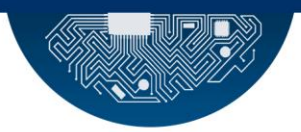
СПбГЭТУ «ЛЭТИ»

ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ



Е.Ю.Белова

Разработка приложений в распределенной среде

Конспект лекций

СПбГЭТУ «ЛЭТИ», 2022 г.

1 РАСПРЕДЕЛЕННАЯ СИСТЕМА. ПОНЯТИЕ, ХАРАКТЕРИСТИКИ, КЛАССЫ

1.1 Сферы применения распределенных систем

Представьте ситуацию, при которой некоторые части вашего программного обеспечения работают в браузере, другие – на одном или нескольких серверах. В тоже самое время серверы функционируют или в вашем корпоративном центре обработки данных, или в крупнейшем центре обработки данных China Telecom Data Centre в Китае, или же в любом другом месте. Все эти фрагменты взаимодействуют друг с другом по сети, возможно, через Интернет, и, вероятно, данные вашего программного обеспечения также являются широко распределенными.

Иными словами, почти каждое современное разрабатываемое приложение, будь то мобильная клиентская программа или картографический сервис, должно являться распределенной системой, которая работает на многих машинах и к которой получает доступ множество пользователей по всему миру через сеть. Такой подход позволяет обеспечить необходимый уровень надежности, гибкости и масштабируемости, ожидаемый от современных компьютерных программ.

Распределенные системы широко используются в различных сферах деятельности человека. Ниже рассмотрены примеры их применения.

✓ Финансы и коммерция. Электронная коммерция (Amazon, eBay, AliExpress,Wildberries, Ozon, Citilink, Labirint) и связанные с ними платформы онлайн-платежей (PayPal, Webmoney, QIWI, YooMoney); платежные агрегаторы (RBK Money, Robokassa, PayKeeper, ЮKassa), инвестирование и трейдинг (FinamTrade, Мой брокер (БКС), Alpari invest); интернет-банкинг.

✓ Информационные сообщества. Поисковые Web-машины (Google, Yandex); цифровые библиотеки (Google Books, Электронная библиотека РГБ, eLIBRARY); социальные сети (Facebook, ВКонтакте, Instagram), видеостриминговые сервисы (Okko, Ivi, Кинопоиск HD).

✓ Здравоохранение. Электронные медицинские карты, телемедицина (удаленные консультации и системы мониторинга состояния здоровья пациента).

✓ Образование. Образовательные онлайн-платформы (Skillbox, Coursera, Stepik), платформы и сервисы для сопровождения педагогического процесса (Google Classroom, Moodle), сервисы для проведения видеоконференций (Zoom, Skype, Microsoft Teams).

✓ Транспорт и логистика. Системы управления современными транспортными средствами; картографические сервисы (Google Maps, Яндекс.Карты, 2gis).

✓ Наука. Использование сложных сетей компьютеров для хранения, анализа и обработки научных данных и организации совместной работы между группами ученых.

✓ Взаимодействие с окружающей средой. Сенсорные технологий для мониторинга природной среды (Всемирный индекс качества воздуха, карта загрязнения океана пластиком); предупреждения о природных бедствиях (Карта наводнений, Карта пожаров СКАНЭКС).

1.2 Влияние развития различных областей информационных технологий на появление распределенных систем

Создание распределенных систем, описанных выше, стало возможным из-за достижений в областях вычислительной техники и телекоммуникаций, а именно появления больших интегральных схем (СБИС) и компьютерных сетей соответственно. Результатом прорыва в технологии СБИС стало создание микропроцессора в начале 1970-х годов, а переход к микропроцессорам привел к появлению персональных компьютеров. Первый общедоступный микропроцессор имел 24-битную адресацию, в настоящее время разрядность микропроцессора достигает 64 бит. Создание в 2000-х годах многоядерного процессора позволило одновременно выполнять несколько задач за счет их распараллеливания на несколько потоков или процессов.

1.3 Понятие распределенной системы

«... система, в которой сбой какого-то компьютера, о существовании которого вы даже не подозревали, может сделать ваш компьютер бесполезным». Лесли Лэмпорт, американский ученый, первый лауреат премии Дейкстры.

