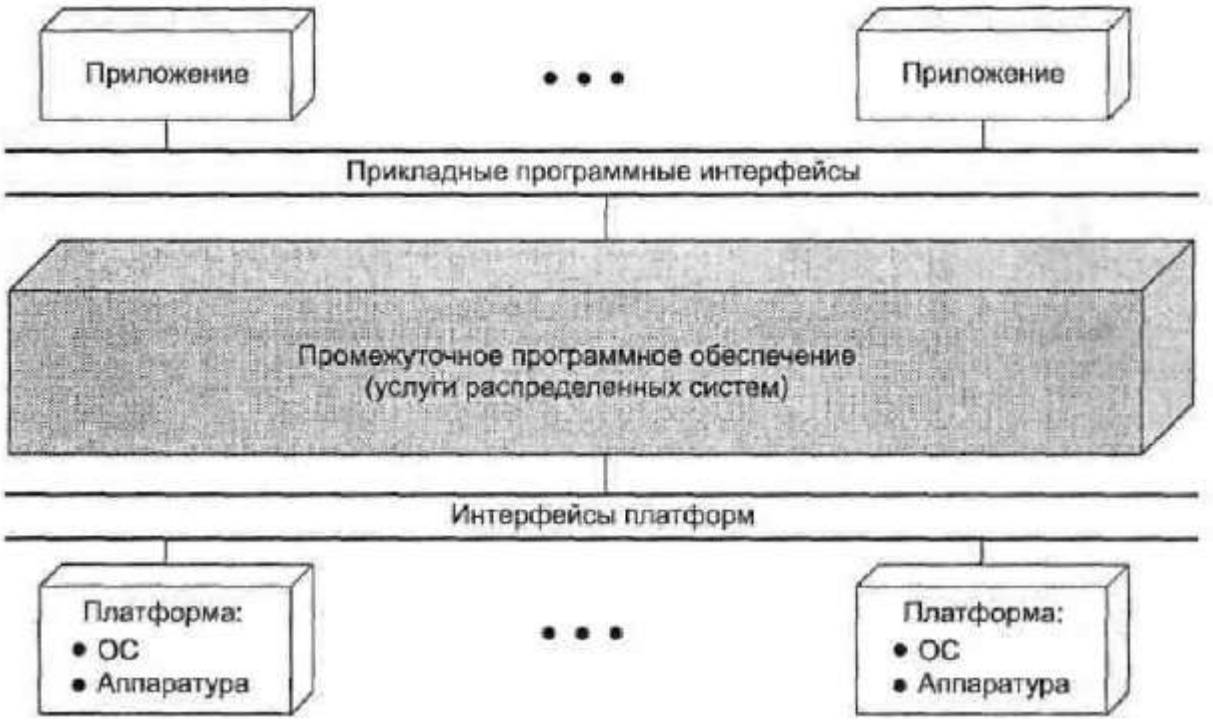

Рис. 7 - Промежуточное программное обеспечение с точки зрения логики

На роль промежуточного программного обеспечения полезно взглянуть с точки зрения логики, а не реализации (рис. 7). Вся распределенная система может рассматриваться как набор приложений и ресурсов, доступных пользователю. Пользователям не нужно беспокоиться о расположении данных или приложений. Все приложения работают поверх унифицированного *прикладного программного интерфейса* (ApplicationProgrammingInterface, API). За маршрутизацию запросов клиента к соответствующему серверу отвечает промежуточное программное обеспечение, присутствующее на всех клиентских и серверных платформах.

Передача сообщений

Распределенную передачу сообщений, реализующую функциональность архитектуры клиент-сервер, иллюстрирует на рис. 8, *а.* Клиентский процесс запрашивает некую услугу (например, прочитать или распечатать на принтере файл). Для этого он посылает сообщение с запросом данной услуги серверному процессу. Серверный процесс принимает сообщение, обрабатывает запрос и посылает ответное сообщение. В простейшем случае требуются только две функции: Send(отправить) и Receive(принять). Функции Sendна входе необходимо указать получателя, а также содержимое сообщения. Функции Receiveнужно указать, от кого ожидается сообщение (включая вариант «все») и адрес буфера, в

который следует поместить принятое сообщение.

