# Некоторые особенности проектирования под конкретные архитектуры

## Архитектура «клиент-сервер»

Архитектура «клиент-сервер» предоставляет пользователю одно основное преимущество - возможность получить графический пользовательский интерфейс, не модернизируя сервер. По сравнению с файл-серверными архитектурами в данном случае резко снижается трафик (правда, только при правильном проектировании).

Существует два ключевых момента любой архитектуры:

* Число пар операций обмена (запрос, ответ), прежде всего там, где используются медленные и неустойчивые каналы связи.
* Строгое разделение задачи управления пользовательским интерфейсом и задачи управления данными. Нарушение этого принципа - одна из главных причин высокого сетевого трафика.

В случае архитектуры «клиент-сервер» обработка данных производится централизованно на сервере. Клиент посылает запрос к данным, сервер обрабатывает этот запрос и передает клиенту ответ. Казалось бы, все очень просто: ПО на сервере одно и то же, равно как и набор клиентских приложений один и тот же, так что все должно замечательно работать в любом случае.

Клиентское приложение должно идентифицировать по имени целевой сервер, к которому оно посылает запрос, а поиск физического сервера осуществляется на основании правил разрешения имен. Эти функции берут на себя серверный и клиентский компоненты СУБД, которые обеспечивают сетевой сервис. Все вопросы разрешения имен находятся в компетенции настроек этого ПО, а не собственно приложения-клиента. По сути, приложение-клиент общается непосредственно с клиентским ПО сетевого драйвера СУБД, а не с ядром СУБД. При этом:

* Клиентский и серверный процессы могут находиться как на одной, так и на разных машинах.
* Клиентский и серверный процессы могут работать в разном окружении (аппаратное обеспечение, операционная система). Архитектура обеспечивает независимость клиентского ПО от серверного ПО СУБД. Независимость достигается за счет строго определенного интерфейса обмена, который, в свою очередь, осуществляется посредством посылки сообщений строго определенного формата.
* Клиентский процесс не знает, какой сервер в итоге будет обрабатывать его запрос. Средства передачи запросов от сервиса к сервису контролируются как СУБД, так и другими серверными процессами (например, запрос был обработан Web-сервером, преобразовавшим его в запрос для монитора транзакций, который обратился непосредственно к СУБД, а передача ответа идет в обратном порядке - в результате клиентский запрос последовательно обработан тремя серверами).

Все эти положения, по сути, описывают инкапсуляцию, которая является необходимым условием успешного проектирования в этой архитектуре.

Проектируя любое приложение в архитектуре «клиент-сервер», следует обратить внимание на минимизацию операций обмена между клиентом и сервером. Вынесение части бизнес-логики на сервер в виде хранимых процедур позволяет снизить количество операций обмена.

Еще одним фактором является количество соединений одного приложения с сервером баз данных. СУБД на каждое соединение выделяет определенный ресурс. Если приложение открыло 10 соединений, а работает только одно из них, то СУБД реально резервирует все 10 ресурсов. Большинство интерактивных приложений характеризуются тем, что более 90% времени по открытым этими приложениями соединениям не ведется никакой работы. Для некоторых СУБД большое количество неактивных соединений серьезно снижает эффективность работы сервера, например для Oracle (поскольку при открытии соединения системные ресурсы под него выделяются сразу). Лучше пусть соединений будет меньше, но они будут работать.

Важным фактором является пропускная способность сети, а также пропускная способность выбранного протокола обмена данными. Большинство СУБД поддерживают протоколы Named Pipes, TCP/IP, NetBIOS, IPX/SPX. Все эти протоколы изначально не предназначались для обмена с базами данных. Некоторые производители реализуют собственные протоколы обмена - например, APPC, предоставляемый IBM DB2, специально оптимизирован для операций обмена, характерных для баз данных. Полоса пропускания протокола становится критичной только в том случае, когда передаваемые пакеты являются большими, например при передаче изображений и звука. Полоса пропускания протокола - один из возможных параметров, которые следует оптимизировать для повышения скорости передачи информации.

Основная задача проектирования в архитектуре «клиент-сервер» - реализовать на клиенте качественный прикладной интерфейс (экранные формы и, может быть, небольшое число проверок корректности ввода данных) и осуществить на сервере обработку разделяемых данных.

Модель, представленная на рис. 1, показывает типы распределения нагрузки между клиентом и сервером. Эту же модель можно распространить и на трехуровневые архитектуры, где уровнями являются клиент, ПО промежуточного слоя, сервер. Чем более жестко вы примените инкапсуляцию при проектировании, тем меньше будет вероятность того, что маленькие фрагменты управления логикой представления данных окажутся спрятанными в коде управления данными. Если этого не сделать, вы получите очень чувствительный к изменениям прикладной код, а значит, создадите трудности как при разработке кода, так и при поддержке приложения.