«... система, которая представляет собой совокупность автономных вычислительных элементов и является единой связанной системой для ее пользователей». Эндрю Стюарт Тененбаум, американский ученый, профессор, автор множества книг для высшей школы в области информатики и вычислительной техники.

Автономным вычислительным элементом в данном случае является узел, который работает независимо от других вычислительных элементов. Узел представляет собой персональный компьютер, сервер, смартфон, промышленную систему управления, датчик или иной тип коммуникационного вычислительного устройства.

Из прошлых дисциплин вам скорее всего знакома модель параллелизма с общей (разделяемой) памятью. Она описывает параллелизм между несколькими процессами или потоками, работающими на одном и том же компьютере. Причем несколько потоков, запущенные в одном процессе, имеют доступ к одному и тому же адресному пространству. При этом имеется возможность для передачи данных из одного потока в другой: переменная или указатель, действительный для одного потока, также действителен для другого.

При рассмотрении распределенных систем параллелизм тоже существует, поскольку узлы выполняют программы параллельно. Однако разделяемая память отсутствует. Узел запускает свою собственную операционную систему со своим собственным адресным пространством, используя память, встроенную в данный узел. Узлы в распределенной системе взаимодействуют друг с другом только путем обмена сообщениями между ними.

✓ Временна́я последовательность взаимодействия независимых узлов определяется каждым из узлов самостоятельно.

✓ Управление совокупностью узлов реализуется путем регистрации узлов, входящих и не входящих в распределенную систему.

✓ Организации совокупности узлов чаще всего происходит в виде оверлейной сети. В данном типе сети узел является программным процессом, имеющим список других процессов, которым он может отправлять сообщения напрямую.

На практике представить распределенную системы в виде единой связанной системы достаточно сложно. Решением проблемы является стремление к реализации такой системы с готовностью к тому, что в любое время какая-то ее часть может отказать.

1.4 Роль промежуточного программного обеспечения

Объединение распределенной системы достигается за счет введения связующего программного слоя – промежуточного программного обеспечения (ПО). Он объединяет автономные вычислительные элементы в единую систему, маскируя их гетерогенность, и создает среду для выполнения распределенных приложений.

Распределенная система предоставляет средства для связи компонентов распределенного приложения В с приложениями А и С и позволяет им взаимодействовать друг с другом (Рис. 1). Одновременно она в разумных пределах скрывает различия в оборудовании и операционных системах каждого из приложений.

