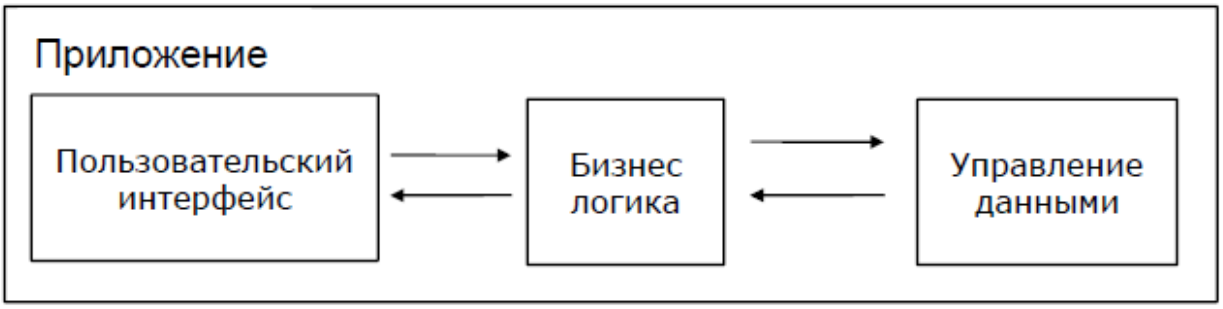
Лекция 3. Архитектура ИС.

Аннотация: Архитектура информационных систем.

Базовые функции информационных систем.

Архитектура информационной системы - концепция, определяющая модель, структуру, выполняемые функции и взаимосвязь компонентов информационной системы. (Глоссарий)

С точки зрения **программно-аппаратной реализации** можно выделить ряд типовых архитектур ИС.



Компоненты информационной системы по выполняемым функциям можно разделить на три слоя: слой представления, слой бизнес-логики и слой доступа к данным.

Слой представления - все, что связано с взаимодействием с пользователем: нажатие кнопок, движение мыши, отрисовка изображения, вывод результатов поиска и т.д.

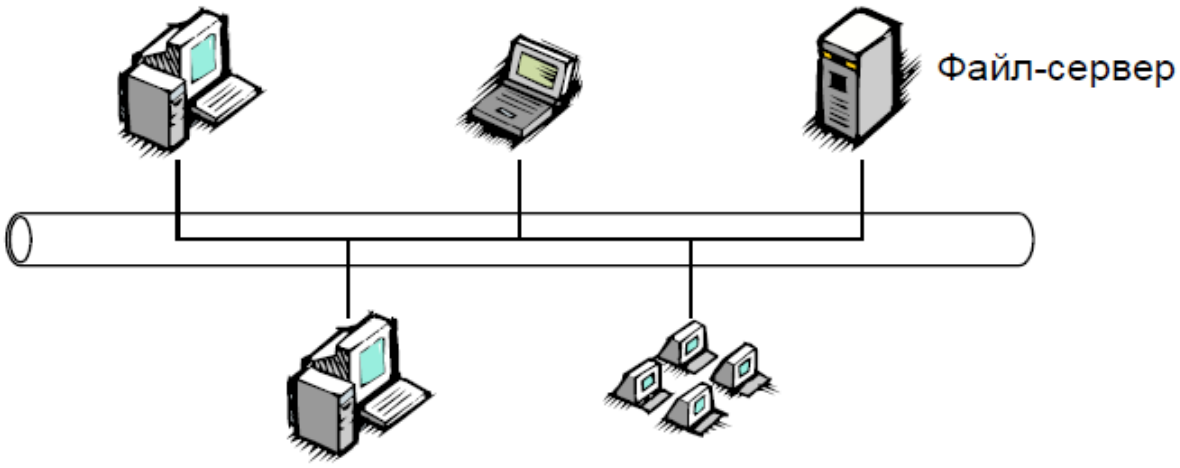
Бизнес логика - правила, алгоритмы реакции приложения на действия пользователя или на внутренние события, правила обработки данных.

Слой доступа к данным - хранение, выборка, модификация и удаление данных, связанных с решаемой приложением прикладной задачей

