Распределенные информационно-аналитические системы

Лекция № 4.

Модели распределенной обработки информации

Профессор кафедры КБ-2: д.т.н. Шатовкин Р.Р.

Учебные цели:

Изучить основы научных знаний по технологии и архитектуре «клиент-сервер»; двухуровневым моделям распределенной обработки информации; модели сервера приложений; моделям серверов баз данных.

Учебные вопросы:

1. Технология и архитектура «клиент-сервер».

2. Двухуровневые модели распределенной обработки информации.

3. Модель сервера приложений.

4. Модели серверов баз данных.

1. Технология и архитектура «клиент-сервер»

Технология «клиент-сервер». Модель «клиент-сервер» исходно связана с парадигмой открытых систем, которая появилась в 90-х годах и быстро эволюционировала. Сам термин «клиент-сервер» исходно применялся к архитектуре программного обеспечения, которое описывало распределение процесса выполнения по принципу взаимодействия двух программных процессов, один из которых в этой модели назывался «клиентом», а другой – «сервером». Клиентский процесс запрашивал некоторые услуги, а серверный процесс обеспечивал их выполнение. При этом предполагалось, что один серверный процесс может обслужить множество клиентских процессов.

Ранее приложение (пользовательская программа) не разделялась на части, оно выполнялось некоторым монолитным блоком. Но возникла идея более рационального использования ресурсов сети. Действительно, при монолитном исполнении используются ресурсы только одного компьютера, а остальные компьютеры в сети рассматриваются как терминалы. Но теперь, в отличие от эпохи main-фреймов, все компьютеры в сети обладают собственными ресурсами, и разумно так распределить нагрузку на них, чтобы максимальным образом использовать их ресурсы.

И как в промышленности, здесь возникает древняя как мир идея распределения обязанностей, разделения труда. Конвейеры Форда сделали в свое время прорыв в автомобильной промышленности, показав наивысшую производительность труда именно из-за того, что весь процесс сборки был разбит на мелкие и максимально простые операции и каждый рабочий специализировался на выполнении только одной операции, но эту операцию он выполнял максимально быстро и качественно.

Конечно, в вычислительной технике нельзя было напрямую использовать технологию автомобильного или любого другого механического производства, но идею использовать было можно. Однако для воплощения идеи необходимо было разработать модель разбиения единого монолитного приложения на отдельные части и определить принципы взаимосвязи между этими частями.

Основной принцип технологии «клиент-сервер» заключается в разделении функций стандартного интерактивного приложения на 5 групп, имеющих различную природу:

- функции ввода и отображения данных (Presentation Logic);
- прикладные функции, определяющие основные алгоритмы решения задач приложения (Business Logic);
- функции обработки данных внутри приложения (Database Logic);
- функции управления информационными ресурсами (Database Manager System);
- служебные функции, играющие роль связок между функциями первых четырех групп.

Структура типового приложения, работающего с базой данных, приведена на рисунке 1.

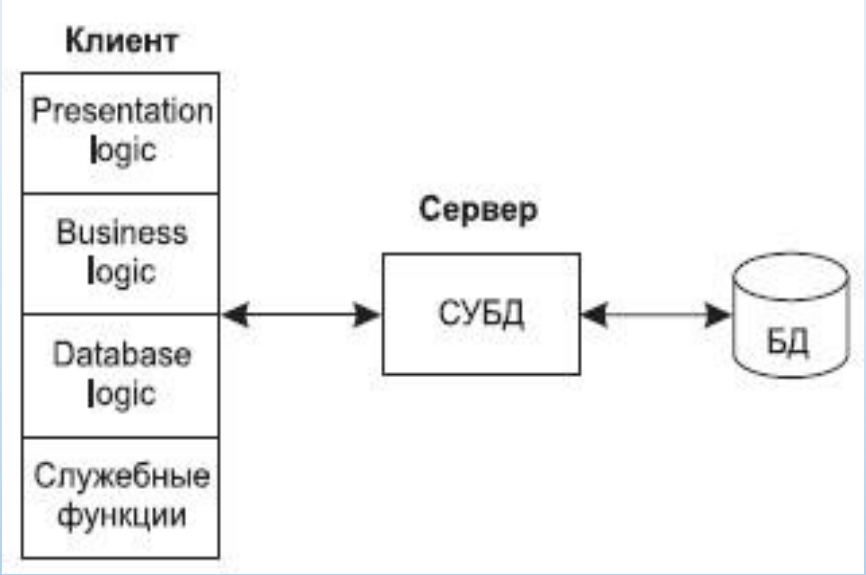

Рисунок 1 – Структура типового интерактивного приложения, работающего с базой данных

Презентационная логика (Presentation Logic) как часть приложения определяется тем, что пользователь видит на своем экране, когда работает приложение. Сюда относятся все интерфейсные экранные формы, которые пользователь видит или заполняет в ходе работы приложения, к этой же части относится все то, что выводится пользователю на экран как результаты решения некоторых промежуточных задач либо как справочная информация.

Поэтому ***основными задачами презентационной логики являются***:

- формирование экранных изображений;
- чтение и запись в экранные формы информации;
- управление экраном;
- обработка движений мыши и нажатие клавиш клавиатуры.

Некоторые возможности для организации презентационной логики приложений предоставляет знако-ориентированный пользовательский интерфейс, задаваемый моделями CCIS (Customer Control Information System) и IMS/DC фирмы IBM и моделью TSO (Time Sharing Option) для централизованной main-фреймовой архитектуры.

Модель GUI – графического пользовательского интерфейса, поддерживается в операционных средах Microsoft's Windows, Windows NT, в OS/2 Presentation Manager, X-Windows и OSF/Motif.

Бизнес-логика, или логика собственно приложений (Business processing Logic), – это часть кода приложения, которая определяет собственно алгоритмы решения конкретных задач приложения. Обычно этот код пишется с использованием различных языков программирования, таких как C, C++, Cobol, SmallTalk, Visual-Basic.

Логика обработки данных (Data manipulation Logic) – это часть кода приложения, которая связана с обработкой данных внутри приложения. Данными управляет собственно СУБД (DBMS). Для обеспечения доступа к данным используются язык запросов и средства манипулирования данными стандартного языка SQL.

Обычно операторы языка SQL встраиваются в языки 3-го или 4-го поколения (3GL, 4GL), которые используются для написания кода приложения.

Процессор управления данными (Database Manager System Processing) – это собственно СУБД, которая обеспечивает хранение и управление базами данных. В идеале функции СУБД должны быть скрыты от бизнес-логики приложения, однако для рассмотрения архитектуры приложения нам надо их выделить в отдельную часть приложения.

В централизованной архитектуре (Host-based processing) эти части приложения располагаются в единой среде и комбинируются внутри одной исполняемой программы.

В децентрализованной архитектуре эти задачи могут быть по-разному распределены между серверным и клиентским процессами. В зависимости от характера распределения можно выделить следующие модели распределений (рисунок 2):

- распределенная презентация (Distribution presentation, DP);
- удаленная презентация (Remote Presentation, RP);
- распределенная бизнес-логика (Distributed Business Logic, DBL);
- распределенное управление данными (Distributed data management, DDM);
- удаленное управление данными (Remote data management, RDM).

Эта условная классификация показывает, как могут быть распределены отдельные задачи между серверным и клиентскими процессами. В этой классификации отсутствует реализация удаленной бизнес-логики. Действительно, считается, что она не может быть удалена сама по себе полностью. Считается, что она может быть распределена между разными процессами, которые в общем-то могут выполняться на разных платформах, но должны корректно кооперироваться (взаимодействовать) друг с другом.

Рисунок 2 – Распределение функций приложения в моделях «клиент-сервер»

Архитектура «клиент-сервер». Архитектура «клиент-сервер» (также используются термины «сеть клиент-сервер» или «модель клиент-сервер») предусматривает разделение процессов предоставление услуг и отправки запросов на них на разных компьютерах в сети, каждый из которых выполняют свои задачи независимо от других (рисунок 3).