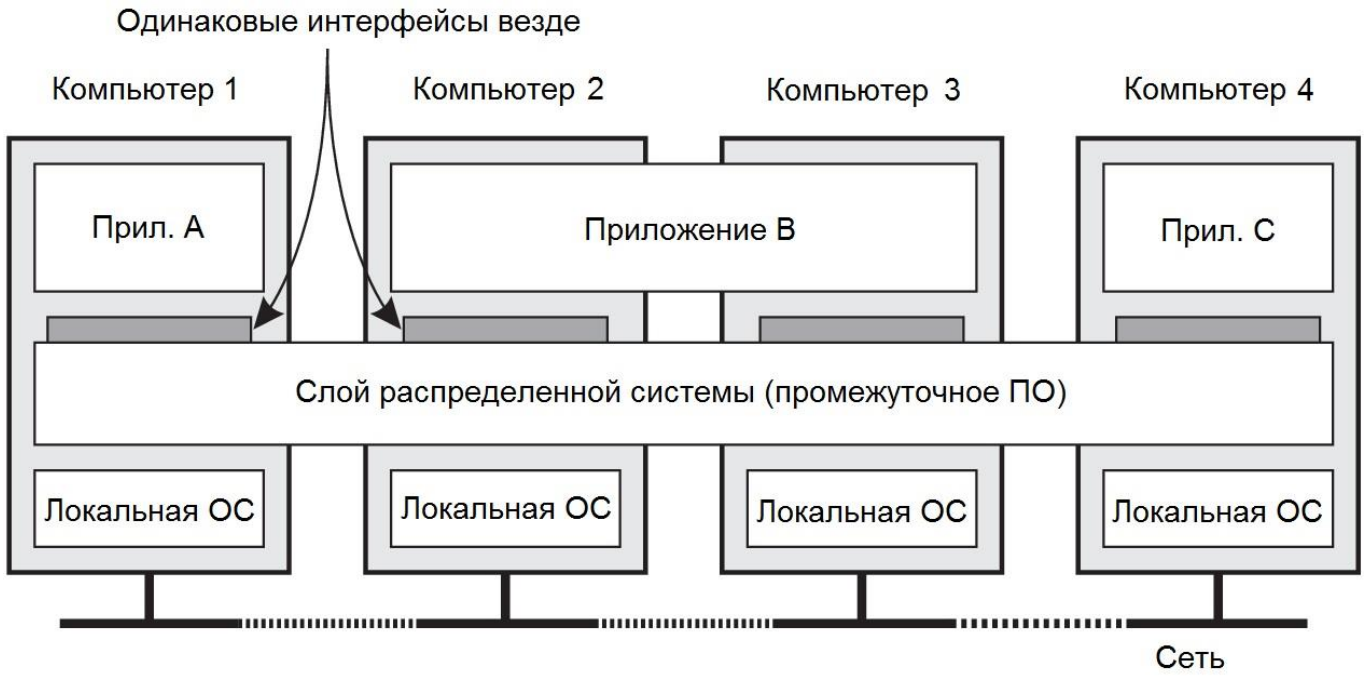


Рис. 1

Промежуточное ПО предоставляет в сетевом окружении средства для взаимодействия приложений, услуги управления ресурсами, безопасности, маскировки и восстановления после сбоев и т.д. Достаточно часто под

промежуточным ПО понимают хранилище часто используемых компонентов и функций, которое позволяет реализовывать приложения отдельно.

1.5 Причины создания распределенных систем

1. Система является распределенной по своей природе. Отправка мгновенных сообщений с одного смартфона на другой, предполагает, что данные устройства связаны одной сетью.

2. Обеспечение высокого уровня надежности. Если пользователи обращаются с запросами только к одному узлу, то при выходе из его строя, обслуживание приостанавливается. При наличии нескольких узлов, другой узел берет работу с пользователями на себя до устранения неисправности.

3. Повышение производительности. Если услуги сервиса предоставляются по всему миру, но пользователи обращаются только к одному узлу (например, в Париже), то скорость доступа к сервису может оказаться низкой. Если узлы размещены в нескольких точках мира, то пользователи направляются к ближайшему узлу. Скорость доступа увеличивается.

4. Решение больших проблем. Ряд задач по обработке данных или вычислениям слишком велики и сложны для выполнения на одном компьютере.

1.6 Характеристики распределенной системы

К основным характеристикам распределенной системы относят:

- ✓ поддержка совместного использования ресурсов;
- ✓ прозрачность;
- ✓ открытость;
- ✓ масштабируемость.

1.6.1 Поддержка совместного использования ресурсов

Упрощение доступа и совместного использования удаленных ресурсов для пользователей (и приложений) позволяет экономить финансовые средства, облегчает сотрудничество и обмен информацией между территориально распределенными группами людей.

1.6.2 Прозрачность

Прозрачность – это скрытие от конечных пользователей и приложений факта физического и территориального распределения процессов и ресурсов по нескольким компьютерам.

Формы прозрачности:

✓ прозрачность доступа – это скрытие различия в представлении данных и того, как получен доступ к объекту (т.е. процессу или ресурсу).

✓ прозрачность местоположения – это скрытие физического местоположения объекта

✓ прозрачность перемещения – это скрытие перемещения объекта в другое место в процессе его использования.

✓ прозрачность миграции – это скрытие перемещения объекта в другое место.

✓ прозрачность репликации – это скрытие тиражирования или дублирования объекта.

✓ прозрачность параллельности – это скрытие разделения объекта между несколькими независимыми пользователями.

✓ прозрачность отказа – это скрытие сбоя и восстановления объекта.

Достижение прозрачности системы во всех аспектах на практике невозможно. Более того, в некоторых случаях сознательное несоблюдение всех форм прозрачности позволяет разработчику и пользователю лучше понять поведение распределенной системы в различных ситуациях.

