Технологии создания прикладных программ для распределенных вычислений

А.Ю. Молчанов (НПП "СпецТек")

Показано, что на рынке существуют системы программирования, поддерживающие две самые распространенные технологии создания прикладных программ, функционирующие в условиях распределенных вычислений. Это технология клиент-сервер, уже довольно давно завоевавшая место на рынке, и относительно новая технология создания многоуровневых (multi-tier) приложений. Приводятся особенности, преимущества и недостатки разработки ПО, а также принципы работы приложений в архитектуре клиент-сервер.

Введение

В настоящее время получило широкое распространение прикладное ПО, построенное на основе технологии распределенных вычислений. В этом случае вся прикладная программа разбивается на несколько уровней, каждый из которых может выполняться на отдельном компьютере. Для работы всех программных составляющих как единого целого компьютеры, задействованные в вычислениях, объединяются в ЛВС, а иногда и в глобальную. Через сеть осуществляется их взаимодействие и обмен данными по ходу вычислений.

Организация распределенных вычислений сама по себе представляет непростую задачу. Она предусматривает создание соответствующих технологий разработки прикладных программ (приложений), ориентированных на функционирование в ее рамках. Эти технологии не могут существовать сами по себе – они должны поддерживаться разработчиками прикладных программ, производителями ОС и средств разработки (систем программирования).

Область применения распределенных вычислений весьма широка: от полуфантастических проектов поиска внеземных цивилизаций на основе обработки принимаемых из космоса радиосигналов до построения информационных систем управления. Опыт НПП "СпецТек" в области разработки АСУ на базе программного комплекса TRIM позволяет сделать некоторые выводы о технологиях создания прикладных программ для распределенных вычислений.

Сейчас на рынке существуют системы программирования, поддерживающие две самых распространенных технологии создания прикладных программ, функционирующих в условиях распределенных вычислений. Это технология клиент-сервер, уже довольно давно завоевавшая место на рынке, и относительно новая технология создания многоуровневых (multi-tier) приложений.

Вниманию читателей предлагается статья, посвященная рассмотрению основных принципов, лежащих в основе технологии клиент-сервер. Многоуровневая архитектура, а также различия, преимущества и недостатки двух указанных технологий будут рассмотрены в следующей публикации.

Архитектура "клиент-сервер": разработка ПО и принципы работы приложений

Распространение динамически загружаемых библиотек и ресурсов пользовательского интерфейса программ привело к ситуации, когда прикладные программы стали представлять собой не единый программный модуль, а набор взаимосвязанных компонентов. Причем не все компоненты создавались теми же разработчиками, что и сама прикладная программа. Некоторые из них входили в состав ОС, другие поставлялись сторонними разработчиками, которые очень часто могли быть никак не связаны с разработчиками прикладной программы.

При этом любую прикладную программу (приложение) можно условно разделить на четыре основных уровня обработки данных:

- пользовательский интерфейс, отвечающий за отображение данных, представление их пользователю и взаимодействие с ним;
- бизнес-логики, реализующий специализированную логику обработки и преобразования данных, характерную для данной прикладной программы;
- БД, отвечающий за типовые операции получения, хранения, защиты и архивации данных, разделение доступа к ним;
- файловых операций, обеспечивающий работу прикладной программы с файловой системой ОС.

Каждый из этих уровней представляет собой логически цельную составляющую единой прикладной программы. При распределенной обработке данных каждая составляющая может выполняться отдельно (при условии сохранения взаимосвязи между всеми составляющими). В принципе, каждый из этих уровней можно далее разбить на подуровни и получить больше составляющих, общее число которых ничем принципиально не ограничено. Но с другой стороны, все составляющие любой прикладной программы можно разделить всего на две крупные части.

Первая из них обеспечивает нижний уровень работы приложения, отвечает за методы хранения, резервирования, доступа и разделения данных. Вторая организует верхний уровень работы приложения, включает логику обработки данных и интерфейс пользователя. Первая часть обеспечивает два нижних

уровня обработки данных. Она, как правило, представляет собой набор компонент, входящих в состав ОС, либо созданных крупными сторонними фирмами-разработчиками, никак не связанными с созданием прикладной программы. Вторая часть обеспечивает реализацию двух верхних уровней обработки данных. Она включает в себя алгоритмы, логику и интерфейс пользователя, созданные разработчиками прикладной программы.