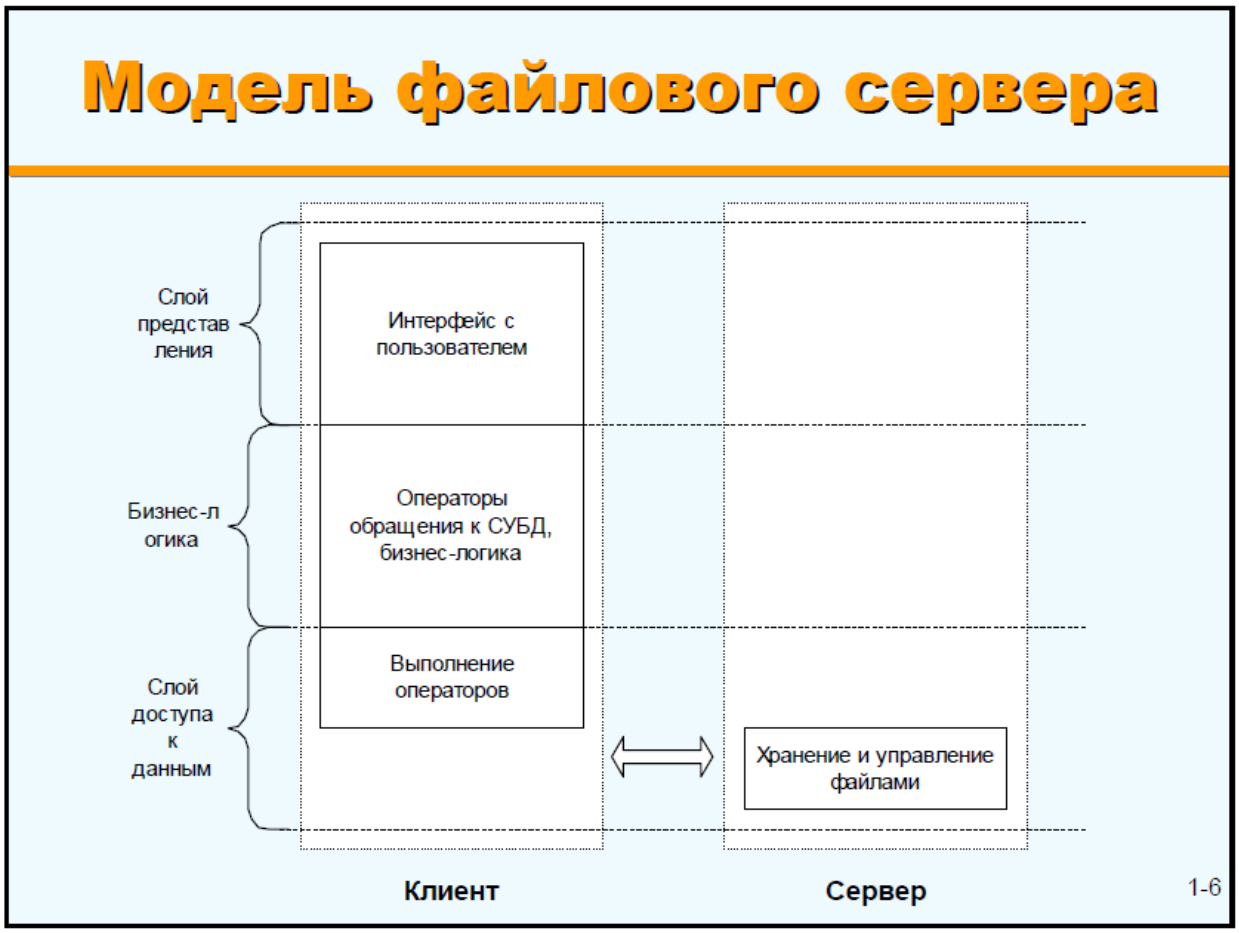
Традиционные архитектуры информационных систем.

Файл-серверная архитектура

Появились локальные сети. Файлы начали передаваться по сети. Сначала были одноранговые сети - все компьютеры равноправны.



Потом возникла идея хранения всех общедоступных файлов на выделенном компьютере в сети - файл-сервере.



Файл-серверные приложения — приложения, схожие по своей структуре с локальными приложениями и использующие сетевой ресурс для хранения программы и данных. Функции сервера: хранения данных и кода программы. Функции клиента: обработка данных происходит исключительно на стороне клиента.

Количество клиентов ограничено десятками.

Плюсы:

1. Многопользовательский режим работы с данными;
2. Удобство централизованного управления доступом;
3. Низкая стоимость разработки;

Минусы:

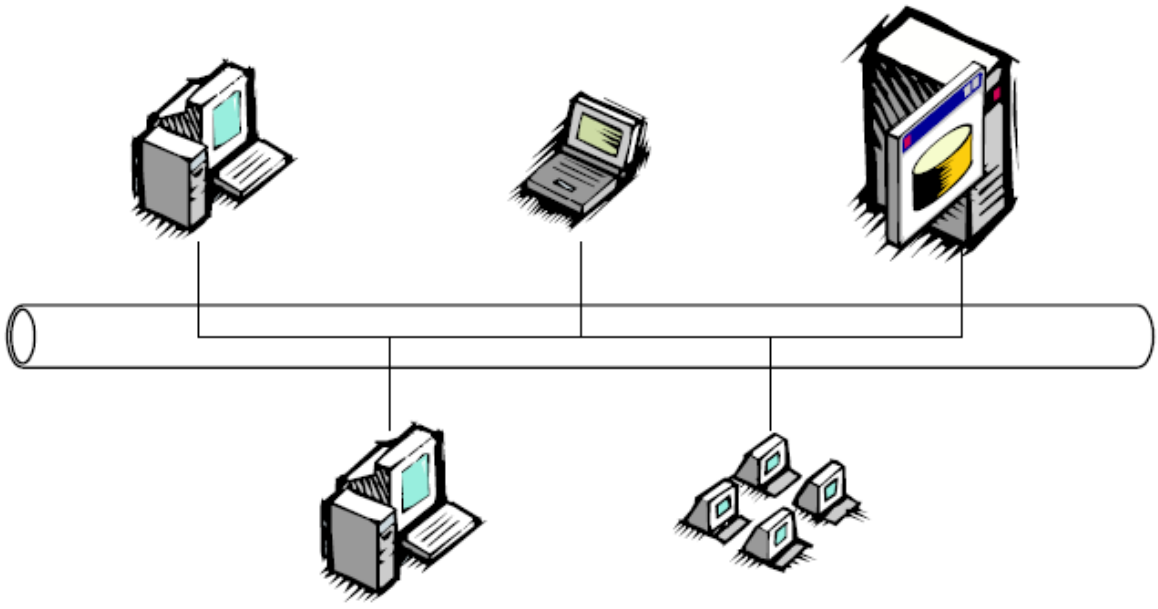
1. Низкая производительность;
2. Низкая надежность;
3. Слабые возможности расширения;

Недостатки архитектуры с файловым сервером очевидны и вытекают главным образом из того, что данные хранятся в одном месте, а обрабатываются в другом. Это означает, что их нужно передавать по сети, что приводит к очень высоким нагрузкам на сеть и, вследствие этого, резкому снижению производительности приложения при увеличении числа одновременно работающих клиентов. Вторым важным недостатком такой архитектуры является децентрализованное решение проблем целостности и согласованности данных и одновременного доступа к данным. Такое решение снижает надежность приложения.