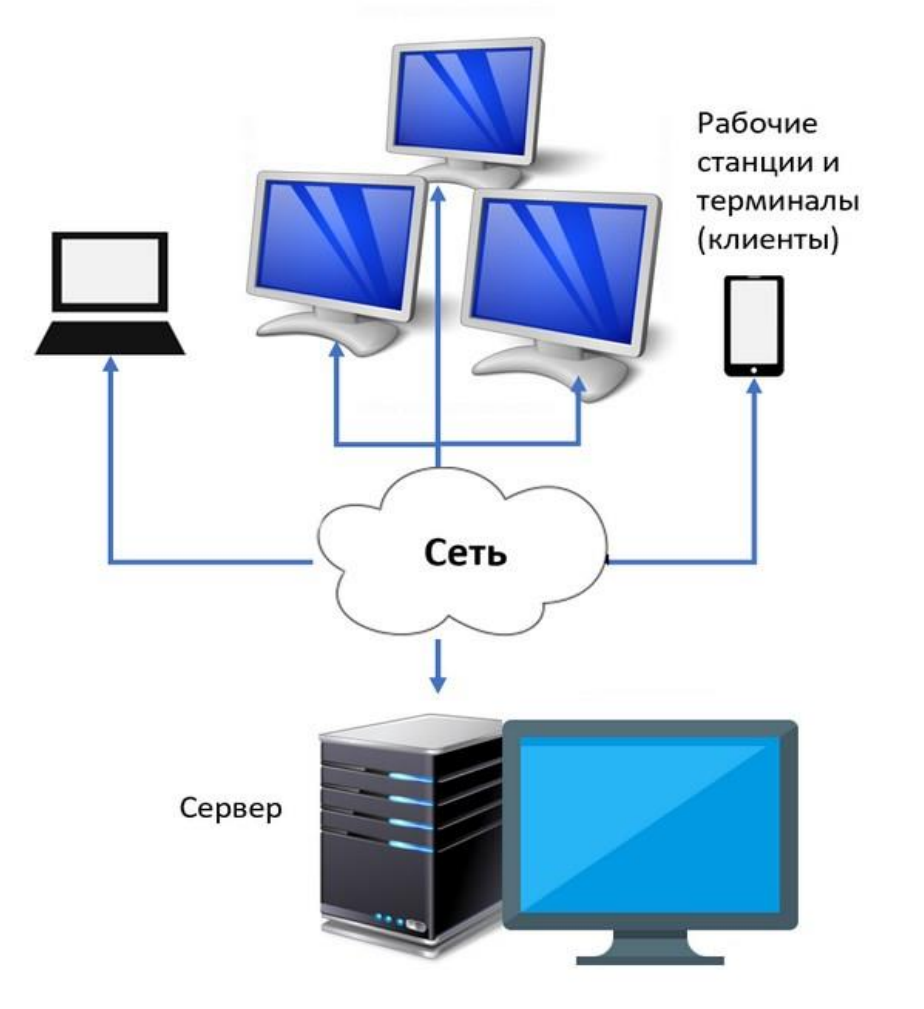

Рисунок 3 – Архитектура «клиент-сервер»

В архитектуре «клиент-сервер» несколько компьютеров-клиентов (удаленные системы) посылают запросы и получают услуги от централизованной служебной машины – сервера (server – англ. «официант, обслуга»), которая также может называться хост-системой (host system, от host – англ. «хозяин», обычно гостиницы).

Клиентская машина предоставляет пользователю так называемый «дружественный интерфейс» (user-friendly interface), чтобы облегчить его взаимодействие с сервером.

Типы клиент-серверной архитектуры. Архитектуру «клиент-сервер» принято разделять на три класса: одно-, двух- и трехуровневую. Однако, нельзя сказать, что в вопросе о таком разделении в сообществе IT-специалистов существует полный консенсус. Многие называют одноуровневую архитектуру двухуровневой и наоборот, то же можно сказать о соотношении двух- и трехуровневой архитектур.

Одноуровневая архитектура «клиент-сервер» (1-Tier) – такая, где все прикладные программы рассредоточены по рабочим станциям, которые обращаются к общему серверу баз данных или к общему файловому серверу. Никаких прикладных программ сервер при этом не исполняет, только предоставляет данные.

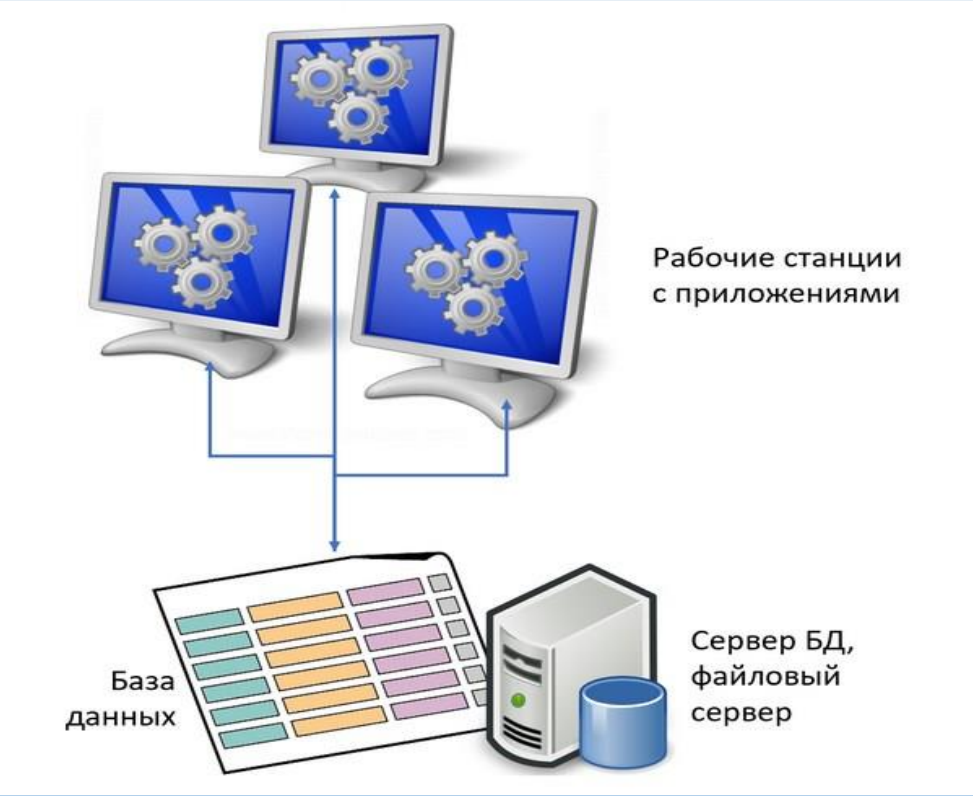

Рисунок 4 – Одноуровневая архитектура «клиент-сервер» (1-Tier)

В целом, такая архитектура очень надежна, однако, ей сложно управлять, поскольку в каждой рабочей станции данные будут присутствовать в разных вариантах. Поэтому возникает проблема их синхронизации на отдельных машинах. В общем, как можно видеть из рисунка, в этой архитектуре просматривается еще один уровень – базы данных, что дает повод во многих случаях называть ее двухуровневой.

Двухуровневая архитектура (2-Tier). К двухуровневой архитектуре «клиент-сервер» следует относить такую, в которой прикладные программы сосредоточены на сервере приложений (Application Server), например, сервере 1С или сервере CRM, а в рабочих станциях находятся программы-клиенты, которые предоставляют для пользователей интерфейс для работы с приложениями на общем сервере (рисунок 5).

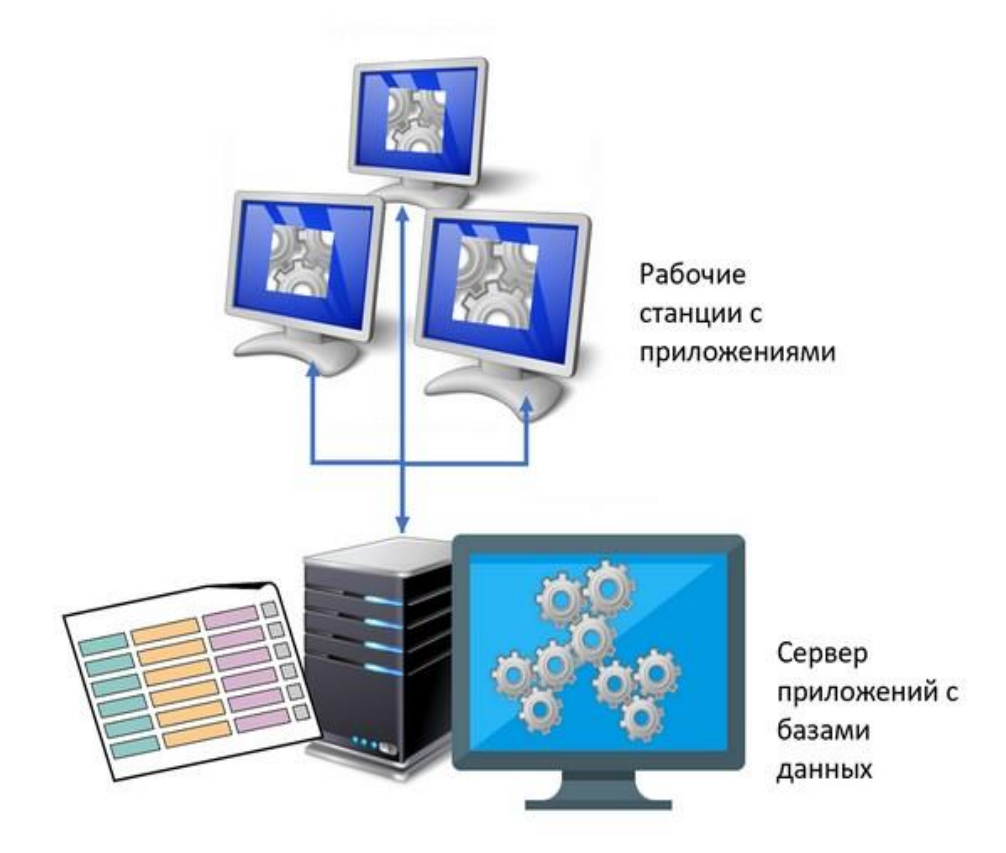

Рисунок 5 – Двухуровневая архитектура «клиент-сервер» (2-Tier)

Такая архитектура представляется наиболее логичной для архитектуры «клиент-сервер». В ней, однако, можно выделить два варианта. Когда общие данные хранятся на сервере, а логика их обработки и бизнес-данные хранятся на клиентской машине, то такая архитектура носит название «fat client thin server» (толстый клиент, тонкий сервер). Когда не только данные, но и логика их обработки и бизнес-данные хранятся на сервере, то это называется «thin client fat server» (тонкий клиент, толстый сервер). Такая архитектура послужила прообразом облачных вычислений (Cloud Computing).