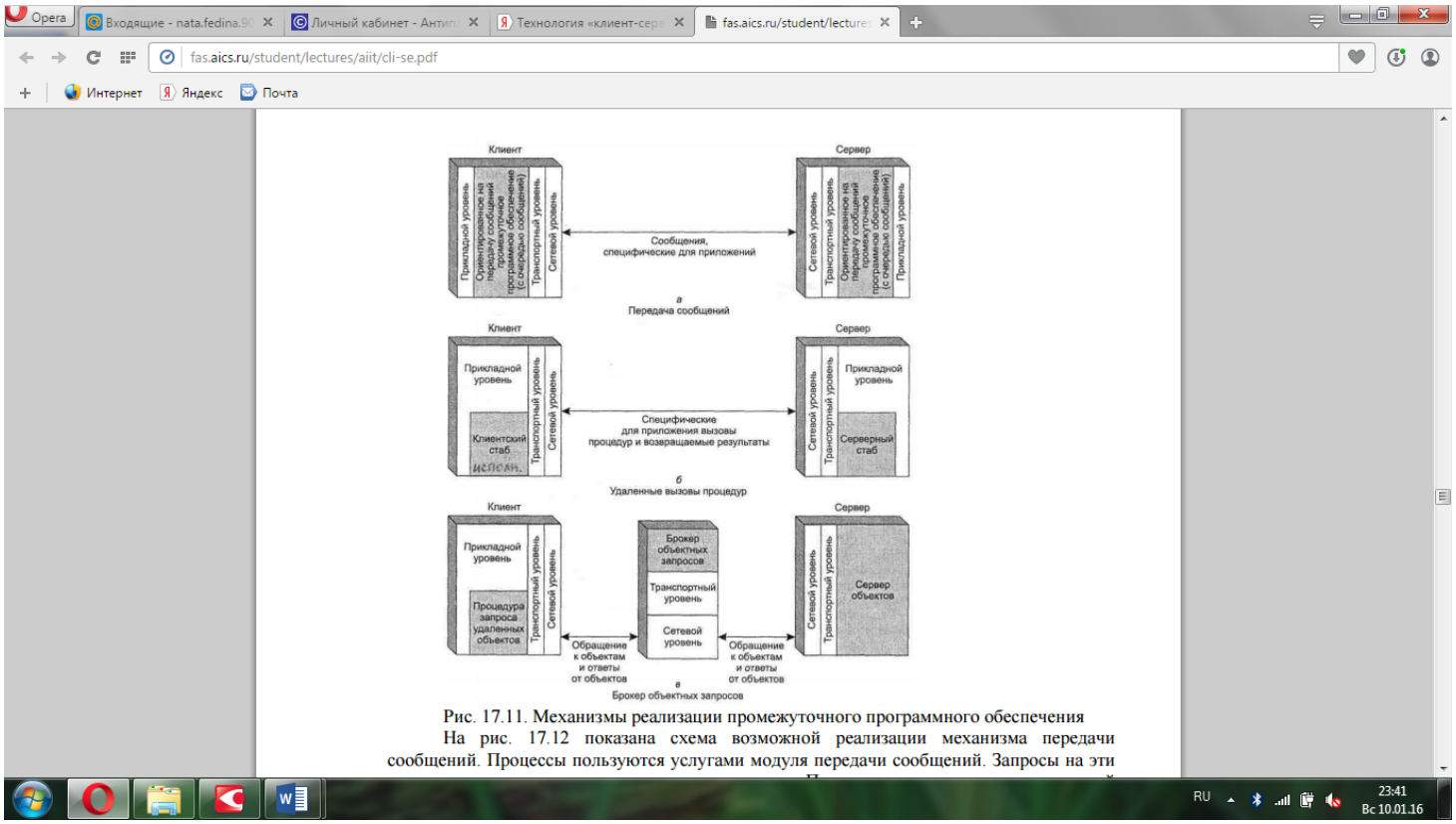

Рис. 17.11. Механизмы реализации промежуточного программного обеспечения

На рис. 17.12 показана схема возможной реализации механизма передачи сообщений. Процессы пользуются услугами модуля передачи сообщений. Запросы на эти

Рис. 8 - Механизмы реализации промежуточного программного обеспечения

На рис. 9 показана схема возможной реализации механизма передачи сообщений. Процессы пользуются услугами модуля передачи сообщений. Запросы на эти услуги могут иметь вид примитивов и параметров. Примитив соответствует исполняемой функции, в параметры используются для передачи данных и контроля над информацией.[5] Конкретная форма примитивов зависит от программного обеспечения передачи сообщений. Это может быть вызов процедуры или передача сообщения процессу операционной системы.