Клиент-серверная архитектура

Ключевым отличием архитектуры клиент-сервер от архитектуры файл-сервер является абстрагирование от внутреннего представления данных (физической схемы данных). Теперь клиентские программы манипулируют данными на уровне логической схемы.

Итак, использование архитектуры клиент-сервер позволило создавать надежные (в смысле целостности данных) многопользовательские ИС с централизованной базой данных, независимые от аппаратной (а часто и программной) части сервера БД и поддерживающие графический интерфейс пользователя (ГИП) на клиентских станциях, связанных локальной сетью. Причем издержки на разработку приложений существенно сокращались.



Основные особенности:

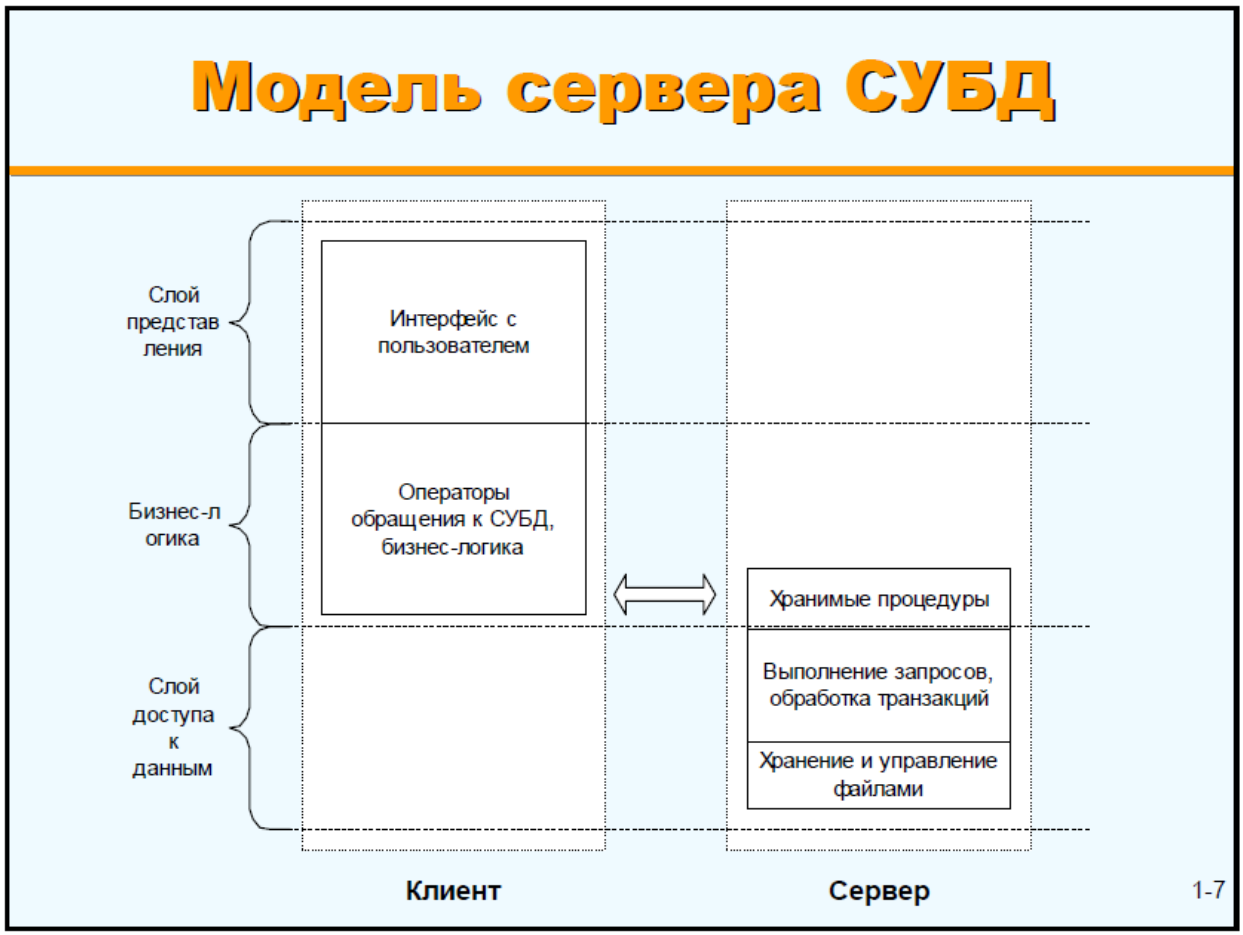
- Клиентская программа работает с данными через запросы к серверному ПО.
- Базовые функции приложения разделены между клиентом и сервером.

Плюсы:

- Полная поддержка многопользовательской работы
- Гарантия целостности данных

Минусы:

- Бизнес логика приложений осталась в клиентском ПО. При любом изменении алгоритмов, надо обновлять пользовательское ПО на каждом клиенте.
- Высокие требования к пропускной способности коммуникационных каналов с сервером, что препятствует использование клиентских станций иначе как в локальной сети.
- Слабая защита данных от взлома, в особенности от недобросовестных пользователей системы.
- Высокая сложность администрирования и настройки рабочих мест пользователей системы.
- Необходимость использовать мощные ПК на клиентских местах.
- Высокая сложность разработки системы из-за необходимости выполнять бизнес-логику и обеспечивать пользовательский интерфейс в одной программе.