Наличие тонкого клиента также важно в современных информационных системах. Большинство серверов баз данных оптимизировано для селективной выборки данных, серверное ПО рассчитано на прием и передачу данных за минимальное время. Не следует злоупотреблять на разделяемых ресурсах сервера прикладными программами, используемыми клиентами. Это, конечно, гарантирует факт запуска одного и того же ПО с любой из клиентских машин, но нужно правильно оценивать стоимость этих операций с точки зрения сетевого трафика.

Если одни и те же данные часто запрашиваются с сервера, то их можно кэшировать на клиенте. Это, как правило, небольшие, редко обновляемые справочники. Такой подход снижает количество запросов к данным с сервера, а следовательно, понижает нагрузку не только на сеть, но и на сам сервер. Приложение-клиент должно уметь работать с таким кэшем данных. Организация его - скорее компетенция разработчиков, чем проектировщиков. Более серьезная задача - это обеспечение непротиворечивости и актуальности данных в кэше. Если в СУБД реализован механизм событий (event), который позволяет по факту модификации данных в справочнике послать клиенту уведомление об этом событии, последнее обрабатывается приложением-клиентом и вызывается функция, которая инициирует новое считывание справочника и обновление данных в кэше. Такие вызовы могут быть асинхронными.

Если же справочники велики, упорядочены всегда одинаково и изменяются мало, можно хранить их на клиенте не в кэше, а в обычных файлах. Это может расцениваться как некий упрощенный вариант распределенной базы данных, когда часть считанных данных хранится на клиенте в течение всего времени его работы. Таким образом можно хранить изображение и звук, а также другую информацию большого объема. Актуализация указанных данных также может быть реализована посредством механизма событий.

## Трехуровневая архитектура

Сейчас часто употребляют термин «ПО промежуточного слоя». Некоторые авторы определяют этим понятием даже драйверы ODBC, иными словами, считают, что все, что работает между интерфейсом native-вызовов сервера и интерфейсом вызовов клиента, является ПО промежуточного слоя. Если допустить, что такое ПО обеспечивает только независимость клиента и сервера, то подобный термин имеет право на существование.

К ПО промежуточного слоя относят мониторы транзакций. В этом случае правила обработки данных, правила процессов определяются на языке монитора транзакций. Клиентское ПО обращается только к монитору транзакций. Вся обработка данных находится в компетенции монитора транзакций.

ПО промежуточного слоя обладает всеми системными функциями, например функциями сервера печати. Большинство проектировщиков мотивируют использование трехуровневой архитектуры необходимостью сконцентрировать логику обработки данных, системные функции в одном месте и убрать эти функции из состава ПО клиента.

ПО промежуточного слоя актуально в тех случаях, когда требуется обеспечить прозрачность доступа к данным, управляемым разными СУБД; здесь проблемы общения с СУБД берет на себя сервер приложений.

Часто ПО промежуточного слоя обеспечивает дополнительную защиту данных от несанкционированного доступа, например когда требуется уровень защиты B2 и выше. Некоторые СУБД обеспечивают такой уровень защиты собственными средствами - речь идет о Trusted-версиях, например Oracle, Sybase, Informix. Функции защиты данных обеспечиваются не только СУБД, но и специальными версиями операционных систем. Функции Trusted-сервера выполняет также IBM DB2 на архитектурах S/390, AS/400 в соответствующем окружении. К Trusted-серверам также относится Linter SQL Server производства российской компании РЕЛЭКС - защита данных здесь полностью обеспечивается ядром СУБД. В России эта СУБД сертифицирована по 2-му классу защищенности и может быть использована для построения защищенных информационных систем (сертификат действителен для версий СУБД, функционирующих в Windows NT, UNIX, Novell NetWare).

## Распределенная база данных, параллельный сервер баз данных

Есть много причин, которые могут подтолкнуть проектировщика к принятию решения о построении распределенной базы данных. Приведем некоторые из них:

* Размещение часто используемых данных ближе к клиенту, что позволяет минимизировать сетевой трафик.
* Расположение часто меняющихся данных в одном месте, обеспечивающее минимизацию затрат по синхронизации копий данных.
* Увеличение надежности комплекса к единичным отказам серверов (горячее резервирование).
* Понижение стоимости системы за счет использования группы небольших серверов вместо одного мощного центрального сервера.

Если в информационной системе наличествуют все перечисленные факторы, то распределенная база данных может оказаться хорошим решением. Часто распределение базы данных изначально определяется требованиями бизнеса, например когда объединяется информация нескольких крупных подразделений одной компании и требуется постоянный обмен данными между филиалами.