Преимущества двухуровневой архитектуры:

- легко конфигурировать и модифицировать приложения;
- пользователю обычно легко работать в такой среде;
- хорошая производительность и масштабируемость.

Однако, ***у двухуровневой архитектуры есть и ограничения***:

- производительность может падать при увеличении числа пользователей;
- потенциальные проблемы с безопасностью, поскольку все данные и программы находятся на центральном сервере;
- все клиенты зависимы от базы данных одного производителя.

Трехуровневая архитектура (3-Tier). В трехуровневой архитектуре сервер баз данных, файловый сервер и другие представляют собой отдельный уровень, результаты работы которого использует сервер приложений. Логика данных и бизнес-логика находятся в сервере приложений. Все обращения клиентов к базе данных происходят через промежуточное программное обеспечение (middleware), которое находится на сервере приложений. Вследствие этого, повышается гибкость работы и производительность (рисунок 6).

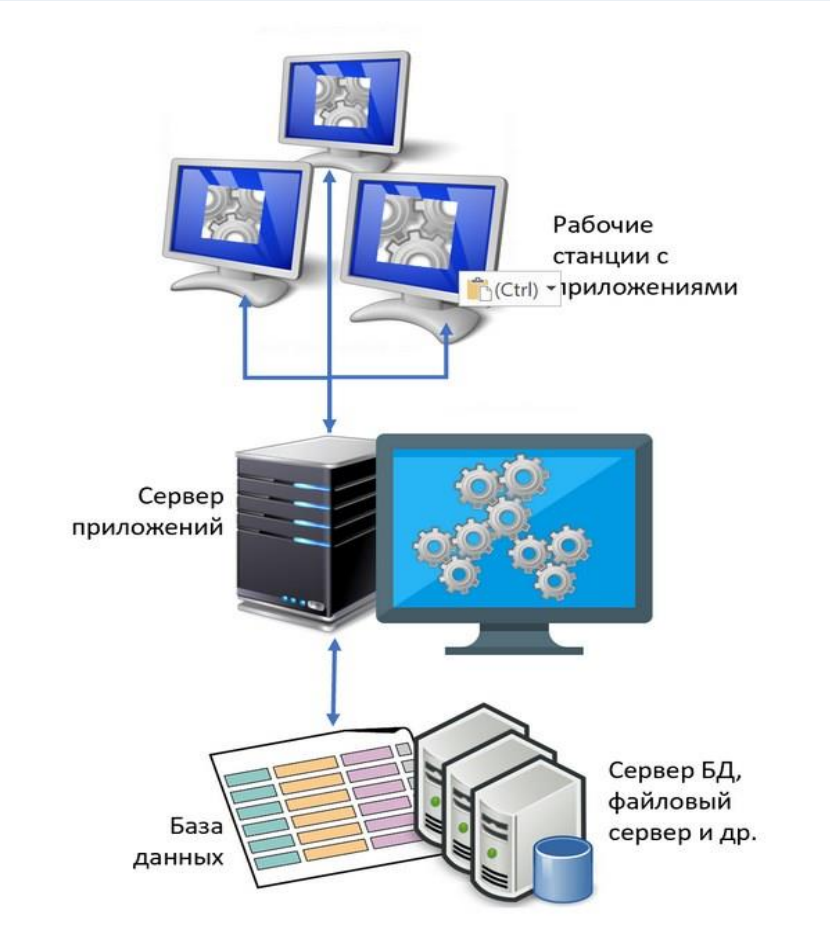

Рисунок 6 – Трехуровневая архитектура «клиент-сервер» (3-Tier)

Преимущества трехуровневой архитектуры:

- целостность данных;
- более высокая безопасность, по сравнению с двухуровневой архитектурой;
- защищенность базы данных от несанкционированного проникновения.

Ограничения:

- более сложная структура коммуникаций между клиентов и сервером, поскольку в нем также находится middleware.

Многоуровневая архитектура (N-Tier). В отдельный класс архитектуры «клиент-сервер» можно вынести многоуровневую архитектуру, в которой несколько серверов приложений используют результаты работы друг друга, а также данные от различных серверов баз данных, файловых серверов и других видов серверов (рисунок 7).

По сути, предыдущий вариант, трехуровневая архитектура – не более, чем частный случай многоуровневой архитектуры.

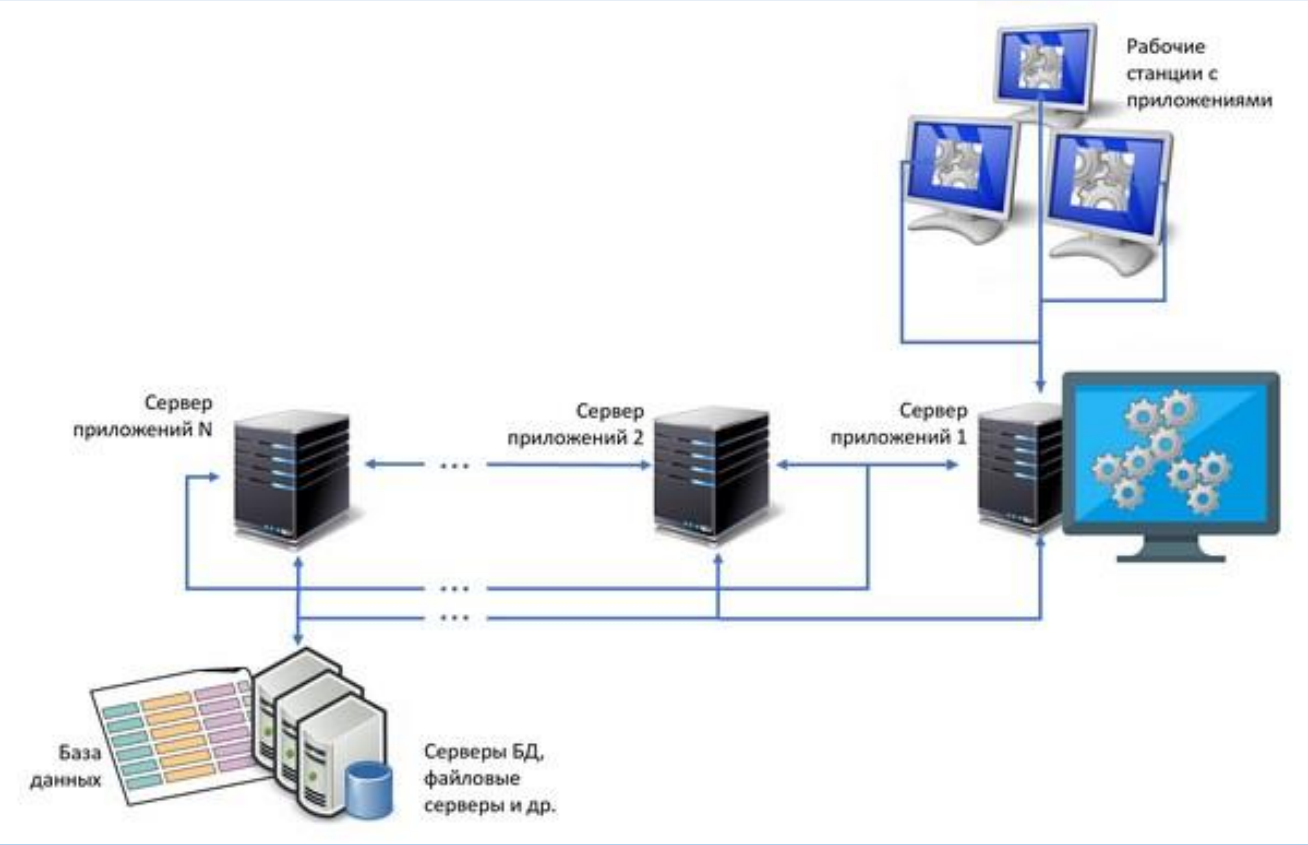

Рисунок 7 – Многоуровневая архитектура «клиент-сервер» (N-Tier)

Преимуществом многоуровневой архитектуры является гибкость предоставления услуг, которые могут являться комбинацией работы различных приложений серверов разных уровней и элементов этих приложений.

Очевидным недостатком является сложность, многокомпонентность такой архитектуры.

Характеристики архитектуры «клиент-сервер»:

Асимметричность протоколов. Между клиентами и сервером существуют отношения «один ко многим». Инициатором диалога с сервером обычно является клиент.

Инкапсуляция услуг. После получения запроса на услугу от клиента, сервер решает, как должна быть выполнена данная услуга. Модификация («апгрейд») сервера может производиться без влияния на работу клиентов, поскольку это не влияет на опубликованный интерфейс взаимодействия между ними. Иными словами, максимум, что может при этом почувствовать пользователь – незначительная задержка отклика сервера в течение небольшого времени апгрейда.

Целостность. Программы и общие данные для сервера управляются централизованно, что снижает стоимость обслуживания и защищает целостность данных. В то же время, данные клиентов остаются персонифицированными и независимыми.

Местная прозрачность. Сервер – это программный процесс, который может исполняться на той же машине, что и клиент, либо на другой машине, подключенной по сети. Программное обеспечение «клиент-сервер» обычно скрывает местоположение сервера от клиентов, перенаправляя запрос на услуги через сеть.