Нетрудно заметить, что большинство недостатков классической или 2-х слойной архитектуры клиент-сервер проистекают от использования клиентской станции в качестве исполнителя бизнес-логики ИС. Поэтому очевидным шагом дальнейшей эволюции архитектур ИС явилась идея "тонкого клиента", то есть разбиения алгоритмов обработки данных на части связанные с выполнением бизнес-функций и связанные с отображением информации в удобном для человека представлении. При этом на клиентской машине оставляют лишь вторую часть, связанную с первичной проверкой и отображением информации, перенося всю реальную функциональность системы на серверную часть.

Переходная к трехслойной архитектуре (2.5 слоя)

Использование хранимых процедур и вычисление данных на стороне сервера сокращают трафик, увеличивают безопасность. Клиент все равно реализует часть бизнес-логики.

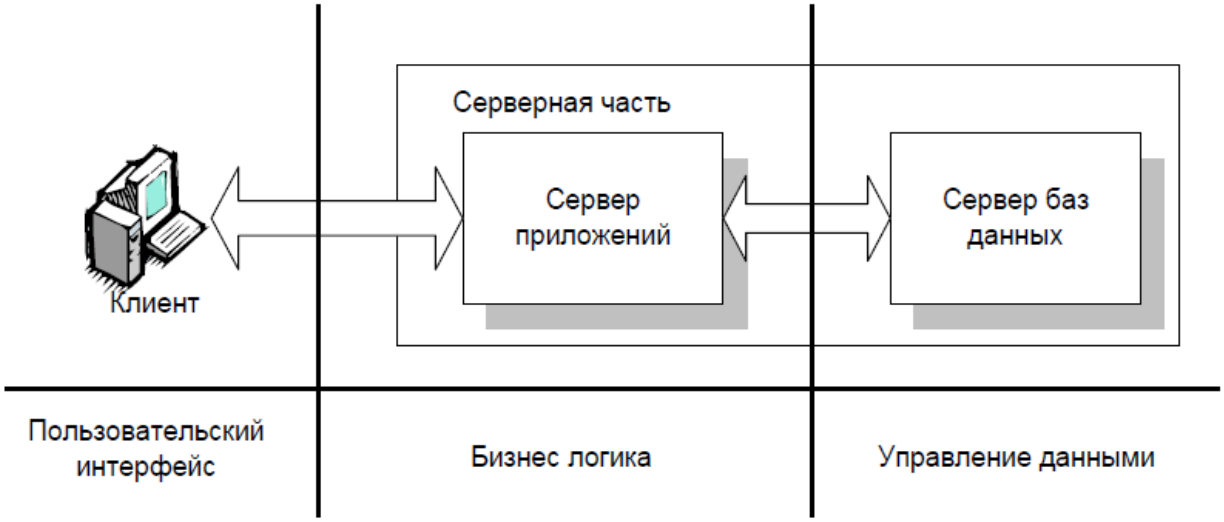
Как видно, такая организация системы весьма напоминает организацию первых унитарных систем с той лишь разницей, что на пользовательском месте стоит не терминал (с пресловутым зеленым экраном), а персональный компьютер, обеспечивающий ГИП, например, в последнее время в качестве клиентских программ часто применяют стандартные www-броузеры. Конечно, такой возврат к почти унитарным системам произошел уже на ином технологическом уровне. Обязательным стало использование СУБД со всеми их преимуществами. Программы для серверной части пишут, в основном, на специализированных языках, пользуясь механизмом хранимых процедур сервера БД. Таким образом, на уровне логической организации, ИС в архитектуре клиент-сервер с тонким клиентом расщепляется на три слоя - слой данных, слой бизнес-функций (хранимые процедуры) и слой представления. К сожалению, обычно, в такой схеме построения ИС не удается написать всю бизнес-логику приложения на не предназначенных для этого встроенных языках СУБД. Поэтому, очень часто часть бизнес-функций реализуется в клиентской части систем, которая от этого неотвратимо "толстеет". Отчасти поэтому, отчасти потому, что физически такие ИС состоят из двух компонентов, эту архитектуру часто называют 2.5-слойный клиент-сервер.