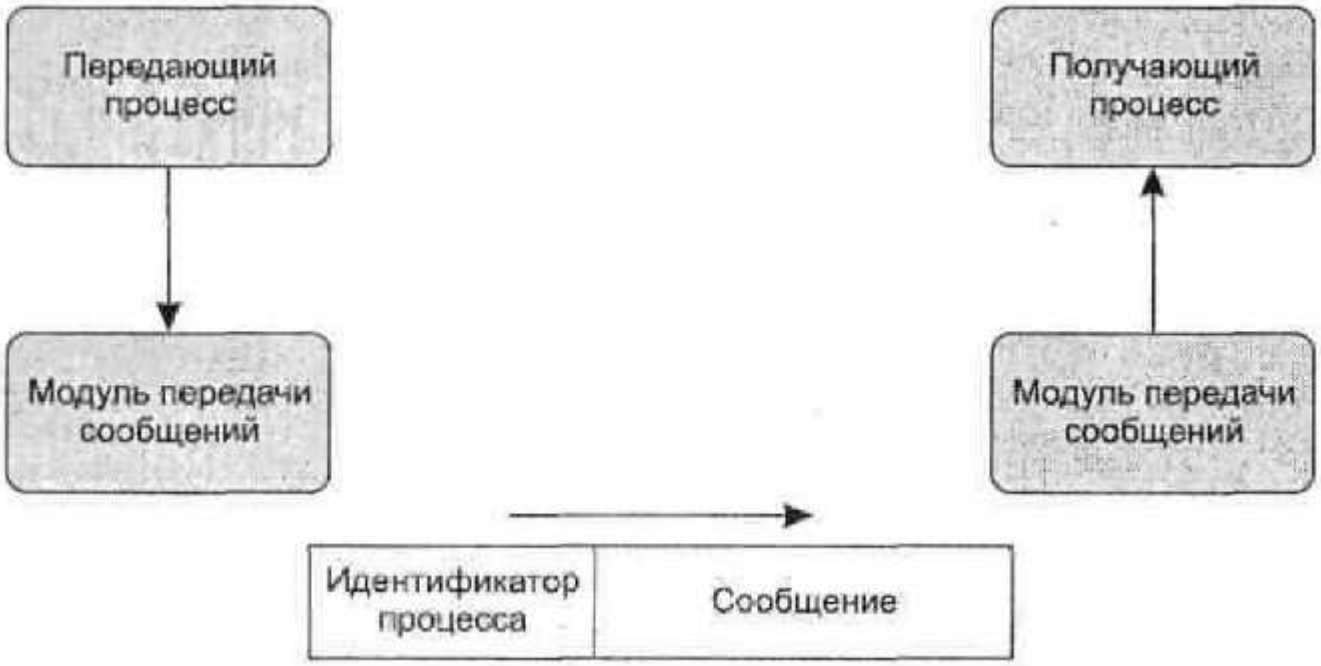

Рис. 9 - Базовые примитивы передачи сообщений

Примитив Sendиспользуется процессом, желающим отправить сообщение. Входными параметрами этого примитива являются идентификатор получающего процесса и содержимое сообщения. Модуль передачи сообщений формирует блок данных, включающий эти два элемента. Этот блок данных посылается на машину, на которой работает получающий процесс, при помощи каких-либо сетевых протоколов, например, TCP/IP. Когда адресат получает блок данных, этот блок направляется модулю передачи сообщений. Этот модуль исследует поле идентификатора процесса и сохраняет сообщениев буфере этого процесса.

В этом сценарии получающий процесс должен объявить о своем желании получать сообщения, выделить буфер для сообщений и информировать модуль передачи сообщений при помощи примитива Receive. Альтернативный метод не требует подобного объявления. Просто когда модуль передачи сообщений получает сообщение, он сигнализирует процессу, которому оно адресовано, а затем помещает сообщение в общий буфер.

Существует несколько вариантов реализации механизма распределенной передачи сообщений, о которых будет говориться далее.

Надежная передача сообщений

Надежной называется такая система передачи сообщений, которая гарантирует доставку сообщения, если такая доставка возможна. В подобной системе используется надежный транспортный протокол или сходная логика, а также должны поддерживаться контроль ошибок, квитирование, повторная передача и восстановление порядка следования сообщений. Поскольку доставка гарантирована, нет необходимости уведомлять передающий процесс о доставке сообщения. Тем не менее может быть полезно возвращать передающему процессу подтверждения, чтобы уведомить его о том, что сообщение доставлено. В любом случае, если системе не удается доставить сообщение (например, из-за неисправности сети или выходе из строя получателя), передающий процесс уведомляется о неудаче. Другую крайность представляет собой система передачи сообщений, которая просто посылает сообщения в компьютерную сеть, но не сообщает ни об успехе, ни о неудаче. Эта альтернатива позволяет значительно упростить систему передачи сообщений и накладные расходы на обработку и взаимодействие. Если приложению требуется подтверждение доставки сообщения, это может быть реализовано на прикладном уровне.