Решение о распределенной базе данных оправданно и для систем, где есть четко выраженные группа меняющихся данных и группа устойчивых данных, по которым выполняются отчеты. Тогда в самом простом варианте работают два сервера: один обслуживает часто меняющиеся данные - это, как правило, OLTP (On-Line Transaction Processing. - Прим. ред.), второй - отчеты, то есть DSS (Decision Support System. - Прим. ред.). Ряд СУБД не очень хорошо совмещает обработку OLTP- и DSS-потоков запросов, поскольку для этих двух типов потоков запросов оптимальные параметры конфигурации серверов будут различаться. Решение такой базы данных как распределенной может оказаться более выгодным.

Преимущества распределенной базы данных имеют свою цену - должны быть обеспечены:

* Непротиворечивость данных, независимо от того, какой клиент к какому серверу обратился.
* Производительность распределенной базы данных должна удовлетворять требованиям заказчика, а это может оказаться более сложной задачей, чем в случае централизованной базы данных.
* Надежность распределенной базы данных не должна уступать надежности централизованной базы данных.

Современные СУБД позволяют осуществлять запрос данных, находящихся на разных узлах распределенной сети в одном SQL-запросе. Прозрачность доступа СУБД также обеспечивают - в каких-то реализациях хуже, в каких-то лучше. Одна из самых распространенных проблем - более сложная обработка транзакций. Практически все промышленные реализации поддерживают только двухфазную фиксацию транзакций, а в этом режиме не все типы отказов разрешимы. Распределенный deadlock (взаимоблокировка. - Прим. ред.) также может представлять значительную проблему, и большинство реализаций используют только таймауты для его детектирования.

Как бы то ни было, стратегия распределения данных должна определяться не политикой предприятия (что обычно пытается навязать руководство), а требованиями надежности и производительности системы. При построении распределенной базы данных следует уделить особое внимание проектированию топологии сети. Узлы распределенной базы данных должны быть соединены скоростными надежными линями связи. Это же касается линий соединения узлов базы данных и серверов приложений. Наличие низкоскоростного и ненадежного канала связи между узлами распределенной базы данных резко повышает количество детектируемых отказов сети.

В распределенной базе данных могут быть использованы следующие типы вызовов:

* Удаленные DDL- и DML- операции, а также выборка данных.
* Синхронные удаленные вызовы процедур.
* Асинхронные удаленные вызовы процедур.
* Непротиворечивые снимки.
* Асинхронная симметричная репликация.
* Синхронная симметричная репликация.
* Вызов распределенного запроса (запрашивает данные на чтение и модификацию с нескольких узлов).

Распределенная база данных может обеспечить горизонтальную фрагментацию; например, в филиале чаще всего используют данные о клиентах, находящихся в городе N (CLIENT\_PLACE = ‘N’). В этом случае на узле распределенной базы данных этого филиала может быть расположен фрагмент таблицы данных, выделенный согласно условию CLIENT\_PLACE = ‘N’.

Стратегия распределения данных для каждой СУБД определяется по-своему и достойна отдельной книги. Определение стратегии преследует две цели: сократить нагрузку на сеть и сервер и/или повысить уровень готовности данных.

Следует отметить, что проектировщики должны четко представлять себе особенности реализации распределенной базы данных используемой СУБД. Не владеющий этой информацией проектировщик рискует создать нерабочую или плохо работающую схему, причем проявится это даже не на моделях, а в реальной эксплуатации. Если вы решили строить распределенную базу данных, позаботьтесь о наличии в штате квалифицированного администратора. Есть одно простое правило: проектировщик не может принять окончательного решения об определении стратегии распределения данных без администратора баз данных. Редко встречаются проектировщики, которые имеют практический опыт администрирования 2-3 серверов баз данных и которые реально работали на них с распределенными базами данных. Для каждой СУБД принципы, влияющие на детали распределения базы данных, индивидуальны.

Особый интерес представляет неоднородная распределенная база данных. Это то, что постоянно, но безуспешно пытаются изживать и проектировщики, и администраторы баз данных. Никто не любит иметь дело со шлюзами данных. Но если неоднородной распределенной базы избежать не удалось, уделите особое внимание выбору шлюзов данных, а также ПО, обеспечивающему синхронизацию информации базах данных под управлением разных СУБД. Если синхронизация данных на узлах под управлением одной СУБД может быть обеспечена самой СУБД (что реализуется большинством производителей), то синхронизация данных на узлах под управлением разных СУБД обеспечивается, как правило, специальным ПО. Тщательно оцените, что будет дешевле: использовать это ПО или импортировать данные и схему и сделать распределенную базу данных однородной.