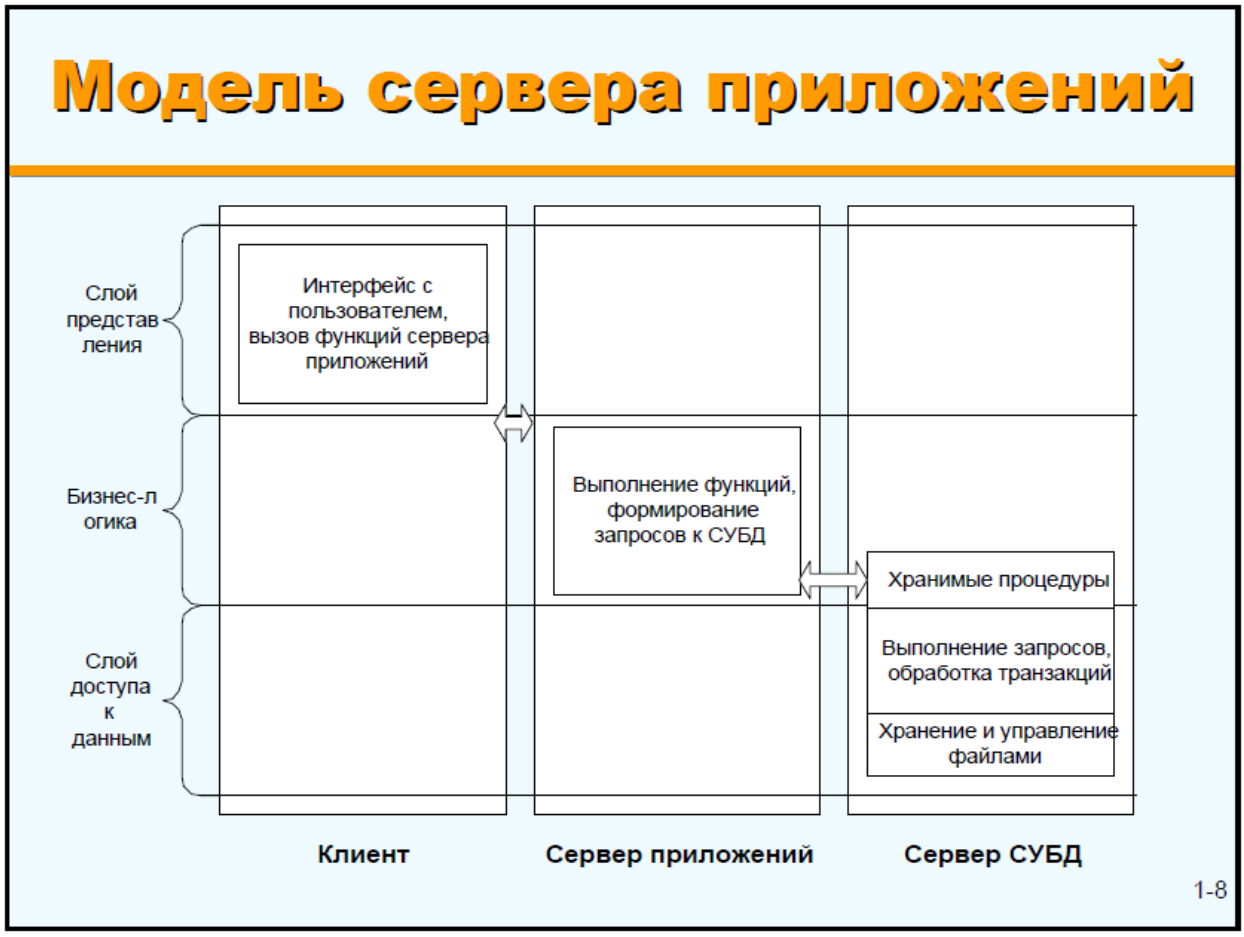
В отличие от 2-х слойной архитектуры 2.5-слойная архитектура обычно не требует наличия высокоскоростных каналов связи между клиентской и серверной частями системы, так как по сети передаются уже готовые результаты вычислений - почти все вычисления производятся на серверной стороне. Существенно улучшается также и защита информации - пользователям даются права на доступ к функциям системы, а не на доступ к ее данным и т.д. Однако вместе с преимуществами унитарного подхода архитектура 2.5 перенимает и все его недостатки, как-то: ограниченную масштабируемость, зависимость от программной платформы, ограниченное использование сетевых вычислительных ресурсов. Кроме того программы для серверной части системы пишутся на встроенных в СУБД языках описания хранимых процедур, предназначенных для валидации данных и построения несложных отчетов, а вовсе не для написания ИС масштаба предприятия. Все это снижает быстродействие системы, повышает трудоемкость создания с модификации ИС и самым негативным образом сказывается на стоимости аппаратных средств, необходимых для ее функционирования.

Трехуровневая клиент-серверная архитектура

Для решения этих проблем и была предложена так называемая 3-х слойная архитектура клиент-сервер. Основным ее отличием от архитектуры 2.5 является **физическое** разделение программ, отвечающих за хранение данных (СУБД) от программ эти данные

обрабатывающих (сервер приложения (СП), application server (AS)). Такое разделение программных компонент позволяет оптимизировать нагрузки как на сетевое, так и на вычислительное оборудование комплекса.





Компоненты трёхзвенной архитектуры, с точки зрения программного обеспечения реализуют определенные сервера БД, web-сервера и браузеры. Место любого из этих компонентов может занять программное обеспечение любого производителя. Ниже представлено описание взаимодействия компонентов трехуровневой архитектуры клиент-серверного приложения. Сервер БД представлен MySQL-сервером; сервер приложений

технологиями: ADO.NET, ASP.NET и web-сервером IIS; роль клиента выполняет любой web-браузер.

Браузер клиента *1*-> **Сервер IIS** *2*-> **Исполняющая среда ASP.NET 2.0** *3*-> **Провайдер данных ADO.NET 2.0** *4*-> **Сервер MySQL** *5*-> **Провайдер данных ADO.NET 2.0** *6*-> **Исполняющая среда ASP.NET 2.0** *7*-> **Сервер IIS** *8*-> Браузер клиента

- 1 — браузер клиента отправляет HTTP-запрос;
- 2 — на стороне сервера служба Web Internet Information Server (web-сервер IIS) определяет тип запрашиваемого ресурса, и для случая запроса *.aspx (расширение файлов страниц ASP.NET) загружает соответствующее ему (запросу) расширение Internet Server Aplication Programming Interface (ISAPI). Для страниц aspx это расширение isapi_aspnet.dll. IIS также осуществляет идентификацию и авторизацию пользователя от которого поступил запрос. В свою очередь расширение isapi_aspnet.dll загружает фабрику обработчиков ASP.NET. Далее, фабрика обработчиков создает объектную модель запрашиваемой страницы и обрабатывает действия пользователя.
- 3 — в ходе генерации ответа приложению ASP.NET может потребоваться обращение к БД, в этом случае используя библиотеки классов провайдера данных ADO.NET 2.0, выполняющая среда обращается к серверу БД;
- 4 — провайдер данных ADO.NET 2.0 передает запрос на операцию с БД серверу MySQL;
- 5 — сервер MySQL осуществляет обработку запроса, выполняя соответствующие операции с БД ;
- 6 — провайдер данных ADO.NET 2.0 передает результаты запроса объекту страницы;
- 7 — объект страницы с учетом полученных данных осуществляет рендеринг графического интерфейса страницы и направляет результаты в выходной поток;
- 8 — сервер IIS отправляет содержимое сгенерированной страницы клиентскому браузеру.