Таким образом, сложилось понятие приложения, построенного на основе архитектуры клиент-сервер. Первая часть в этой архитектуре стала носить название сервер данных, а вторая – клиент или клиентское приложение. При этом первая (серверная) часть приложения обеспечивает обработку данных на уровне файлов и БД. Для работы ее компонентов зачастую требуется наличие высокопроизводительной вычислительной системы. Вторая (клиентская) часть приложения, получая данные от сервера данных, обеспечивает их обработку и отображение в интерфейсе пользователя. По командам клиентской части сервер данных выполняет их добавление, обновление и удаление. Требования к вычислительным ресурсам, необходимым для выполнения компонент второй части, обычно существенно ниже, чем для первой.

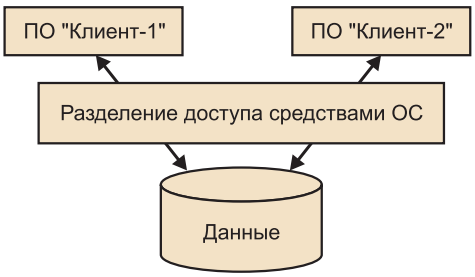

Рис. 1. Схема доступа к данным для приложений типа "файл-сервер"

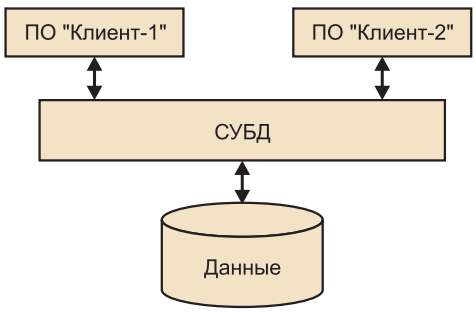

Рис. 2. Схема доступа к данным для приложений в архитектуре клиент-сервер

На начальном этапе развития архитектуры клиент-сервер доступ к данным сервера осуществлялся на уровне файлов, а разделение доступа к файлам обеспечивалось средствами ОС. Сервер данных выполнял только примитивные процедуры хранения, копирования и защиты файлов от несанкционированного доступа. Такие приложения иногда называют приложениями, построенными по принципу "файл-сервер". В общем виде схема их работы представлена на рис. 1.

Видно, что в этом случае серверная часть выполняет только один, самый нижний уровень обработки данных, остальные уровни реализуются на клиентской части. Поэтому приложения типа "файл-сервер" можно только весьма условно отнести к приложениям, построенным в архитектуре "клиент-сервер". Хотя в них присутствует серверная компонента, ее функции, в основном, выполняются в ОС.

Главными недостатками приложений, построенных по принципу "файл-сервер", являются:

- высокая нагрузка на клиентскую часть, и как следствие, высокие требования к вычислительным ресурсам клиентской части;
- невозможность эффективно разделить доступ к данным при их одновременном использовании несколькими пользователями;
- невозможность организовать защиту данных иначе, как на уровне доступа ОС;
- высокая нагрузка на сеть для передачи файлов, если сервер данных и клиентское приложение работают на разных компьютерах в составе сети.

Постепенно с развитием архитектуры клиент-сервер на рынке стали появляться компании, специализирующиеся на производстве серверов данных, функциональные возможности которых стали расширяться. В то же время, разработчики прикладных программ чаще используют серверы данных от известных производителей. Эту тенденцию не могли не отметить производители средств разработки, и на рынке появились системы программирования, специально ориентированные на разработку приложений, работающих в архитектуре клиент-сервер. В итоге сложились все условия для выхода в свет прикладных программ, построенных на основе полноценной архитектуры типа клиент-сервер.

Как правило, разработчики, создающие ПО в архитектуре клиент-сервер, разрабатывают именно клиентскую часть. В качестве сервера данных чаще всего выбирается сервер одного из известных производителей. Реже разрабатываются обе составляющих – и сервер данных, и клиентские приложения. Но тогда с одним сервером данных работают несколько различных клиентских приложений (иначе нет никакого смысла создавать ПО в данной архитектуре).

Проще всего полноценное приложение в архитектуре клиент-сервер возможно получить, если использовать БД для хранения данных и СУБД для доступа к данным. Схема работы такого приложения представлена на рис. 2.

Здесь серверная компонента реализует два уровня обработки данных – файловые операции и работу с БД. Все функции по хранению данных, доступу к ним, защите и резервному копированию данных в такой схеме реализует СУБД на сервере. Такой подход освобождает клиентские рабочие места от реализации этих функций и снижает требования к ним. Клиентское приложение не работает непосредственно с файлами и данными, оно обращается к СУБД с запросами на получение (просмотр) данных, их добавление, модификацию и удаление. При работе сервера данных и клиентского приложения на разных компьютерах в составе сети через сеть будут передаваться не все данные из

БД, а только запросы клиента и ответы СУБД на них. Таким образом, в архитектуре клиент-сервер снижается нагрузка на сеть при обмене данными.