Обмен на основе сообщений. Клиенты и сервер являются нежестко связанными («loosely-coupled») процессами, которые обмениваются сообщениями: запросами на услуги и ответами на них.

Модульный дизайн, способный к расширению. Модульный дизайн программной платформы «клиент-сервер» придает ей устойчивость к отказам, то есть, отказ в каком-то модуле не вызывает отказа всего приложения. В такой системе, один или больше серверов могут отказать без остановки всей системы в целом, до тех пор, пока услуги отказавшего сервера могут быть предоставлены с резервного сервера. Другое преимущество модульности в том, что приложение «клиент-сервер» может автоматически реагировать на повышение или понижение нагрузки на систему, путем добавления или отключения услуг или серверов.

Независимость от платформы. Идеальное приложение «клиент-сервер» не зависит от платформ оборудования или операционной системы. Клиенты и серверы могут развертываться на различных аппаратных платформах и разных операционных системах.

Масштабируемость. Системы «клиент-сервер» могут масштабироваться как горизонтально (по числу серверов и клиентов), так и вертикально (по производительности и спектру услуг).

Разделение функционала. Система «клиент-сервер» – это соотношение между процессами, работающими на одной или на разных машинах. Сервер – это процесс предоставления услуг. Клиент – это потребитель услуг.

Общее использование ресурсов. Один сервер может предоставлять услуги множеству клиентов одновременно, и регулировать их доступ к совместно используемым ресурсам.

Практические применения архитектуры «клиент-сервер». Архитектура «клиент-сервер» – один из основных принципов работы сети Интернет. Любой веб-сайт, или приложение в Интернет работает на сервере, а его пользователи являются клиентами. Социальные сети (Фейсбук, ВК и пр.), сайты электронной коммерции (Amazon, Озон и др.), мобильные приложения (Instagram и т.д.), устройства Интернета вещей (умные колонки или смарт-часы) работают на основе клиент-серверной архитектуры.

Хорошим примером работы системы «клиент-сервер» является автомобильный навигатор. Приложение навигации на сервере собирает данные с многих смартфонов пользователей, на которых установлены клиенты приложения. Кроме того, приложение навигации использует еще и данные с сервера базы данных – геоинформационной системы, который предоставляет данные, например, о текущих ремонтах дорог, о появлении новых дорог и пр. Данные со многих клиентов (местоположение, скорость) обрабатывается сервером навигации и выдается на смартфоны пользователей в виде информации о средней скорости движения по тому или иному участку маршрута.

Практически любая корпоративная сеть или IT-система предприятия, как правило, строится по архитектуре «клиент-сервер». В небольших сетях (3-5 компьютеров в компании) функции сервера может выполнять один из рабочих компьютеров. Если число машин в организации более 10, то лучше сделать выделенный сервер (почтовый сервер, приложений, баз данных и пр.), который будет заниматься обслуживанием клиентов – компьютеров и телефонов сотрудников организации.

В домашних сетях архитектура «клиент-сервер» тоже используется довольно часто. Например, в домашнюю сеть могут быть объединены компьютеры членов семьи, один из которых выполняет функции сервера. В домашнюю сеть также могут быть включены такие устройства, как умные колонки, умные домашние устройства (пылесосы-роботы, фотоаппараты, DVD-плееры и пр.), а также «умные» счетчики (вода, электричество) и т.д. Тогда в системе управления сервера, будут видны все параметры, данные и медиа-файлы (музыка, видео, фото), а также «умные устройства».

Преимущества и недостатки архитектуры «клиент-сервер».

К преимуществам архитектуры «клиент-сервер» можно отнести:

- *централизованность*, поскольку все данные и управление сосредоточены в центральном сервере;
- *информационная безопасность*, поскольку ресурсы общего пользования администрируются централизованно;
- *производительность*, использование выделенного сервера повышает скорость работы ресурсов общего пользования;
- *масштабируемость*, количество клиентов и серверов можно увеличивать независимо друг от друга.

К недостаткам архитектуры «клиент-сервер» следует отнести:

- *перегрузку трафика в сети*, что является главной проблемой в сетях «клиент-сервер». Когда большое число клиентов одновременно запрашивают одну услугу на сервере, то число запросов может создать перегрузку в сети;
- *наличие единой точки отказа в небольших сетях с одним сервером*. Если он отказывает, все клиенты остаются без обслуживания;
- *превышение пределов ресурсов сервера*, когда новые клиенты, запрашивающие услугу, остаются без обслуживания. В таких случаях, требуется срочное расширение ресурсов сервера.

Иногда клиентские программы могут не работать на терминалах пользователей, если не установлены соответствующие драйверы. Например, пользователь посылает запрос на печать документа, а на сервере нет подходящего драйвера для печати данного формата документа на определенном принтере.

В настоящее время можно встретить термин Serverless Architecture, так называемая «бессерверная архитектура». Однако, по сути, она представляет собой процесс получения функций сервера в виде облачной услуги. То есть, серверы в облаке тоже есть, но для конечного пользователя они не видны, и он получает их сервисы в виде абстрактной «функции как услуги» FaaS (Function as a Service).

Архитектура «клиент-сервер» является основой большинства корпоративных сетей и берет свое начало от самых первых вычислительных машин – main-фреймов. Программное обеспечение для локальных компьютерных сетей, подавляющее большинство которых основано на архитектуре «клиент-сервер», начало создаваться около 50 лет назад.

Дальнейшее развитие информационных технологий также будет происходить в значительной степени с использованием архитектуры «клиент-сервер».

2. Двухуровневые модели распределенной обработки информации

Двухуровневые модели фактически являются результатом распределения пяти функций, определяющих основной принцип технологии «клиент-сервер», между двумя процессами, которые выполняются на двух платформах: на клиенте и на сервере. В чистом виде почти никакая модель не существует, однако рассмотрим наиболее характерные особенности каждой двухуровневой модели.

Модель удаленного управления данными (модель файлового сервера). Модель удаленного управления данными также называется моделью файлового сервера (File Server, FS). В этой модели презентационная логика и бизнес-логика располагаются на клиенте. На сервере располагаются файлы с данными и поддерживается доступ к файлам. Функции управления информационными ресурсами в этой модели находятся на клиенте. Распределение функций в этой модели представлено на рисунке 8.

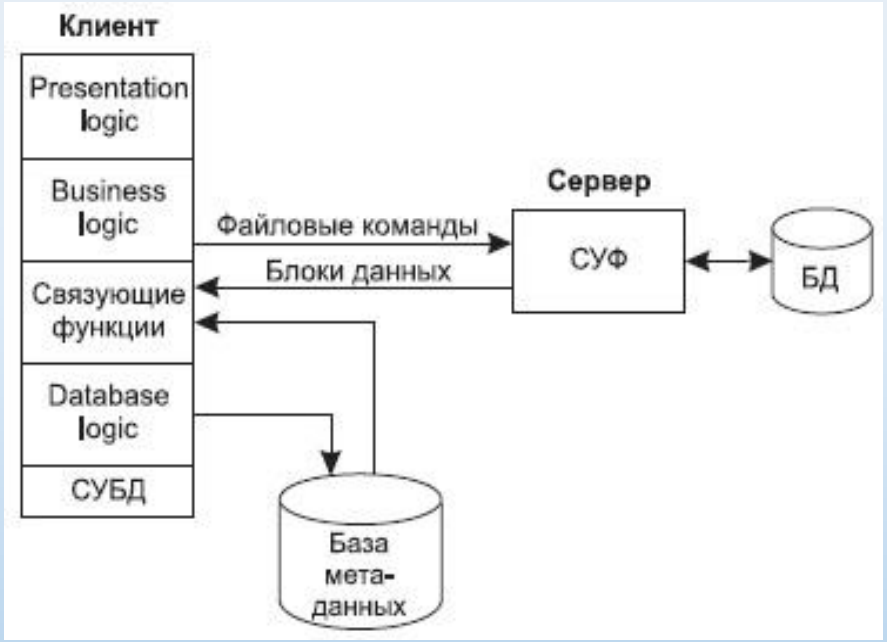

Рисунок 8 – Модель файлового сервера

В этой модели файлы базы данных хранятся на сервере, клиент обращается к серверу с файловыми командами, а механизм управления всеми информационными ресурсами, собственно база метаданных, находится на клиенте.

Достоинства этой модели в том, что мы уже имеем разделение монопольного приложения на два взаимодействующих процесса. При этом сервер (серверный процесс) может обслуживать множество клиентов, которые обращаются к нему с запросами. Собственно СУБД должна находиться в этой модели на клиенте.

Алгоритм выполнения запроса клиента. Запрос клиента формулируется в командах языка модели данных. СУБД переводит этот запрос в последовательность файловых команд. Каждая файловая команда вызывает перекачку блока информации на клиента, далее на клиенте СУБД анализирует полученную информацию, и если в полученном блоке не содержится ответ на запрос, то принимается решение о перекачке следующего блока информации и т. д.

Перекачка информации с сервера на клиент производится до тех пор, пока не будет получен ответ на запрос клиента.

Недостатки:

- высокий сетевой трафик, который связан с передачей по сети множества блоков и файлов, необходимых приложению;
- узкий спектр операций манипулирования с данными, который определяется только файловыми командами;
- отсутствие адекватных средств безопасности доступа к данным (защита только на уровне файловой системы).