Плюсы:

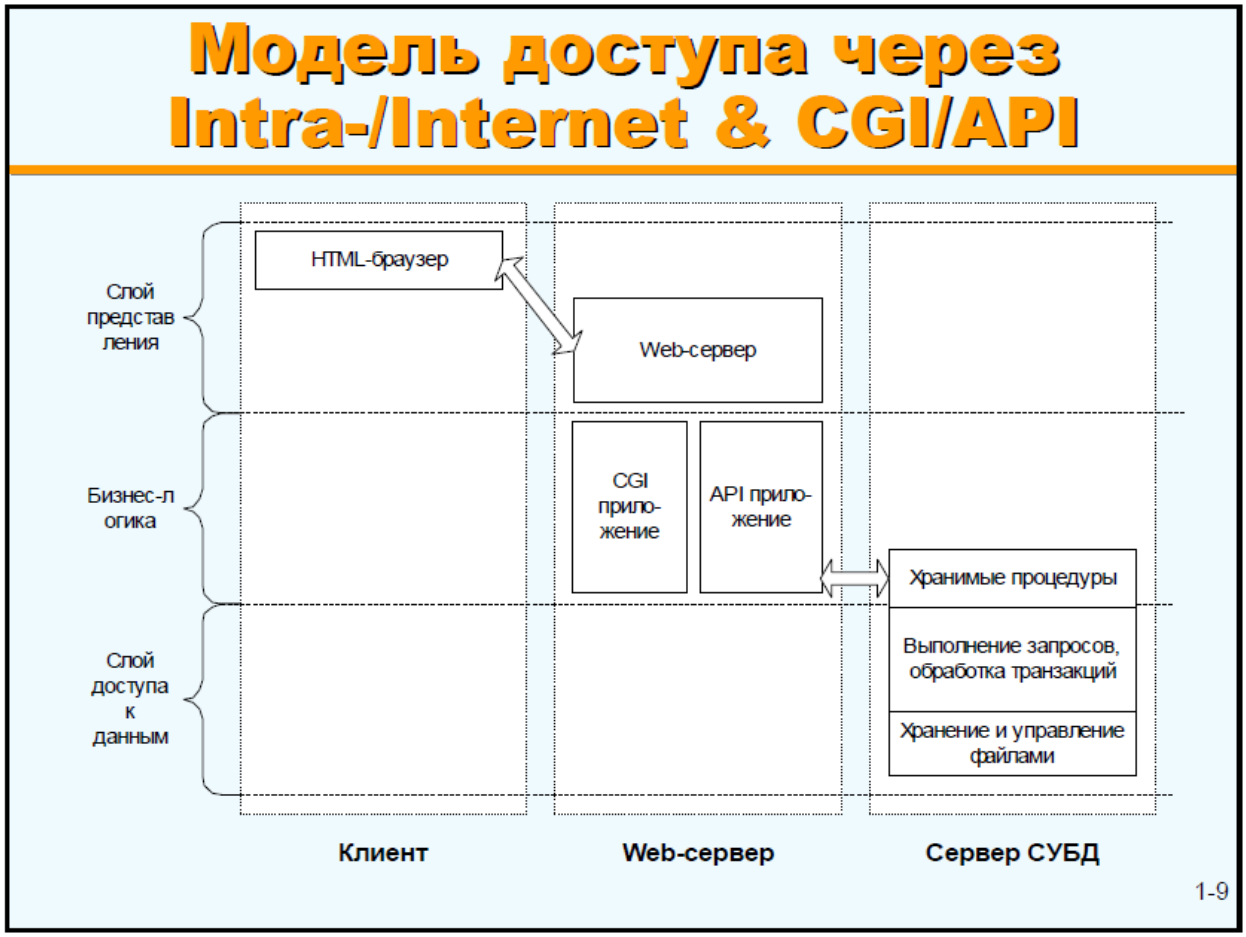
1. Тонкий клиент.
2. Между клиентской программой и сервером приложения передается лишь минимально необходимый поток данных - аргументы вызываемых функций и возвращаемые от них значения. Это теоретический предел эффективности использования линий связи, даже работа с ANSI-терминалами (не говоря уже об использование протокола http) требует большей нагрузки на сеть.
3. Сервер приложения ИС может быть запущен в одном или нескольких экземплярах на одном или нескольких компьютерах, что позволяет использовать вычислительные мощности организации столь эффективно и безопасно как этого пожелает администратор ИС.
4. Дешевый трафик между сервером приложений и СУБД. Трафик между сервером приложений и СУБД может быть большим, однако это всегда трафик локальной сети, а их пропускная способность достаточно велика и дешева. В крайнем случае, всегда можно запустить СП и СУБД на одной машине, что автоматически сведет сетевой трафик к нулю.
5. Снижение нагрузки на сервер данных по сравнению с 2.5-слойной схемой, а значит и повышение скорости работы системы в целом.
6. Дешевле наращивать функциональность и обновлять ПО.