1.6.3 Открытость

Открытая распределенная система – это система, предлагающая компоненты, которые могут легко использоваться или интегрироваться в другие системы.

Компоненты должны следовать стандартным правилам, описывающим синтаксис и семантику услуги, которую они предлагают. Услуги определяются через интерфейсы, использующие язык определения интерфейса (Interface Definition Language, IDL). Определения интерфейсов описывают только синтаксис сервисов (имена функций, типы параметров, возвращаемые значения, исключения и пр). Семантика интерфейсов, а

именно определение того, что эти сервисы делают, предоставляется в неформальной форме с помощью естественного языка.

Цели открытой распределенной системы:

✓ Функциональная совместимость – это степень, в которой две реализации систем или компонентов разных производителей могут сосуществовать и работать совместно, полагаясь только на взаимные услуги в соответствии с общим стандартом.

✓ Портативность – это степень, в которой приложение распределенной системы A может работать без изменений в другой распределенной системе B, реализация интерфейсов в которой аналогична, представленной в A.

✓ Расширяемость – это легкость конфигурирования системы из разных компонентов возможно, от разных разработчиков и простота добавления новых компонентов или замены существующих без затрагивания остающихся компонентов.

1.6.4Масштабируемость

Масштабируемость системы может определяться как минимум по трем различным измерениям:

✓ масштабируемость размеров. Система, масштабируемая относительно ее размера, – это система, в которую можно легко добавить большее количество пользователей и ресурсов без заметной потери ее производительности;

✓ географическая масштабируемость. Географически масштабируемая система – это система, в которой пользователи и ресурсы могут находиться далеко друг от друга, но факт наличия значительных задержек в линии связи не должен быть замечен системой;

✓ административная масштабируемость. Административно масштабируемая система – это система, которая легко поддается управлению несмотря на то, что она охватывает много независимых административных организаций.

1.7 Классы распределенных систем

1. Высокопроизводительные распределенные вычисления
   ✓ Кластерные вычисления (cluster computing)

- ✓ Сетевые вычисления (grid computing)
- ✓ Облачные вычисления

2. Распределенные информационные системы
3. Распространенные системы
   - ✓ Повсеместно распространенные вычислительные системы
   - ✓ Мобильные вычислительные системы
   - ✓ Сенсорные сети

Высокопроизводительные распределенные вычисления – класс систем, который используется для обеспечения высокой производительности вычислительных задач.

В кластерных вычислениях базовое оборудование (суперкомпьютер) состоит из набора аналогичных рабочих станций или компьютеров, которые связаны между собой посредством высокоскоростной локальной сети. Все узлы имеют одну и ту же операционную систему и все подключены к одной сети. Кластерные вычисления используются во всех случаях параллельного программирования, в котором одна программа выполняется параллельно на нескольких машинах.

Сетевые вычисления базируются на распределенных системах, которые представляются в виде федерации компьютерных систем. Каждая система настроена для выполнения определенных задач. Она может отличаться от других систем в плане аппаратных средств, операционных систем, сетей, административных доменов, политики безопасности и т. д. Федерация компьютерных систем образуется в том случае, если создана виртуальная организация, обеспечивающая сотрудничество группы людей из разных учреждений. Процессы, принадлежащие одной виртуальной организации, имеют право доступа к предоставляемым этой организации ресурсам (серверам, системам хранения данных, базам данных и пр.).

В основе облачных вычислений лежит концепция утилитарных вычислений, с помощью которых клиент загружает задачи в центр обработки данных и оплачивает каждый ресурс. Облачные вычисления характеризуются легко используемым и доступным пулом динамически настраиваемых виртуальных ресурсов, которые обеспечивают основу для масштабируемости: если требуется выполнить больше работы, клиент просто приобретает больше ресурсов. Таким образом, они обеспечивают

наличие средств для динамичного построения инфраструктуры и объединения того, что необходимо для доступа к услугам.