Синхронные и асинхронные системы передачи сообщений

При обращении к асинхронному (также называемому неблокирующим) примитиву, процесс не приостанавливается. Таким образом, после того как процесс вызывает примитив Send, операционная система возвращает процессу управление сразу после установки сообщения в очередь на передачу или после создания копии сообщения. Когда сообщение передано или скопировано в безопасное место для последующей передачи, передающий процесс прерывается и, таким образом, информируется о том, что буфером сообщения можно пользоваться снова. Если копии сообщения не создается, то любые изменения сообщения, производимые передающим процессом уже после обращения к примитиву Send, но до отправки сообщения, являются рискованными. Аналогично, после обращения к асинхронному примитиву Receiveпроцесс продолжает работу. Когда сообщение прибывает, процесс информируется об этом событии путем прерывания или периодического опроса.

Асинхронные примитивы обеспечивают эффективное и гибкое обращение процессов к системе передачи сообщений. Недостаток этого подхода заключается в том, что программы, использующие подобные примитивы, трудно тестировать и отлаживать. События, которые зависят от времени и которые невозможно воспроизвести, могут стать источником трудноразрешимых проблем.[4]

Альтернатива заключается в использовании синхронных или, как их еще называют, блокирующих примитивов. При вызове синхронного примитива Sendуправление не возвращается передающему процессу до тех пор, пока сообщение не будет передано (ненадежное обслуживание), или до тех пор, пока не будет получено подтверждение о доставке сообщения (надежное обслуживание). Блокирующий примитив Receivene возвратит управление, пока сообщение не окажется в выделенном для него буфере.

Уделенные вызовы процедур

Базовые сведения клиент-сервер

Уделенный вызов процедуры (RemoteProcedureCall, RPC) представляет собой вариант базовой модели передачи сообщения. Сегодня уделенные вызовы процедур являются общим и широко применяемым методом инкапсуляции взаимодействия в распределенной системе. Суть этой техники состоит в том, чтобы позволить программам на разных машинах взаимодействовать друг с другом путем простого вызова процедур, как если бы они работали на одной машине. Таким образом, механизм вызова процедур используется для доступа к услугам, предлагаемым удаленной машиной. Популярность этого подхода связана со следующими преимуществами.

- Вызов процедуры представляет собой широко распространенную и понятную абстракцию.
- Уделенные вызовы процедур позволяют специфицировать удаленный интерфейс в виде множества именованных операций с объектами указанных типов. Таким образом, интерфейс может быть четко и ясно документирован, а в распределенной программе можно выполнить статический контроль типов.
- Поскольку интерфейс стандартизован и точно определен, коммуникационная программа приложения может быть сгенерирована автоматически.
- Поскольку интерфейс стандартизован и точно определен, разработчики могут написать клиентский и серверный модули, для перемещения которых на другие платформы и операционные системы потребуется лишь небольшая модификация исходного текста программы.

Механизм удаленного вызова процедур может рассматриваться как усовершенствованная система надежной синхронной передачи сообщений.

Вызывающая программа выполняет на своей машине обычный вызов процедуры с параметрами. Например:

CALL P(X. Y) Здесь

- P — имя процедуры;
- X — передаваемые аргументы;
- Y — возвращаемые значения.

То, что на самом деле происходит удаленный вызов процедуры на какой-то другой машине, может быть прозрачным или непрозрачным для пользователя. Так называемый исполнитель процедуры P, или стаб, должен быть включен в адресное пространство вызывающего процесса или динамически скомпонован во время вызова. Стаб создает сообщение, идентифицирующее вызываемую процедуру и содержащее ее параметры. Затем он посылает это сообщение удаленной системе и ждет ответа. Когда ответ получен, стаб возвращает управление вызвавшей ее программе и передает ей возвращаемые значения.

На удаленной машине с вызываемой процедурой ассоциируется другой стаб. Когда приходит сообщение, стаб исследует его и на основе полученных имени процедуры и параметров формирует обычное локальное обращение CALLP(X, Y). То есть удаленная процедура вызывается локально, при этом выполняется стандартная передача параметров через стек.[4]

1.9. Привязка клиента и сервера

Привязка позволяет специфицировать отношения между удаленной процедурой и вызывающей ее программой. Привязка формируется после того, как два приложения установили логическое соединение и готовы обмениваться командами и данными.