Полноценное приложение, созданное в архитектуре клиент-сервер, решает все проблемы, связанные с недостатками приложений типа "файл-сервер", но требует наличия СУБД и развитых средств взаимодействия с ней. По мере развития данной архитектуры на рынке стали появляться СУБД различных типов. Все СУБД можно разделить на три основных типа: реляционные, сетевые (не путать с распределенными БД) и объектные (объектно-ориентированные). Кроме того, СУБД подразделяются по способам доступа и хранения информации.

Со временем на рынке СУБД стали доминировать несколько наиболее известных компаний-производителей. Сейчас на нем доступен широкий спектр различных серверов данных от самых простейших настольных СУБД, практически мало чем отличающихся от приложений типа "файл-сервер" (PARADOX, Microsoft Access) до промышленных СУБД с развитыми средствами управления данными (Sybase, Microsoft SQL Server, ORACLE). Компании-производители СУБД предлагают стандартизованные интерфейсы для доступа к создаваемым ими СУБД. На них, в свою очередь, стали ориентироваться и разработчики прикладных программ.

Современные сервера данных. Язык запросов данных

В настоящее время наибольшее распространение на рынке СУБД получили реляционные БД. В качестве примеров таких СУБД, рассчитанных на применение в качестве промышленных серверов хранения данных (вплоть до корпоративного уровня), можно назвать Sybase, Microsoft SQL Server, Oracle, DB2. Принципы, лежащие в основе реляционных СУБД, стали определяющими в выборе методов взаимодействия между клиентской и серверной частью в архитектуре клиент-сервер. Этих базовых принципов стараются придерживаться все ведущие разработчики СУБД, в том числе и СУБД, построенных на основании иных технологий, чем реляционные БД.

Главным фактором, определяющим взаимодействие клиентской части приложения и сервера данных, стал механизм запросов, с которыми клиентское приложение обращается к серверу. Этот механизм должен предоставить клиенту средства, которые давали бы ему возможность определить, какие данные он хочет получить от сервера и какие операции с ними выполнить. Для этой цели был предложен специальный язык описания запросов – SQL (Structured Query Language, язык структурированных запросов). По мере развития языка SQL он был признан всеми разработчиками серверов данных и со временем появились международные стандарты языка запросов (например, известный стандарт SQL 92, который в настоящее время поддерживается всеми серверами данных от известных производителей).

Язык SQL предоставляет средства для выполнения четырех основных операций с данными на сервере: выборка (SELECT), добавление (INSERT), обновление (UPDATE) и удаление (DELETE). Кроме этих основных операций существуют другие вспомогательные, связанные с распределением прав доступа и прочими функциями сервера данных. Каждая операция описывается соответствующим предложением языка, построенным с учетом его синтаксических и семантических правил. Клиентское приложение, желающее выполнить какое-либо действие с данными, должно сформулировать его в виде предложения языка SQL, направить запрос с этим предложением серверу данных и дождаться ответа от него, в котором будет сформулирован результат выполнения операции. Если действие не может быть выполнено с помощью одной операции языка SQL, оно разбивается на несколько операций, которые выполняются последовательно.

Другим важным механизмом, определяющим взаимодействие клиентской и серверной части, является механизм транзакций. Его суть заключается в том, что клиентское приложение может определить группу взаимосвязанных действий над данными, которые либо должны быть выполнены все вместе, либо не выполнены вообще. Взаимосвязь действий определяется логикой клиентской части. Начав выполнение этой группы действий (открыв транзакцию), клиент последовательно передает серверу данных все команды в виде предложений языка SQL и ожидает результатов. При правильности выполнения всех команд клиентская часть подтверждает это (закрывает транзакцию), и только после этого операция считается выполненной и все сделанные изменения вносятся в данные на сервере. Если же по каким-то причинам операция не могла быть выполнена, клиент отказывается от нее (отменяет транзакцию) и сервер должен восстановить то состояние данных, которое было до начала транзакции.

Взаимодействие клиентской и серверной части приложения в архитектуре "клиент-сервер" описано только в самом общем виде. О языке SQL, механизме транзакций и других средствах современных СУБД можно узнать из книг, посвященным современным БД [1,2].