Корпоративные информационные системы имеют инфраструктуру, в которую проще интегрировать приложения. При разработке промежуточного программного обеспечения для данных систем учитывается уровень интеграции. Интеграция на самом низком уровне позволяет клиентам упаковать несколько запросов, возможно для разных серверов, в один большой запрос и выполнить его как распределенную транзакцию. Основная идея заключается в том, что выполняются или все, или ни один из запросов. С течением времени выяснилось, что интеграция приложений также имеет право на существование. Это привело появлению отрасли, которая сосредоточена на корпоративных приложениях.

Распространенные системы предназначены для работы в естественной среде. Они содержат датчики (например, IoT-датчики), которые фиксируют различные аспекты поведения пользователя, и исполнительные механизмы, чтобы выявлять и возвращать различные аспекты поведения пользователей и управлять ими.

2 ТИПЫ ПРОЦЕССОВ И ИХ МОДЕЛИ ВЗАИМОДЕЙСТВИЯ В РАСПРЕДЕЛЕННЫХ СИСТЕМАХ

2.1 Понятие межпроцессного взаимодействия

Межпроцессное взаимодействие является основой распределенных систем. Пользовательские процессы запущены на хостах, которые подключены друг к другу через сеть. Сеть распространяет сигналы, которые передаются от одного процесса к другому.

Межпроцессное взаимодействие можно условно разделить на две части:

✓ Взгляд пользователей на взаимодействие. Пользовательские процессы имеют абстрактное высокоуровневое представление межпроцессного взаимодействия. Пользователю необязательно знать, как происходит передача данных. В большинстве распределенных систем используется модель клиент-сервер, но в ряде случаев применяют и модель одноранговой сети (peer-to-peer, P2P), основанной на равноправии клиентов и серверов.

✓ Работа в сети. В данной части рассматривается взаимодействие процессов через протоколы разных уровней. Процессы на разных машинах могут обмениваться информацией различными способами, которые основываются на передаче сообщений низкого уровня, предлагаемой базовой сетью. Наиболее часто используются две модели связи – удаленный вызов процедур (Remote Procedure Call, RPC) и промежуточное программное обеспечение, ориентированное на сообщения (Message-Oriented Middleware, MOM). RPC стремится скрыть большинство тонкостей передачи сообщений и идеально подходит для клиент-серверных приложений. При применении MOM обмен данными происходит почти так же, как в системах электронной почты.

2.2 Типы процессов в распределенных системах

2.2.1 Процессы и потоки

Первые компьютеры не имели операционных систем. Они выполняли единственную программу от начала до конца, которая имела прямой доступ ко всем ресурсам машины.

С появлением операционных систем появилась возможность выполнять несколько программ одновременно в рамках процессов – изолированных, независимо выполняемых программ, которые пользуются ресурсами операционной системы, такими как память, дескрипторы файлов и учетные данные системы безопасности. Операционная система создает несколько виртуальных процессоров, каждый из которых предназначен для запуска одной программы. Процесс является программой, которая в конкретный момент времени выполняется на одном из виртуальных процессоров операционной системы. Операционная система уделяет большое внимание обеспечению того, чтобы независимые процессы не могли злонамеренно или непреднамеренно повлиять на правильность поведения друг друга. Другими словами, тот факт, что несколько процессов могут одновременно использовать один и тот же процессор и другие аппаратные ресурсы, становится прозрачным.

Процессы могут взаимодействовать друг с другом с помощью коммуникационных механизмов: сокетов, обработчиков сигналов, совместной памяти, семафоров и файлов.

Последовательное программирование интуитивно понятно: человек решает одну задачу за один раз, выстраивая последовательность действий. Выключить будильник, надеть халат, приготовить кофе. Как и в программировании, любое из этих реальных действий (например приготовить кофе) является абстракцией для другой последовательности: открыть шкафчик, достать кофе, насыпать его в кофемашину, выбрать режим приготовления, включить кофемашину, и т. д. Некоторые шаги предусматривают асинхронность – пока готовиться кофе, вы можете сделать горячий бутерброд в микроволновке. Производители кофемашин и микроволновок знают, что их изделия часто используются асинхронно, поэтому снабжают их звуковым сигналом завершения задачи. Нахождение баланса между последовательностью и асинхронностью является критерием эффективности как для человека, так и для программы.