Модель удаленного доступа к данным. В модели удаленного доступа (Remote Data Access, RDA) база данных хранится на сервере. На сервере же находится ядро СУБД. На клиенте располагается презентационная логика и бизнес-логика приложения. Клиент обращается к серверу с запросами на языке SQL. Структура модели удаленного доступа приведена на рисунке 9.

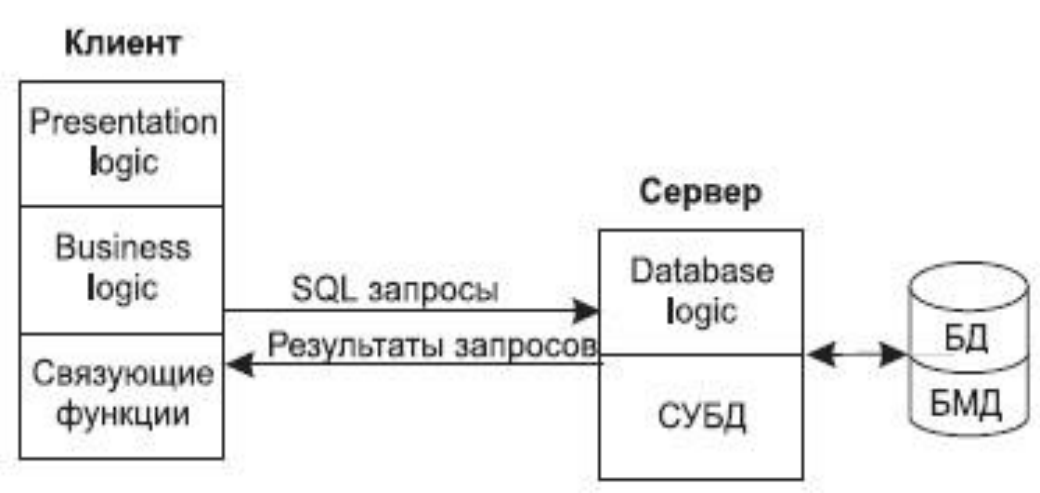

Рисунок 9 – Модель удаленного доступа (RDA)

Преимущества данной модели:

- перенос компонента представления и прикладного компонента на клиентский компьютер существенно разгрузил сервер БД, сводя к минимуму общее число процессов в операционной системе;
- сервер БД освобождается от несвойственных ему функций; процессор или процессоры сервера целиком загружаются операциями обработки данных, запросов и транзакций. (Это становится возможным, если отказаться от терминалов, не располагающих ресурсами, и заменить их компьютерами, выполняющими роль клиентских станций, которые обладают собственными локальными вычислительными ресурсами);
- резко уменьшается загрузка сети, так как по ней от клиентов к серверу передаются не запросы на ввод-вывод в файловой терминологии, а запросы на SQL, и их объем существенно меньше. В ответ на запросы клиент получает только данные, релевантные запросу, а не блоки файлов, как в FS-модели.

Основное достоинство RDA-модели – унификация интерфейса «клиент-сервер», стандартом при общении приложения-клиента и сервера становится язык SQL.

Недостатки:

- все-таки запросы на языке SQL при интенсивной работе клиентских приложений могут существенно загрузить сеть;
- так как в этой модели на клиенте располагается и презентационная логика, и бизнес-логика приложения, то при повторении аналогичных функций в разных приложениях код соответствующей бизнес-логики должен быть повторен для каждого клиентского приложения. Это вызывает излишнее дублирование кода приложений;
- сервер в этой модели играет пассивную роль, поэтому функции управления информационными ресурсами должны выполняться на клиенте. Действительно, например, если нам необходимо выполнять контроль страховых запасов товаров на складе, то каждое приложение, которое связано с изменением состояния склада, после выполнения операций модификации данных, имитирующих продажу или удаление товара со склада, должно выполнять проверку на объем остатка, и в случае, если он меньше страхового запаса, формировать соответствующую заявку на поставку требуемого товара. Это усложняет клиентское приложение, с одной стороны, а с другой – может вызвать необоснованный заказ дополнительных товаров несколькими приложениями.

Модель сервера баз данных. Для того чтобы избавиться от недостатков модели удаленного доступа, должны быть соблюдены следующие условия:

1. Необходимо, чтобы БД в каждый момент отражала текущее состояние предметной области, которое определяется не только собственно данными, но и связями между объектами данных. То есть данные, которые хранятся в БД, в каждый момент времени должны быть непротиворечивыми.

2. БД должна отражать некоторые правила предметной области, законы, по которым она функционирует (business rules). Например, завод может нормально работать только в том случае, если на складе имеется некоторый достаточный запас (страховой запас) деталей определенной номенклатуры, деталь может быть запущена в производство только в том случае, если на складе имеется в наличии достаточно материала для ее изготовления, и т.д.

3. Необходим постоянный контроль за состоянием БД, отслеживание всех изменений и адекватная реакция на них: например, при достижении некоторым измеряемым параметром критического значения должно произойти отключение определенной аппаратуры, при уменьшении товарного запаса ниже допустимой нормы должна быть сформирована заявка конкретному поставщику на поставку соответствующего товара.

4. Необходимо, чтобы возникновение некоторой ситуации в БД четко и оперативно влияло на ход выполнения прикладной задачи.

5. Одной из важнейших проблем СУБД является контроль типов данных. В настоящий момент СУБД контролирует синтаксически только стандартно-допустимые типы данных, то есть такие, которые определены в DDL (data definition language) – языке описания данных, который является частью SQL. Однако в реальных предметных областях у нас действуют данные, которые несут в себе еще и семантическую составляющую, например, это координаты объектов или единицы различных метрик, например рабочая неделя в отличие от реальной имеет сразу после пятницы понедельник.

Данную модель поддерживают большинство современных СУБД: Informix, Ingres, Sybase, Oracle, MS SQL Server.

Основу данной модели составляет механизм хранимых процедур как средство программирования SQL-сервера, механизм триггеров как механизм отслеживания текущего состояния информационного хранилища и механизм ограничений на пользовательские типы данных, который иногда называется механизмом поддержки доменной структуры. Модель сервера баз данных представлена на рисунке 10.

Рисунок 10 – Модель активного сервера БД

В этой модели бизнес-логика разделена между клиентом и сервером. На сервере бизнес-логика реализована в виде хранимых процедур – специальных программных модулей, которые хранятся в БД и управляются непосредственно СУБД. Клиентское приложение обращается к серверу с командой запуска хранимой процедуры, а сервер выполняет эту процедуру и регистрирует все изменения в БД, которые в ней предусмотрены. Сервер возвращает клиенту данные, релевантные его запросу, которые требуются клиенту либо для вывода на экран, либо для выполнения части бизнес-логики, которая расположена на клиенте. Трафик обмена информацией между клиентом и сервером резко уменьшается.

Централизованный контроль в модели сервера баз данных выполняется с использованием механизма триггеров. Триггеры также являются частью БД.

Термин ***«триггер»*** взят из электроники и семантически очень точно характеризует механизм отслеживания специальных событий, которые связаны с состоянием БД. Триггер в БД является как бы некоторым тумблером, который срабатывает при возникновении определенного события в БД. Ядро СУБД проводит мониторинг всех событий, которые вызывают созданные и описанные триггеры в БД, и при возникновении соответствующего события сервер запускает соответствующий триггер. Каждый триггер представляет собой также некоторую программу, которая выполняется над базой данных. Триггеры могут вызывать хранимые процедуры.

Механизм использования триггеров предполагает, что при срабатывании одного триггера могут возникнуть события, которые вызовут срабатывание других триггеров. Этот мощный инструмент требует тонкого и согласованного применения, чтобы не получился бесконечный цикл срабатывания триггеров.

В данной модели сервер является активным, потому что не только клиент, но и сам сервер, используя механизм триггеров, может быть инициатором обработки данных в БД.

И хранимые процедуры, и триггеры хранятся в словаре БД, они могут быть использованы несколькими клиентами, что существенно уменьшает дублирование алгоритмов обработки данных в разных клиентских приложениях.

Для написания хранимых процедур и триггеров используется расширение стандартного языка SQL, так называемый встроенный SQL.

Недостатком данной модели является очень большая загрузка сервера. Действительно, сервер обслуживает множество клиентов и выполняет следующие функции:

- осуществляет мониторинг событий, связанных с описанными триггерами;
- обеспечивает автоматическое срабатывание триггеров при возникновении связанных с ними событий;
- обеспечивает исполнение внутренней программы каждого триггера;
- запускает хранимые процедуры по запросам пользователей;
- запускает хранимые процедуры из триггеров;
- возвращает требуемые данные клиенту;
- обеспечивает все функции СУБД: доступ к данным, контроль и поддержку целостности данных в БД, контроль доступа, обеспечение корректной параллельной работы всех пользователей с единой БД.

Если мы переложили на сервер большую часть бизнес-логики приложений, то требования к клиентам в этой модели резко уменьшаются. Иногда такую модель называют моделью с «тонким клиентом», в отличие от предыдущих моделей, где на клиента возлагались гораздо более серьезные задачи. Эти модели называются моделями с «толстым клиентом».

Для разгрузки сервера была предложена трехуровневая модель.

3. Модель сервера приложений

Эта модель является расширением двухуровневой модели и в ней вводится дополнительный промежуточный уровень между клиентом и сервером. Архитектура трехуровневой модели приведена на рисунке 11. Этот промежуточный уровень содержит один или несколько серверов приложений.