На стороне сервера каждое предложение SQL должно интерпретироваться в последовательность операций, выполняемых сервером. Получив SQL-запрос от клиента, сервер распознает его на основе грамматики SQL, определяющей синтаксис запроса, а также проверяет семантику, основываясь на структуре БД, которая известна серверу данных. Если запрос содержит синтаксические или семантические ошибки, сервер сообщает об этом клиенту. Если же запрос правильный, на его основе сервер строит схему выполнения запроса, которую затем выполняет и передает результат ее выполнения клиенту.

Видно, что схема обработки и выполнения SQL-запросов на сервере данных полностью соответствует принципам, на основе которых функционируют интерпретаторы и компиляторы языков программиро-

вания. С этой точки зрения язык SQL является полноправным компьютерным языком (хотя нельзя сказать, что он является языком программирования).

Видно, что этой схеме присущ недостаток, который связан со всеми интерпретируемыми языками – каждый раз сервер данных вынужден тратить время на распознавание запроса и проверки его правильности. В том случае, когда клиент направляет серверу часто повторяющиеся типовые запросы, это ведет к неэффективным затратам времени. Эффективность схемы была бы выше, если бы сервер данных мог работать по схеме компилятора – один раз распознавать запрос, а потом выполнять его каждый раз при обращении клиента. Разработчики СУБД обратили на это внимание, и в современных серверах данных присутствуют соответствующие решения в виде запросов с параметрами, предопределенных выборок (VIEW) и хранимых процедур сервера [2].

Используя стандартизованный язык SQL, разработчик клиентской части приложения, работающего в архитектуре клиент-сервер, может строить запросы к серверу данных, не ориентируясь на определенный тип СУБД. При этом взаимодействие клиентской части с сервером на уровне запросов не будет зависеть от типа используемого сервера данных (СУБД). Однако остается открытым вопрос о том, как SQL-запросы будут передаваться от клиента серверу, и каким образом клиент будет получать ответы.

Наиболее очевидным решением было решение об использовании для этой цели динамически загружаемых библиотек. Клиентское приложение обращается к функции соответствующей библиотеки и передает ей запрос к серверу в виде предложения на языке SQL. В задачу библиотеки входит установить связь с сервером, передать ему запрос, дождаться ответа и вернуть результат клиенту в качестве результата выполнения функции. Однако такое простое решение обладает одним существенным недостатком: для каждого типа сервера должна использоваться своя динамически загружаемая библиотека со своими функциями, что делает клиентское приложение зависимым от типа СУБД.

В настоящее время на рынке СУБД доминируют несколько крупных производителей, среди которых всегда можно выбрать сервер данных, удовлетворяющий тем или иным требованиям, предъявляемым для решения прикладной задачи. Поэтому зачастую разработчики клиентской части ПО, работающего в архитектуре клиент-сервер, пренебрегают указанным недостатком, используя описанную выше схему для взаимодействия с сервером данных. Они разрабатывают свои приложения, ориентируясь на определенный тип СУБД. Этому способствует тот факт, что многие фирмы-производители СУБД (такие, например, как Oracle или Microsoft) предлагают не только сервера данных, но и системы программирования, ориентированные на работу с конкретным типом СУБД. Такой путь имеет свои преимущества, так как прямое обращение к библиотеке, взаимодействующей с сервером данных, дает наибольшую эффективность по скорости выполнения запросов клиента.

Однако зависимость от типа СУБД не всегда допустима для клиентской части, особенно в том случае, когда разработчики стремятся добиться переносимости своего ПО. Для этой цели служат универсальные интерфейсы взаимодействия клиентской и серверной части в архитектуре "клиент-сервер". Существует несколько вариантов таких интерфейсов от разных производителей. Одним из наиболее распространенных средств, позволяющих унифицировать организацию взаимодействия с различными СУБД, является интерфейс ODBC. Доступ к БД осуществляется при помощи специального ODBC драйвера, который транслирует запросы к БД на язык, поддерживаемый конкретной СУБД.

Для установления соединения с БД технология ODBC использует ODBC драйверы и источники данных, которые позволяют настроиться на сеанс конкретного пользователя СУБД. Для этого источник содержит имя пользователя и его пароль, а также, при необходимости, другую информацию, требуемую для присоединения к СУБД. ODBC драйвер представляет собой динамически загружаемую библиотеку, которая может использоваться приложением для получения доступа к конкретному серверу данных. Каждому типу СУБД соответствует свой ODBC драйвер. Система ODBC также включает ряд служебных утилит.

Взаимодействие клиентской части ПО с СУБД осуществляется при помощи набора системных вызовов, которые выполняются по отношению к источнику данных. Источник данных взаимодействует с драйвером ODBC, транслирует ему запросы клиента и получает ответы, а тот в свою очередь обращается к СУБД. При необходимости работать с типом СУБД достаточно указать другой тип ODBC драйвера в источнике данных, при этом не требуется изменять клиентскую часть ПО.