Минусы:

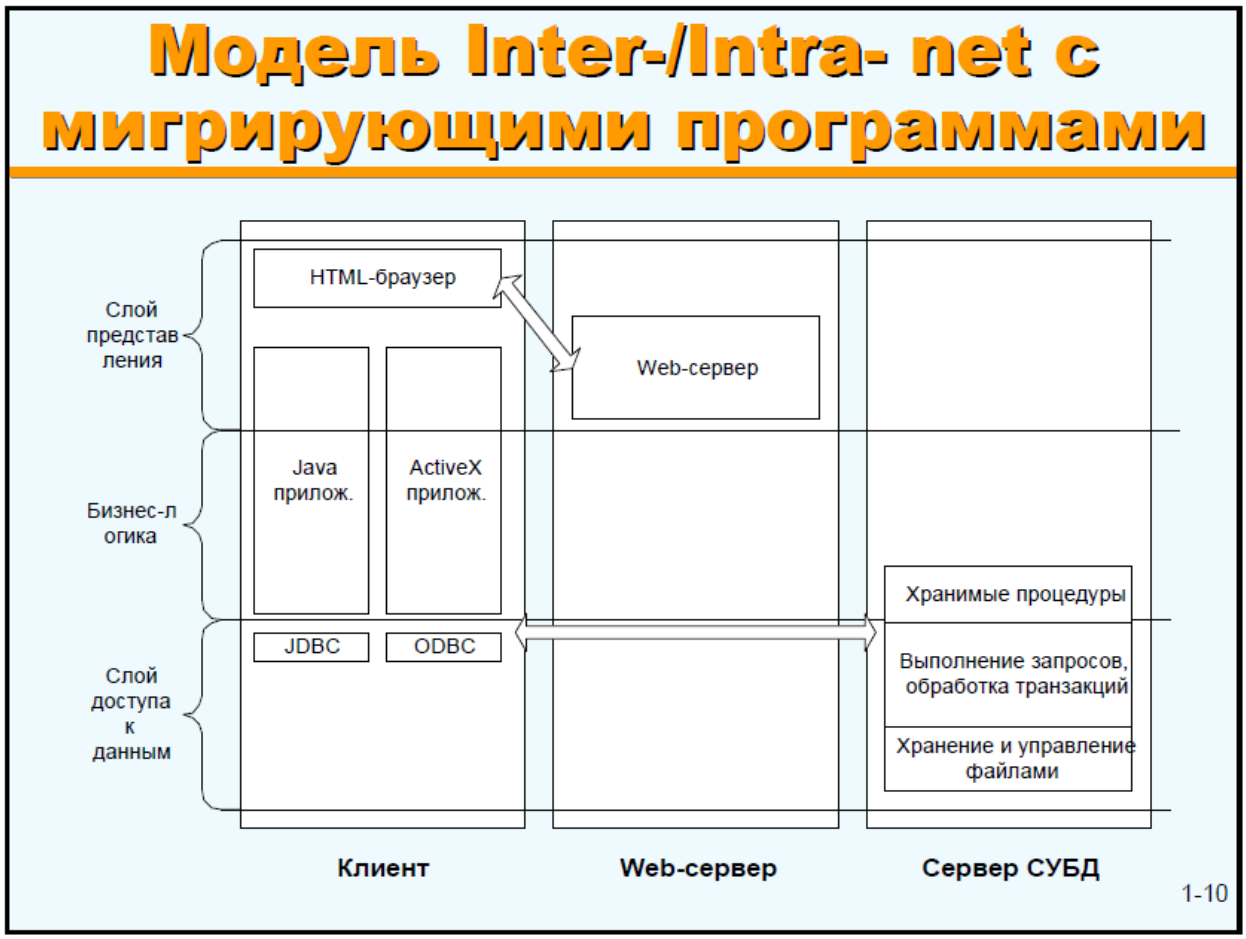
1. Выше расходы на администрирование и обслуживание серверной части.

Масштабируемость систем выполненных в 3-х слойной архитектуре впечатляет. Одна и та же система может работать как на одном отдельно стоящем компьютере, выполняя на нем программы СУБД, СП и клиентской части, так и в сети, состоящей из сотен и тысяч машин. Как уже было отмечено, единственным фактором, препятствующим бесконечной масштабируемости, является лишь требование ведения единой базы данных.

Помимо требования увеличения производительности системы с ростом масштабов деятельности важным фактором является и расширение ее функциональной наполненности. И здесь 3-х слойная схема выигрывает у своих предшественников. Для расширения функциональности не обязательно менять **всю** систему как в случае 2.5-слойной схемы - достаточно установить новый сервер приложения с требуемой функцией. Отпадают и многие проблемы связанные с переустановкой клиентских частей программы на множестве компьютеров, быть может весьма удаленных, столь актуальные для 2-слойной схемы - парадигма "тонкого" клиента предоставляет для этого целый ряд возможностей.


Модель доступа через Intra-/Internet & CGI/API
Слой представления
Бизнес-логика
Слой доступа к данным
HTML-браузер
Web-сервер
CGI прило-жение
API прило-жение
Хранимые процедуры
Выполнение запросов, обработка транзакций
Хранение и управление файлами
Клиент
Web-сервер
Сервер СУБД
1-9

Архитектура на основе Internet/Intranet с мигрирующими программами



Распределенные информационные системы.

В литературе можно найти различные определения распределенных систем, причем ни одно из них не является удовлетворительным и не согласуется с остальными.

Для наших задач хватит достаточно вольной характеристики.

Распределенная система — это набор независимых вычислительных машин, представляющийся их пользователям единой объединенной системой.

В этом определении оговариваются два момента. Первый относится к аппаратуре: все машины автономны.

Второй касается программного обеспечения: пользователи думают, что имеют дело с единой системой. Важны оба момента. Позже в этой главе мы к ним вернемся, но сначала рассмотрим некоторые базовые вопросы, касающиеся как аппаратного, так и программного обеспечения.