В этой модели компоненты приложения делятся между тремя исполнителями:

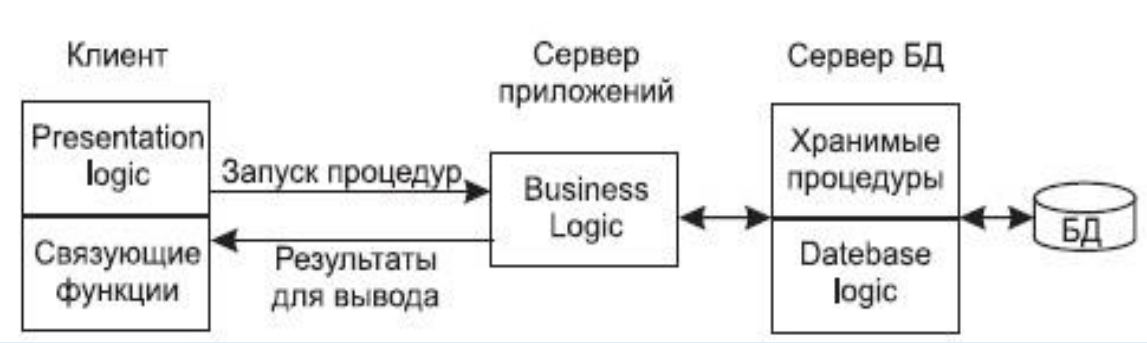

Рисунок 11 – Модель сервера приложений

Клиент обеспечивает логику представления, включая графический пользовательский интерфейс, локальные редакторы; клиент может запускать локальный код приложения клиента, который может содержать обращения к *локальной БД*, расположенной на компьютере-клиенте. Клиент исполняет коммуникационные функции front-end части приложения, которые обеспечивают доступ клиенту в локальную или глобальную сеть. Дополнительно реализация взаимодействия между клиентом и сервером может включать в себя управление распределенными транзакциями, что соответствует тем случаям, когда клиент также является клиентом менеджера распределенных транзакций.

Серверы приложений составляют новый промежуточный уровень архитектуры. Они спроектированы как исполнения общих незагружаемых функций для клиентов. Серверы приложений поддерживают функции клиентов как частей взаимодействующих рабочих групп, поддерживают сетевую доменную операционную среду, хранят и исполняют наиболее общие правила бизнес-логики, поддерживают каталоги с данными, обеспечивают обмен сообщениями и поддержку запросов, особенно в распределенных транзакциях.

Серверы баз данных в этой модели занимаются исключительно функциями СУБД: обеспечивают функции создания и ведения БД, поддерживают целостность реляционной БД, обеспечивают функции хранилищ данных (warehouse services). Кроме того, на них возлагаются функции создания резервных копий БД и восстановления БД после сбоев, управления выполнением транзакций и поддержки устаревших (унаследованных) приложений (legacy application).

Отметим, что эта модель обладает большей гибкостью, чем двухуровневые модели. Наиболее заметны преимущества модели сервера приложений в тех случаях, когда клиенты выполняют сложные аналитические расчеты над базой данных, которые относятся к области OLAP-приложений (On-line analytical processing). В этой модели большая часть бизнес-логики клиента изолирована от возможностей встроенного SQL, реализованного в конкретной СУБД, и может быть выполнена на стандартных языках программирования, таких как C, C++, SmallTalk, Cobol. Это повышает переносимость системы, ее масштабируемость. Функции промежуточных серверов могут быть в этой модели распределены в рамках глобальных транзакций путем поддержки XA-протокола (X/Open transaction interface protocol), который поддерживается большинством поставщиков СУБД.

4. Модели серверов баз данных

В период создания первых СУБД технология «клиент-сервер» только зарождалась. Поэтому изначально в архитектуре систем не было адекватного механизма организации взаимодействия процессов типа «клиент» и процессов типа «сервер». В современных же СУБД он является фактически основополагающим и от эффективности его реализации зависит эффективность работы системы в целом.

Рассмотрим эволюцию типов организации подобных механизмов. В основном этот механизм определяется структурой реализации серверных процессов, и часто он называется архитектурой сервера баз данных.

Первоначально, как мы уже отмечали, существовала модель, когда управление данными (функция сервера) и взаимодействие с пользователем были совмещены в одной программе. Это можно назвать нулевым этапом развития серверов БД.

Затем функции управления данными были выделены в самостоятельную группу – сервер, однако модель взаимодействия пользователя с сервером соответствовала парадигме «один-к-одному» (рисунок 12), то есть сервер обслуживал запросы только одного пользователя (клиента), и для обслуживания нескольких клиентов нужно было запустить эквивалентное число серверов.

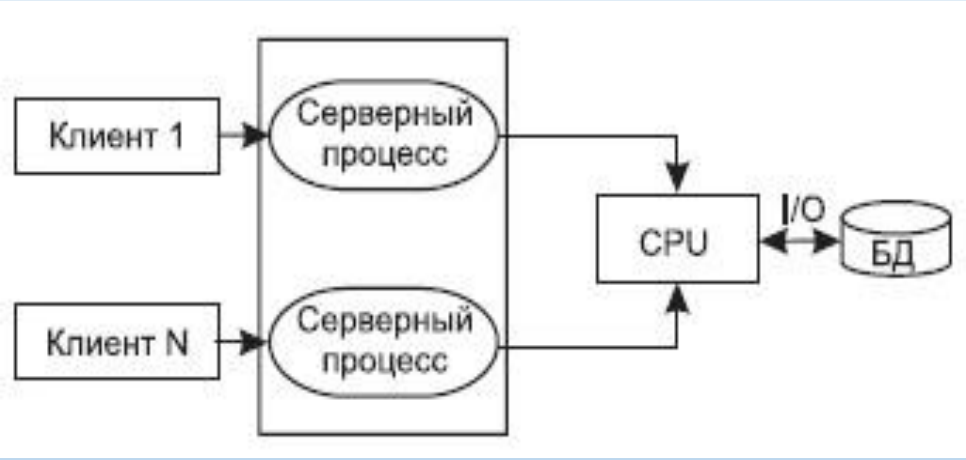

Рисунок 12 – Взаимодействие пользовательских и клиентских процессов в модели «один-к-одному»

Выделение сервера в отдельную программу было революционным шагом, который позволил, в частности, поместить сервер на одну машину, а программный интерфейс с пользователем — на другую, осуществляя взаимодействие между ними по сети. Однако необходимость запуска большого числа серверов для обслуживания множества пользователей сильно ограничивала возможности такой системы.

Для обслуживания большого числа клиентов на сервере должно быть запущено большое количество одновременно работающих серверных процессов, а это резко повышало требования к ресурсам ЭВМ, на которой запускались все серверные процессы. Кроме того, каждый серверный процесс в этой модели запускался как независимый, поэтому если один клиент сформировал запрос, который был только что выполнен другим серверным процессом для другого клиента, то запрос тем не менее выполнялся повторно. В такой модели весьма сложно обеспечить взаимодействие серверных процессов. Эта модель самая простая, и исторически она появилась первой.

Проблемы, возникающие в модели «один-к-одному», решаются в архитектуре ***«систем с выделенным сервером»***, который способен обрабатывать запросы от многих клиентов. Сервер единственный обладает монополией на управление данными и взаимодействует одновременно со многими клиентами (рисунок 13).

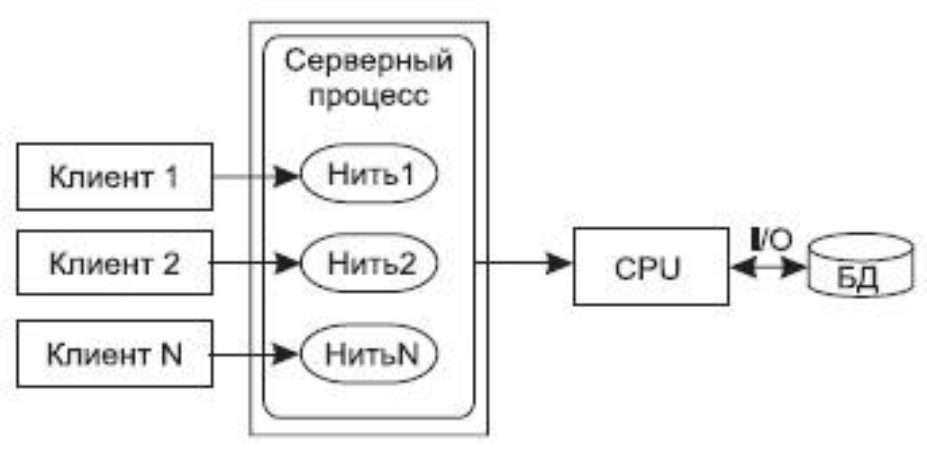

Рисунок 13 – Многопотоковая односерверная архитектура

Логически каждый клиент связан с сервером отдельной нитью (thread), или потоком, по которому пересылаются запросы. Такая архитектура получила название ***многопотоковой односерверной*** (multi-threaded).

Она позволяет значительно уменьшить нагрузку на операционную систему, возникающую при работе большого числа пользователей (trashing).

Кроме того, возможность взаимодействия с одним сервером многих клиентов позволяет в полной мере использовать разделяемые объекты (начиная с открытых файлов и кончая данными из системных каталогов), что значительно уменьшает потребности в памяти и общее число процессов операционной системы. Например, системой с архитектурой «один-к-одному» будет создано 100 копий процессов СУБД для 100 пользователей, тогда как системе с многопотоковой архитектурой для этого понадобится только один серверный процесс.