Такая двухступенчатая схема взаимодействия клиента с сервером данных обеспечивает независимость клиентской части от типа СУБД на сервере данных, но в целом снижает эффективность работы приложения, поскольку запрос, посланный клиентом, дважды передается различным библиотекам функций прежде, чем попадет на сервер. Решение вопроса о выборе способа взаимодействия с сервером данных остается за разработчиком клиентской части ПО и зависит от предъявляемых к нему требований.

Преимущества и недостатки архитектуры клиент-сервер

Ситуация, связанная с распространением на рынке приложений, построенных на основе архитектуры клиент-сервер, оказала свое влияние и на структуру систем программирования. Многие из них стали предлагать средства, ориентированные на создание приложений в архитектуре клиент-сервер. Как правило, эти средства поставляются в составе системы программирования и поддерживают возможность работы с широким диапазоном известных СУБД через

один или несколько доступных интерфейсов обмена данными. Разработчик прикладной программы выбирает одно из доступных средств плюс возможный тип СУБД (или несколько возможных типов), и тогда его задача сводится только к созданию клиентской части приложения, построенной на основе выбранного интерфейса.

Создав клиентскую часть, разработчик может далее использовать и распространять ее только в комплексе с соответствующими средствами из состава системы программирования. Интерфейс обмена данными обычно входит в состав системы программирования или в состав ОС. Большинство систем программирования предоставляют возможность распространения средств доступа к серверной части без каких-либо дополнительных ограничений.

Что касается самой серверной части, то возможны два пути: простейшие СУБД требуют от разработчиков приобретения лицензий на средства создания и отладки БД, но часто позволяют распространять результаты работы без дополнительных ограничений; мощные промышленные СУБД, ориентированные на работу десятков и сотен пользователей, требуют приобретения лицензий как на создание, так и на распространение серверной части приложения. В этом случае конечный пользователь приложения получает целый комплекс программных продуктов от множества разработчиков.

По сравнению с ранее существовавшими приложениями, приложения, построенные на основе архитектуры клиент-сервер обеспечивали своим разработчикам ряд преимуществ:

- разработчики приложений в архитектуре клиент-сервер избавлены от необходимости самостоятельно создавать средства для хранения данных, разделения доступа к ним, защиты и резервного копирования данных – все эти функции берет на себя СУБД;
- СУБД обеспечивает высоко надежные механизмы разделения доступа к данным и защиты их от несанкционированного доступа, удовлетворяющие общепризнанным стандартам;
- все функции по управлению данными выполняются на сервере данных, что снижает требования к вычислительным ресурсам рабочих станций, на которых выполняются клиентские приложения;
- для обмена данными между клиентским приложением и сервером данных через сеть передаются не все данные, а только запросы клиента и ответы сервера, что снижает нагрузку на сеть;
- при необходимости увеличить число клиентов в системе достаточно включить в сеть новые рабочие станции, увеличить мощность сервера и пропускную способность сети (если требуется) – нет необходимости обновлять программное и аппаратное обеспечение уже существующих рабочих станций.

В то же время, сама по себе архитектура клиент-сервер не лишена некоторых недостатков:

- функции управления данными возложены на сервер данных, но обработка данных по-прежнему выполняется клиентскими приложениями, что не позволяет существенно снизить требования к ним, если логика системы предусматривает достаточно сложные манипуляции с данными;
- при необходимости изменить или дополнить логику обработки данных необходимо выполнить обновление клиентских приложений на всех рабочих местах, что может быть достаточно трудоемко;
- если необходимо изменить только внешний вид интерфейса пользователя (отображение данных), но оставить неизменной логику обработки данных, при этом чаще всего требуется заново создать и установить на рабочем месте новый вариант клиентской части системы;
- при использовании мощной промышленной СУБД требуется наличие лицензии на подключение к СУБД для каждого рабочего места, где установлена клиентская часть ПО.

Для устранения этих недостатков была предложена следующая модель разработки приложений – трехуровневая, а в общем случае, многоуровневая архитектура.

Список литературы

1. *Олифер В.Г., Олифер Н.А.* Сетевые операционные системы / СПб.: Питер. 2002.
2. *Карпова Т.С.* Базы данных: модели, разработка, реализация /СПб.: Питер. 2001.

Молчанов Алексей Юрьевич *– канд. техн. наук, директор по разработкам НПП "СпецТек".*

Контактный телефон (812) 329-45-60.

E-mail: mill@spectec.ru Http://www.trim.ru