Поток выполнения – наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Потоки выполнения позволяют многочисленным программным потокам данных сосуществовать внутри процесса. Они совместно используют общепроцессные ресурсы, причем каждый поток выполнения имеет свой программный счетчик, стек и локальные переменные. Потоки внутри одной программы могут действовать одновременно в нескольких процессорах.

2.2.2Уровни распределенных приложений

Многие распределенные приложения служат для поддержки доступа пользователей или приложений к базам данных. Тогда в них выделяют три логических уровня:

- ✓ уровень интерфейса приложения – часть, которая обрабатывает взаимодействие с пользователем или каким-либо внешним приложением;
- ✓ уровень обработки – часть, которая содержит основные функциональные возможности приложения;
- ✓ уровень данных – часть, которая работает с базой данных или файловой системой.

Уровень обработки содержит не так много общих аспектов, как два других уровня. Его содержание можно прояснить на примере системы поддержки принятия решений для врача-травматолога.

Она разделяется на три уровня:

✓ интерфейс, реализующий пользовательский интерфейс или предлагающий интерфейс программирования для внешних приложений;

✓ интерфейс для доступа к базе данных с информацией о персональных данных, диагнозе, результатах лабораторных исследований и лучевой диагностики пациентов;

✓ программы анализа между двумя первыми уровнями.

Анализ медицинских данных подразумевает использования методов интеллектуального анализа данных и математической статистики. В некоторых случаях ядро системы поддержки принятия решений должно работать на высокопроизводительных компьютерах с целью достижения пропускной способности и ожидаемой пользователями скорости реагирования. Например, в случае, когда происходит мониторинг состояния здоровья человека в режиме реального времени с использованием носимых датчиков.

2.2.3 Способы организации распределенных систем

Распределенная система может быть организована различными способами. Одним из традиционных подходов является использование клиент-серверной архитектуры (Рис.2). Клиент – это процесс, который запрашивает услугу. Сервер – это процесс, реализующий конкретную службу. Клиент отправляет запрос на сервер, который затем выдает результат, возвращающийся клиенту. Данное взаимодействие проиллюстрировано на рис. 2. Путем размещения разных компонентов на разных машинах достигается естественное физическое распределение функций по совокупности машин.

Ранее говорилось о том, что многие распределенные приложения делятся на три уровня: уровень пользовательского интерфейса, уровень обработки и уровень данных. В этой связи возникает ряд возможностей для физического распределения клиент-серверного приложения на нескольких машинах.

Самый простой способ заключается в использовании клиентского компьютера с программами, реализующими уровень (часть)

пользовательского интерфейса, и серверного компьютера с остальными программами, реализующими уровень обработки и данных.

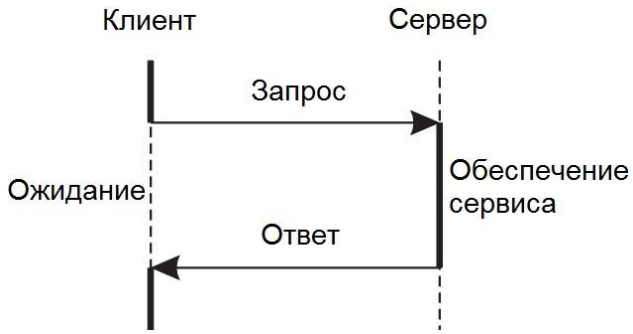


Рис. 2

В другом подходе к организации клиентов и серверов предложено распределение этих уровней по разным машинам (Рис. 3).