Характеристики распределенных систем:

1. От пользователей скрыты различия между компьютерами и способы связи между ними. То же самое относится и к внешней организации распределенных систем.

2. Пользователи и приложения единообразно работают в распределенных системах, независимо от того, где и когда происходит их взаимодействие.

Распределенные системы должны также относительно легко поддаваться расширению, или масштабированию. Эта характеристика является прямым следствием наличия независимых компьютеров, но в то же время не указывает, каким образом эти компьютеры на самом деле объединяются в единую систему.

Распределенные системы обычно существуют постоянно, однако некоторые их части могут временно выходить из строя. Пользователи и приложения не должны уведомляться о том, что части системы заменены или починены или, что добавлены новые для поддержки дополнительных пользователей.

Для того чтобы поддержать представление системы в едином виде, организация распределенных систем часто включает в себя дополнительный уровень программного обеспечения, находящийся между верхним уровнем, на котором находятся пользователи и приложения, и нижним уровнем, состоящим из операционных систем.

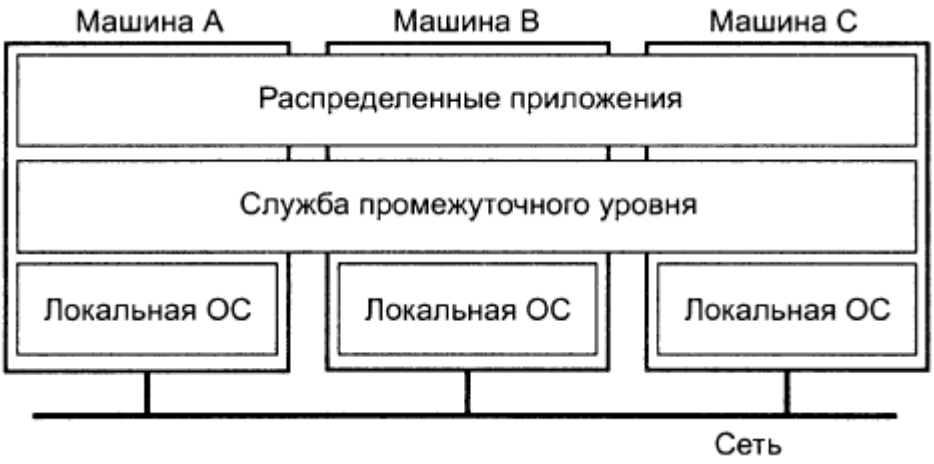


Рис. 1.1. Распределенная система организована в виде службы промежуточного уровня.

Соответственно, такая распределенная система обычно называется *системой промежуточного уровня (middleware)*. Отметим, что промежуточный уровень распределен среди множества компьютеров.

Особенности распределенных ИС

- Ссылки
- Задержки выполнения запросов
- Активация/деактивация
- Постоянное хранение
- Параллельное исполнение
- Отказы
- Безопасность

Ссылки

Ссылки на объекты в программных модулях на ОО языках программирования (например, C++) являются указателями в памяти.

1. Ссылки на объекты в распределенных системах в противоположность являются более комплексными:
   1.1.Содержат информацию о размещении
   1.2.Информацию о безопасности
   1.4.Ссылки на объектные типы
2. Ссылки на распределенные объекты значительно больше (40 байт для Orbix)

Задержки выполнения запросов

Локальные вызовы требуют порядка пары сотен наносекунд

Запрос к объекту требует от 0.1 до 10 миллисекунд

Интерфейсы в распределенной системе должны быть спроектированы так, чтобы снизить время выполнения запросов:

1. Снизить частоту обращения;
2. Укрупнить выполняемые функции.

Активация/Деактивация

Объекты в ОО языках находятся в виртуальной памяти от создания до уничтожения

В распределенных системах

1. Больше объектов
2. Объекты могут не использоваться на протяжении долгого времени

Реализации распределенных объектов

1. Переносятся в память при активации
2. Удаляются из памяти при деактивации

Постоянное хранение

Объекты могут иметь или не иметь состояние.

Объекты имеющие состояние должны сохранять его на постоянный носитель между:

1. Деактивацией объекта
2. Активацией объекта

Может быть достигнуто:

1. Записью в файловую систему
2. Отражением на реляционные БД
3. С помощью объектных БД

Параллельное исполнение

В нераспределенных системах исполнение в основном последовательное, иногда конкурентное в разных нитях процессов.

Распределенные компоненты выполняются параллельно, что приводит к необходимости согласования выполнения.

Отказы

Запросы в распределенных системах имеют большую вероятность отказов

Клиенты обязаны проверять факт выполнения запросов сервером

Безопасность

Безопасность в ОО приложениях может выполняться на основе контроля сеансов.

При работе распределенных систем возникают вопросы безопасности:

1. Кто запрашивает выполнение операции?
2. Как мы можем удостовериться, что субъект является именно тем за кого он себя выдает?
3. Как мы примем решение предоставлять или нет субъекту право на выполнение сервиса?
4. Как мы можем неопровержимо доказать, что сервис был предоставлен?

Источники

1. Бурдаков А.В. – Распределенные вычислительные системы. Лекция №1, слайд 3-5.
2. Кузнецов – Архитектуры ИС. Слайд 12, 14, 20,21,22,23, 24, 25;
3. СофтСиб – Преимущества клиент-серверной перед файл-серверной технологией. http://www.soft-sib.ru/articles/programs/10/
4. Танненбаум – Распределенные вычислительные системы. Глава 1.1
5. Бурдаков А.В. – Распределенные вычислительные системы. Лекция №3, слайд 3-5.