Однако такое решение имеет свои недостатки. Так как сервер может выполняться только на одном процессоре, возникает естественное ограничение на применение СУБД для мультипроцессорных платформ. Если компьютер имеет, например, четыре процессора, то СУБД с одним сервером используют только один из них, не загружая оставшиеся три.

В некоторых системах эта проблема решается вводом промежуточного диспетчера. Подобная архитектура называется ***архитектурой виртуального сервера*** (virtual server) (рисунок 14).

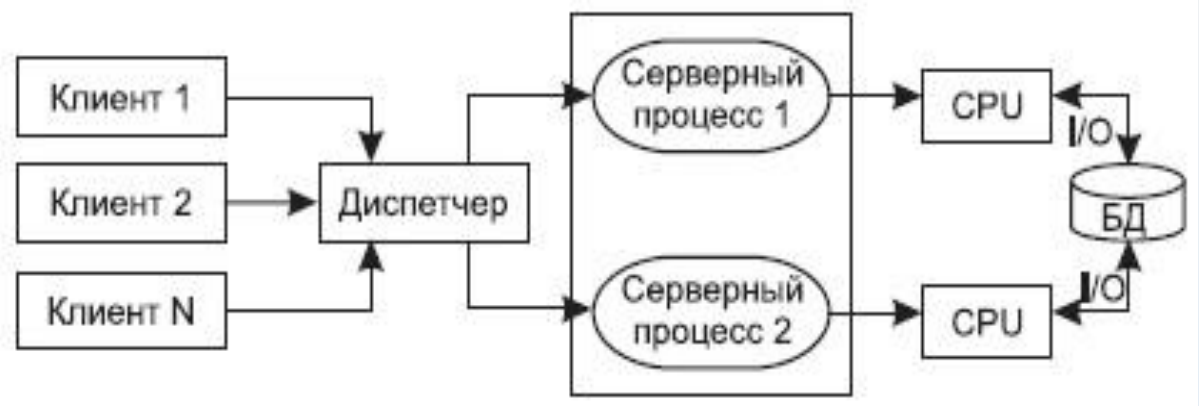

Рисунок 14 – Архитектура с виртуальным сервером

В этой архитектуре клиенты подключаются не к реальному серверу, а к промежуточному звену, называемому диспетчером, который выполняет только функции диспетчеризации запросов к актуальным серверам. В этом случае нет ограничений на использование многопроцессорных платформ. Количество актуальных серверов может быть согласовано с количеством процессоров в системе.

Однако и эта архитектура не лишена недостатков, потому что здесь в систему добавляется новый слой, который размещается между клиентом и сервером, что увеличивает трату ресурсов на поддержку баланса загрузки актуальных серверов (load balancing) и ограничивает возможности управления взаимодействием «клиент-сервер». Во-первых, становится невозможным направить запрос от конкретного клиента конкретному серверу, во-вторых, серверы становятся равноправными – нет возможности устанавливать приоритеты для обслуживания запросов.

Подобная организация взаимодействия клиент-сервер может рассматриваться как аналог банка, где имеется несколько окон кассиров, и специальный банковский служащий – администратор зала (диспетчер) направляет каждого вновь пришедшего посетителя (клиента) к свободному кассиру (актуальному серверу). Система работает нормально, пока все посетители равноправны (имеют равные приоритеты), однако стоит лишь появиться посетителям с высшим приоритетом, которые должны обслуживаться в специальном окне, как возникают проблемы. Учет приоритета клиентов особенно важен в системах оперативной обработки транзакций, однако именно эту возможность не может предоставить архитектура систем с диспетчеризацией.

Современное решение проблемы СУБД для мультипроцессорных платформ заключается в возможности запуска нескольких серверов базы данных, в том числе и на различных процессорах. При этом каждый из серверов должен быть многопотоковым. Если эти два условия выполнены, то есть основания говорить о ***многопотоковой архитектуре с несколькими серверами***, представленной на рисунке 15.

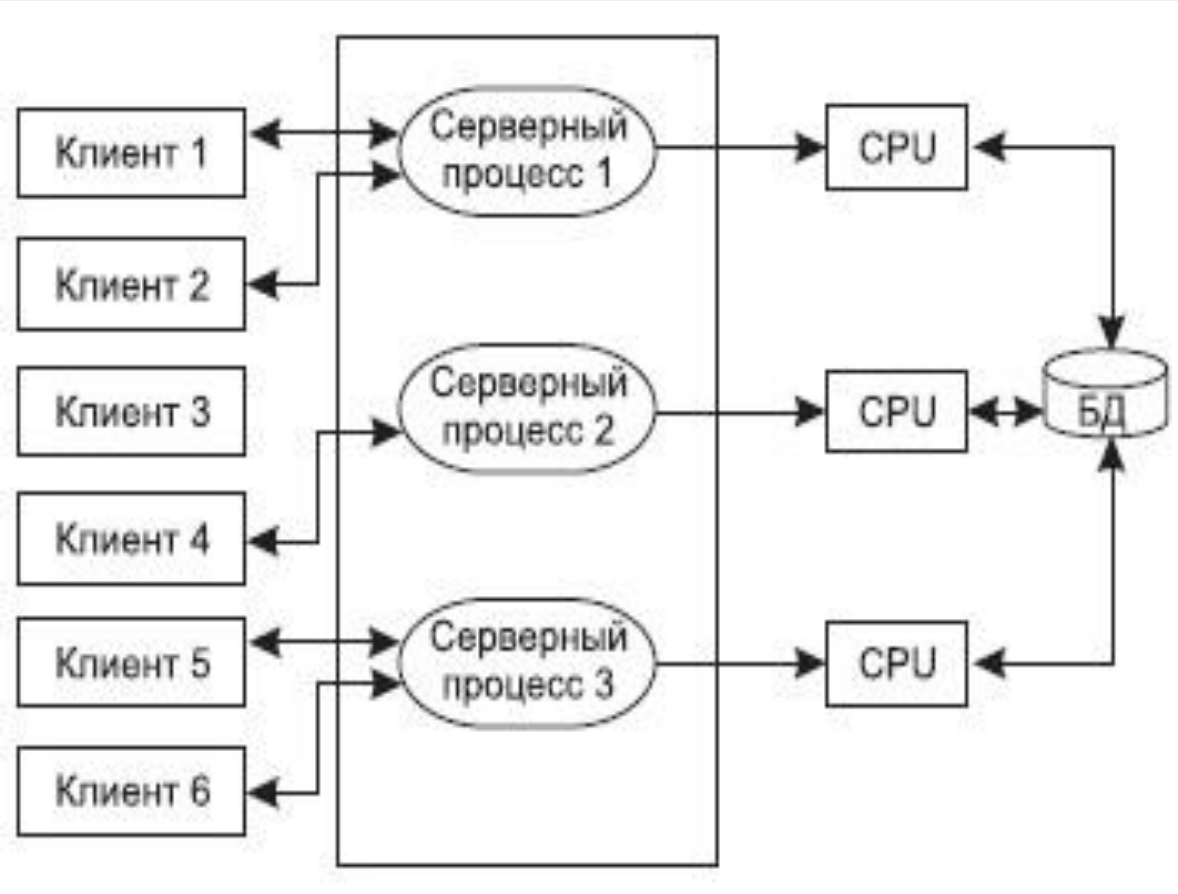

Рисунок 15 – Многопотоковая мультисерверная архитектура

Она также может быть названа ***многонитевой мульти-серверной архитектурой***.

Эта архитектура связана с вопросами распараллеливания выполнения одного пользовательского запроса несколькими серверными процессами.

Существует несколько возможностей распараллеливания выполнения запроса. В этом случае пользовательский запрос разбивается на ряд подзапросов, которые могут выполняться параллельно, а результаты их выполнения потом объединяются в общий результат выполнения запроса. Тогда для обеспечения оперативности выполнения запросов их подзапросы могут быть направлены отдельным серверным процессам, а потом полученные результаты объединены в общий результат (рисунок 16).