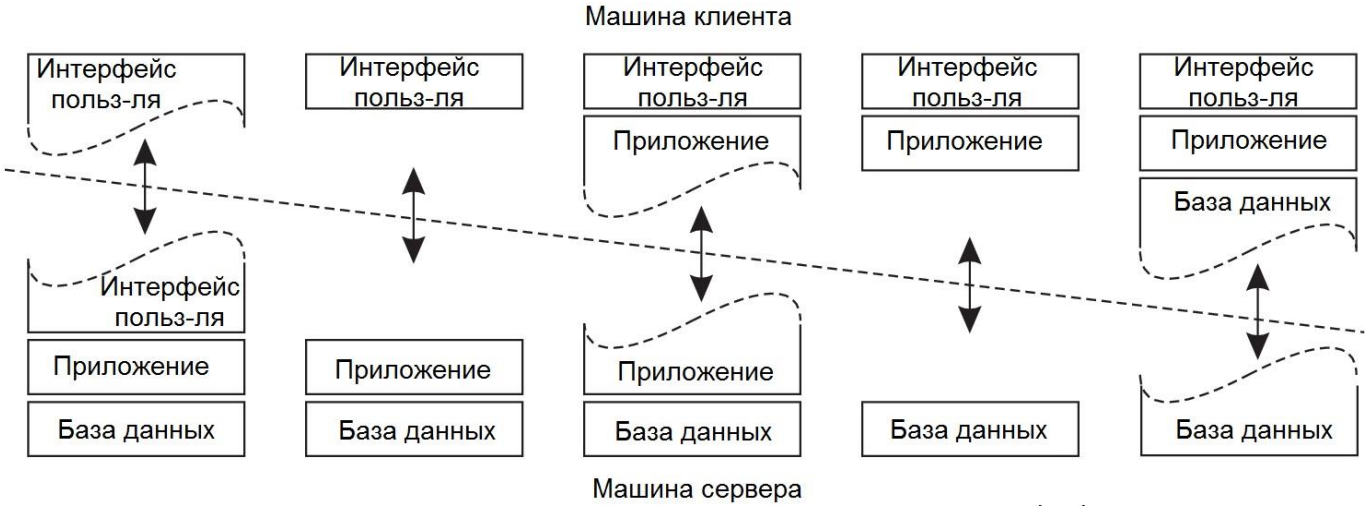


Рис. 3

Существование различия только между двумя типами машин (клиентскими и серверными), приводит к тому, что называется (физическая) двухуровневая архитектура.

В некоторых случаях серверу может потребоваться действовать как клиент, что говорит о применении (физической) трехуровневой архитектуры (Рис.4). В ней программы, которые являются частью уровня обработки, выполняются на отдельном сервере, но могут частично распределяться по клиентским и серверным машинам.

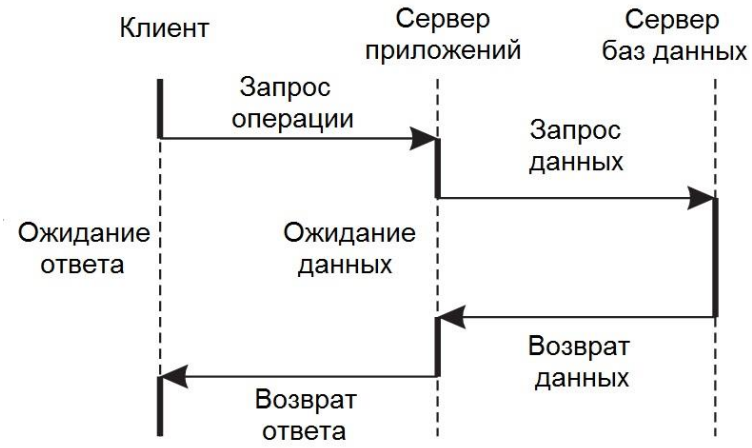


Рис. 4

2.3 Модели взаимодействия в распределенных системах

2.3.1 Служба и протокол

Служба (или сервис) – это набор примитивов (операций), которые более низкий уровень предоставляет более высокому. Служба определяет, какие именно операции уровень будет выполнять от лица своих пользователей, но никак не оговаривает, как должны реализовываться эти операции. Служба описывает интерфейс между двумя уровнями, в котором нижний уровень является поставщиком сервиса, а верхний — его потребителем.

Протокол – это набор правил, описывающих формат и назначение кадров, пакетов или сообщений, которыми обмениваются объекты одного ранга внутри уровня. Объекты используют протокол для реализации определений своих служб. Они могут менять протокол по желанию, при условии, что при этом остаются неизменными службы, предоставляемые ими своим пользователям. Таким образом, служба и протокол оказываются практически независимыми.