Привязка может быть постоянной и непостоянной. При непостоянной привязке логическое соединение между двумя процессами устанавливается только на время удаленного вызова процедуры. Как только удаленная процедура возвращает значения, соединение разрывается. Активное соединение потребляет ресурсы, так как обе стороны должны хранить информацию о его состоянии. Использование временных соединений позволяет сберечь эти ресурсы. С другой стороны, на установку соединения требуется время и обмен служебными данными по сети, поэтому данный подход неприемлем для удаленных процедур, часто вызываемых одним и тем же процессом.

При постоянной привязке соединение, устанавливаемое для удаленного вызова процедуры, сохраняется и после того, как удаленная процедура возвращает управление. Это соединение может использоваться для последующих удаленных вызовов процедуры. Если в течение определенного интервала соединение не используется, оно разрывается. Такая схема удобна для приложений, делающих много удаленных вызовов процедур. В этом случае постоянная привязка позволяет делать вызовы и получать их результаты через одно и то же логическое соединение.

Объектно-ориентированные механизмы

После того как объектно-ориентированные технологии стали доминирующими в системном программировании, разработчики приложений клиент-сервер также начали применять этот подход. В этом случае объекты на клиентах и серверах обмениваются сообщениями. Взаимодействие между объектами может быть основано на обмене сообщениями, на удаленном вызове процедур или непосредственно на объектно- ориентированных возможностях операционной системы.

Клиент, которому требуется услуга, посылает запрос брокеру объектных запросов, действующему как каталог всех доступных в сети удаленных услуг. Брокер вызывает соответствующий объект и передает ему необходимые данные. Затем удаленный объект обслуживает запрос и отвечает брокеру, который возвращает ответ клиенту.

Успех объектно-ориентированного подхода зависит от того, насколько хорошо стандартизирован объектный механизм. К сожалению, в этой области сосуществуют сразу несколько конкурирующих схем. Одной из них является модель COM (ComponentObjectModel— модель компонентных объектов) компании Microsoft, являющаяся основой технологии OLE (ObjectLinkingandEmbedding— связывание и внедрение объектов).

Этот метод получил поддержку со стороны компании DigitalEquipmentCorporation, разработавшей механизм COM для операционной системы UNIX. С этим подходом конкурирует получившая широкую промышленную поддержку технология CORBA (CommonObjectRequestBrokerArchitecture— обобщенная архитектура брокера объектных запросов), разработанная компанией ObjectManagementGroup. Архитектуру CORBAподдерживают компании IBM, Apple, Sun, а также многие другие.[2]

Заключение

Клиент-сервер (англ. Client-server) - вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг (сервисов), называемых серверами, и заказчиками услуг, называемых клиентами. Нередко клиенты и серверы взаимодействуют через компьютерную сеть и могут быть как различными физическими устройствами, так и программным обеспечением.

Преимущества

Делает возможным, в большинстве случаев, распределить функции вычислительной системы между несколькими независимыми компьютерами в сети. Это позволяет упростить обслуживание вычислительной системы. В частности, замена, ремонт, модернизация или перемещение сервера, не затрагивают клиентов.

Все данные хранятся на сервере, который, как правило, защищён гораздо лучше большинства клиентов. На сервере проще обеспечить контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа.

Позволяет объединить различные клиенты. Использовать ресурсы одного сервера часто могут клиенты с разными аппаратными платформами, операционными системами и т.п.

Недостатки

Неработоспособность сервера может сделать неработоспособной всю вычислительную сеть.

Поддержка работы данной системы, требует отдельного специалиста - системного администратора.

Высокая стоимость оборудования.

Список использованной литературы

1. Олифер В. Г., Олифер Н. А. Компьютерные сети. Принципы, технологии, протоколы:Учебник для вузов. 4-е изд. – СПб.: Питер, 2010. – 944 с.: ил.
2. Э. Таненбаум. Компьютерные сети. 4-е изд. – СПб.: Питер, 2003. – 992 с.: ил.
3. Сетевые операционные системы/ В.Г.Олифер, Н.А.Олифер. – СПб.: "Питер", 2001. –544с.: илл.
4. Дж. Уолрэнд. Телекоммуникационные и компьютерные сети. Вводный курс. М.: Постмаркет, 2001. – 480с.
5. http://www.intuit.ru/studies/courses/2195/55/lecture/1620?page=2
6. http://edu.dvgups.ru/METDOC/GDTRAN/YAT/ITIS/PROEK_INF_SIS/METOD/UMK_DO/frame/UM
7. http://www.intuit.ru/studies/courses/2298/598/lecture/128567.
8. http://www.pmmagazine.ru
9. http://www.pmi.org/