Следует также отметить, что в подавляющем большинстве реализаций каждый из узлов распределенной базы данных может обслуживаться параллельным сервером баз данных, и это является важным критерием повышения производительности системы. Промышленные СУБД также используют параллельный доступ к хранилищам данных; RAID, например, могут размещать данные, используя не файловую систему операционной системы, а свою файловую систему. Реализация всех этих возможностей существенно различается у различных СУБД. Это означает, что выбор параллельного сервера баз данных должен осуществляться только после серьезных консультаций с администраторами тех СУБД, которые рассматриваются как возможные претенденты на роль сервера баз данных в проекте.

Метод распределения данных по дискам для поддержки параллельных вычислений во многом зависит и от особенностей реализации СУБД. Администратор базы данных не может выбрать такой метод в отрыве от особенностей конкретной информационной системы. Еще одна возможность параллельной обработки данных, предоставляемая СУБД: обработка одного запроса несколькими менеджерами ресурсов. В реализациях также имеется возможность использования одного хранилища данных несколькими серверами баз данных (Parallel Server). Такая архитектура может быть эффективно использована на кластерах.

## Хранилище данных

Информационные системы такого вида сегодня одни из самых актуальных. Хранилище данных характеризуется большим объемом редко изменяемой информации. Хранилище данных обменивается информацией с другими хранилищами; как правило, речь идет о пакетной загрузке. Здесь решаются задачи обработки и хранения больших объемов информации (VLDB, Very Large Databases). Хранилище данных формирует сводное представление обо всех данных информационной системы.

В проектировании хранилищ данных важное место занимают интерфейсы обмена данными и интерфейсы конвертации данных. Следует четко выделить те системы, которые поставляют данные в хранилище, и те, что забирают данные из хранилища.

Внутренняя структура хранилища рассчитана на выполнение сложных запросов. Как правило, хранилище данных обслуживается выделенным сервером, а нередко - параллельным сервером баз данных. Хранилища данных используются для прогнозирования развития бизнеса. Они оснащаются средствами аналитической обработки данных (OLAP, On-Line Analytical Processing). На данный момент существуют специально ориентированные на OLAP версии серверов баз данных.

ПО хранилища баз данных состоит из следующих компонентов (рис. 2):

* ПО источников данных.

Здесь формируются входные пакеты данных, которые в состоянии интерпретировать конвертор и загрузчик данных. Это ПО входит в состав внешних систем (если повезло) или разрабатывается специально; в последнем случае основная его задача - достать данные из внешней системы.

* Загрузочная секция (отвечает и за трансформацию данных).

ПО принимает входные пакеты данных в необработанном формате, проверяет целостность пакета (например, контрольную сумму и т.п.), выполняет первичную обработку пакета (например, расшифровку), а также переводит данные во внутренний формат. Теперь они готовы для передачи в хранилище. Данные загрузочной секции хранятся во внутреннем формате системы, обеспечивающей пересылку данных.

* ПО трансформации данных.

Здесь обеспечиваются пересылка данных в хранилище и логика обработки данных. Данные проходит проверку на корректность, переводятся в нужный формат и интегрируются в хранилище.

* Собственно хранилище данных.

Преимущественно это один или несколько мощных параллельных серверов баз данных, которые обеспечивают обработку информации в хранилище. Само хранилище может быть распределенной среды (распределенная база данных) или трехуровневой среды (централизованная или распределенная база данных и сервер приложений). Собственно база данных хранилища редко представляет собой объединение рабочих баз данных. База данных не обязательно должна быть реляционной. Здесь могут с успехом использоваться расширения, предоставляемые современными СУБД, например Spatial Data-расширения, которые поддерживают многомерные данные.

* Интерфейс клиента.

ПО этого уровня обеспечивает взаимодействие приложений-клиентов, запрашивающих информацию из хранилища. Здесь запрещены операции модификации данных.

Рассмотрим один из приемов проектирования хранилищ данных.

Звездообразная схема (она же многомерная схема, соединение по схеме «звезда», куб данных) состоит из центральной таблицы фактов и нескольких таблиц измерений (рис. 3).

Факты представляют собой основные виды деятельности и факторы, влияющие на бизнес. В таблице измерений хранятся данные, способные оказывать определенное влияние либо порождать те или иные пути развития фактов.

Часто таблица фактов разбивается на секции для повышения скорости доступа к хранящимся в ней записям в зависимости от запроса, поскольку оптимизатор в этом случае сканирует только часть таблицы/индекса (определенную секцию), а не весь объем данных. Отношение между таблицей фактов и таблицей измерения должно быть как можно более простым, так как должен существовать единственно возможный путь их соединения.