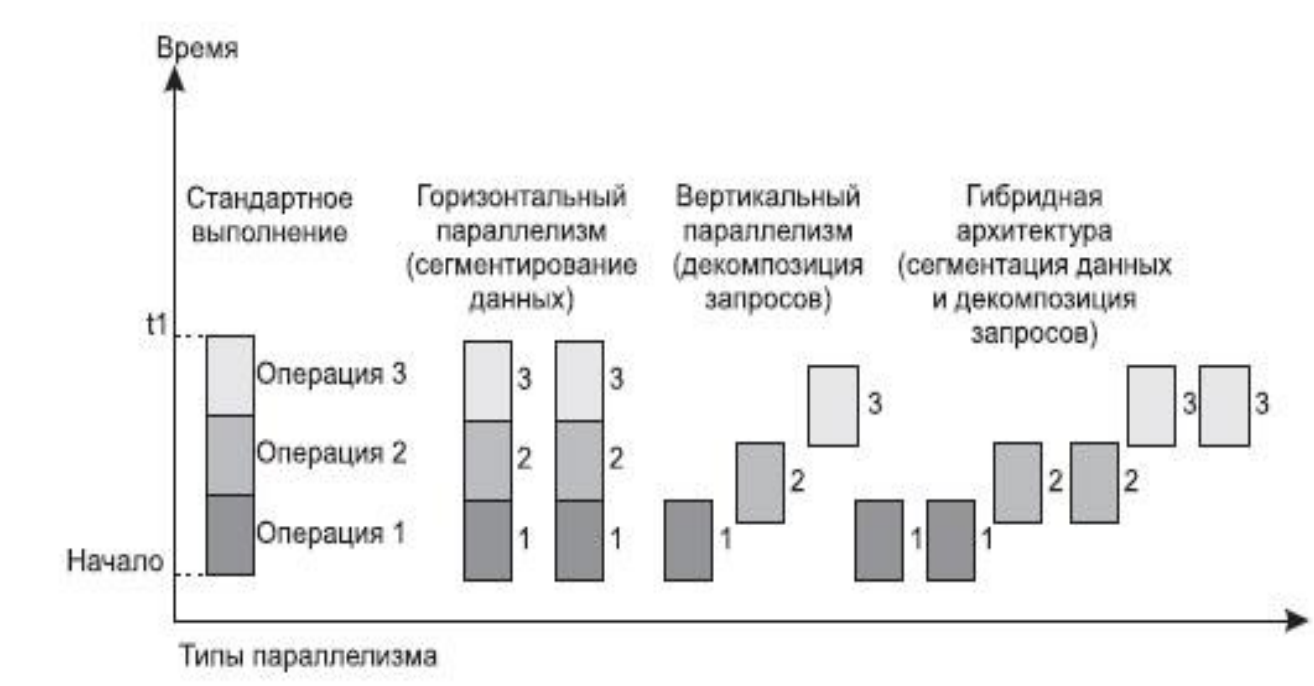

Рисунок 16 – Многонитевая мультисерверная архитектура

В данном случае серверные процессы не являются независимыми процессами, такими, как рассматривались ранее. Эти серверные процессы принято называть ***нитями*** (treads), и управление нитями множества запросов пользователей требует дополнительных расходов от СУБД, однако при оперативной обработке информации в хранилищах данных такой подход наиболее перспективен.

Типы параллелизма. Рассматривают несколько путей распараллеливания запросов.

Горизонтальный параллелизм. Этот параллелизм возникает тогда, когда хранимая в БД информация распределяется по нескольким физическим устройствам хранения – нескольким дискам. При этом информация из одного отношения разбивается на части по горизонтали (рисунок 17).

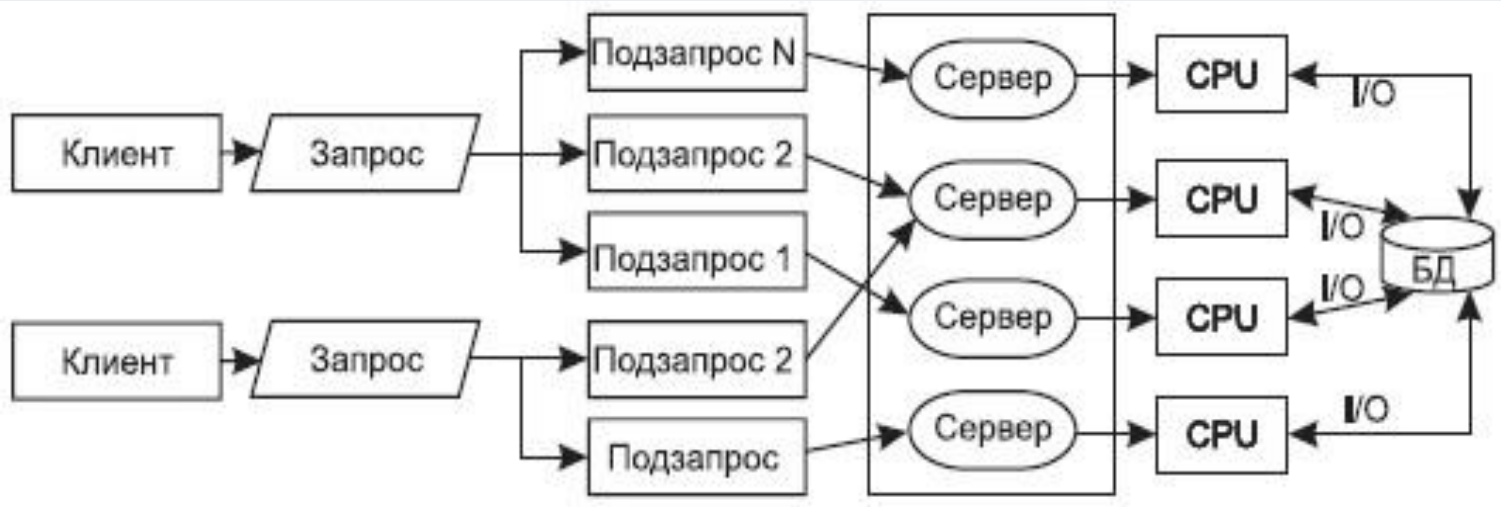

Рисунок 17 – Выполнение запроса при горизонтальном параллелизме

Этот вид параллелизма иногда называют ***распараллеливанием*** или ***сегментацией данных***.

И параллельность здесь достигается путем выполнения одинаковых операций, например фильтрации, над разными физическими хранимыми данными.

Эти операции могут выполняться параллельно разными процессами, они независимы.

Результат выполнения целого запроса складывается из результатов выполнения отдельных операций.

Время выполнения такого запроса при соответствующем сегментировании данных существенно меньше, чем время выполнения этого же запроса традиционными способами одним процессом.

Вертикальный параллелизм. Этот параллелизм достигается конвейерным выполнением операций, составляющих запрос пользователя. Этот подход требует серьезного усложнения в модели выполнения реляционных операций ядром СУБД. Он предполагает, что ядро СУБД может произвести декомпозицию запроса, базируясь на его функциональных компонентах, и при этом ряд подзапросов может выполняться параллельно, с минимальной связью между отдельными шагами выполнения запроса.

Действительно, если мы рассмотрим, например, последовательность операций реляционной алгебры:

R5=R1 [A,C] R6=R2 [A,B,D] R7 = R5[A > 128] R8 = R5[A]R6,

то операции первую и третью можно объединить и выполнить параллельно с операцией два, а затем выполнить над результатами последнюю четвертую операцию.

Общее время выполнения подобного запроса, конечно, будет существенно меньше, чем при традиционном способе выполнения последовательности из четырех операций (см. рисунок 17).

И ***третий вид параллелизма является гибридом двух ранее рассмотренных*** (рисунок 18).

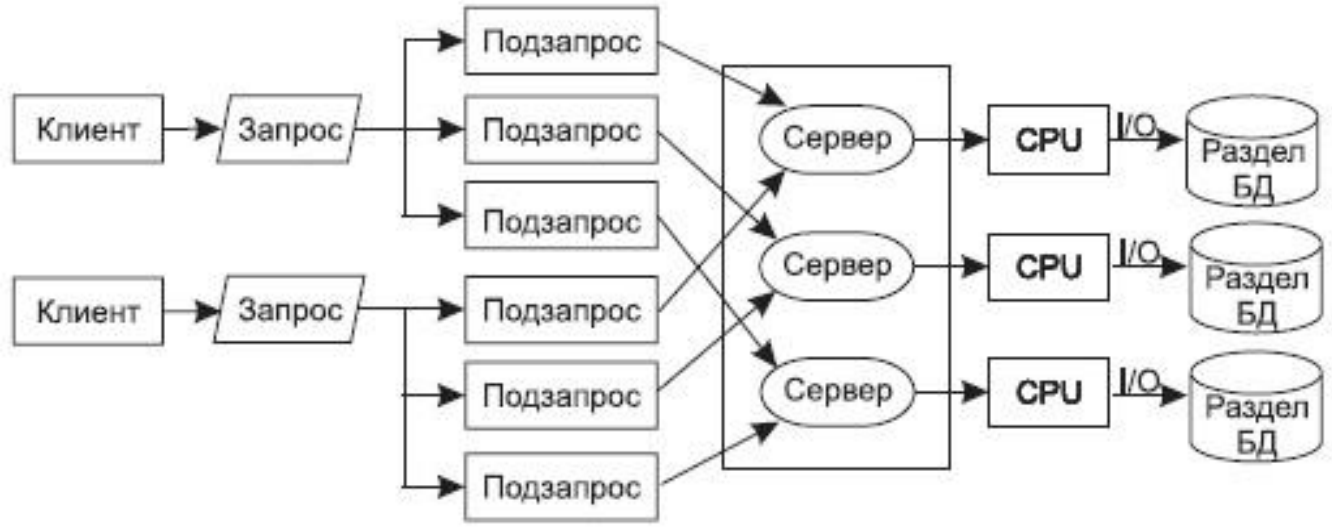

Рисунок 18 – Выполнение запроса при гибридном параллелизме

Наиболее активно применяются все виды параллелизма в OLAP-приложениях, где эти методы позволяют существенно сократить время выполнения сложных запросов над очень большими объемами данных.

Выводы

В ходе лекции рассмотрены следующие вопросы:

- технология и архитектура «клиент-сервер»;

- двухуровневые модели распределенной обработки информации;

- модель сервера приложений;

- модели серверов баз данных.

Задание на самостоятельную работу

1. Конспект лекций.

Вид и тема следующего занятия

Практическое занятие №4. Основы в ASP.NET Core (ч. 4)