2.3.2 Эталонная модель OSI

Всеобщий переход на общий для всех систем стек протоколов начался с разработки стандартной модели взаимодействия открытых систем (OSI). Данная модель появилась вследствие объединения результатов работы двух независимых проектов, осуществляемых Международной организацией по стандартизации (International Organization for Standardization, ISO) и

Международным союзом телекоммуникаций (International Telecommunications Union, ITU), и была опубликована в 1983 году.

В модель OSI закладывалась идея построения общей модели расположенных на разных уровнях протоколов, которые соответствуют основным процессам в компьютерных сетях и определяют взаимодействие между этими уровнями в различных системах.

Открытой может быть названа любая система (компьютер, вычислительная сеть, ОС, программный пакет, другие аппаратные и программные продукты), которая построена в соответствии с открытыми спецификациями. Под открытыми спецификациями понимаются опубликованные, общедоступные спецификации, соответствующие стандартам и принятые в результате достижения согласия после всестороннего обсуждения всеми заинтересованными сторонами.

Назначение модели OSI состоит в обобщенном представлении средств сетевого взаимодействия.

Модель OSI определяет:

- ✓ уровни взаимодействия систем в сетях с коммутацией пакетов;
- ✓ стандартные названия уровней;
- ✓ функции, которые должен выполнять каждый уровень.

Таким образом, модель OSI считается теоретической моделью и только дает представление о том, что должен делать каждый уровень. Она не является сетевой архитектурой, поскольку не содержит описания служб и протоколов, используемых на каждом уровне. На основании нормативов, приведенных в данной модели, разрабатываются протоколы, сетевые устройства и программы. Их часто привязывают к определенным уровням модели OSI, предполагая выполнение ими функций данных уровней.

Модель OSI разделяет взаимодействие в сети на семь отдельных уровней (рис. 5).

Функции уровней модели OSI представлены ниже:

- ✓ прикладной уровень – обеспечение связи между приложениями;
- ✓ уровень представления – форматирование, шифрование и дешифрование данных;
- ✓ сеансовый уровень – создание, поддержка и управление сеансами;

- ✓ транспортный уровень – создание, поддержка и управление сквозным соединением;
- ✓ сетевой уровень – адресация и маршрутизация;
- ✓ канальный уровень – предоставление доступа к среде и управление каналами;
- ✓ физический уровень – передача битового потока.

Уровни формируют модель, в соответствии с которой разные сетевые функции могут быть реализованы в иерархической форме и в нужной последовательности. В совокупности набор уровней называется стеком протоколов или сетевым стеком (похоже на башню из кирпичей).

Каждый уровень выполняет определенные коммуникационные задачи и с помощью соответствующих протоколов взаимодействует с соседними уровнями иерархии. Передача информации между двумя сетевыми устройствами осуществляется с использованием данной иерархии уровней, т.е. стека в каждом из устройств.

Вертикальные связи модели OSI описывают услуги, которые оказывают соседние уровни на одном хосте. Горизонтальные связи определяют протоколы взаимодействия – правила и процедуры, регулирующие порядок взаимодействия хостов в сети.

Уровень не содержит определения отдельного протокола; определяемая уровнем функциональность может быть реализована любым числом протоколов. Таким образом, каждый уровень способен вмещать произвольное число протоколов, каждый из которых реализует службу, соответствующую функциональности этого уровня.

При создании модели OSI предполагалось, что протоколы, разрабатываемые в ее рамках, будут строго соответствовать семи уровням и доминировать в средствах компьютерной связи, что приведет к вытеснению существующих на тот момент фирменных протоколов. Однако этого не произошло. Сейчас протоколы, связанные с моделью OSI, не применяются, и ни один из используемых в настоящее время стеков сетевых протоколов не имеет точного соответствия уровням модели OSI. Во многих случаях протоколы обладают функциональностью, которая перекрывает два или более уровней. Данный факт объясняется тем, что, во-первых, многие протоколы являлись работоспособными и отлаженными еще до публикации

документов, описывающих модель OSI, во-вторых, разработчики отдавали предпочтение практической функциональности, а не полному соответствию уровням модели.

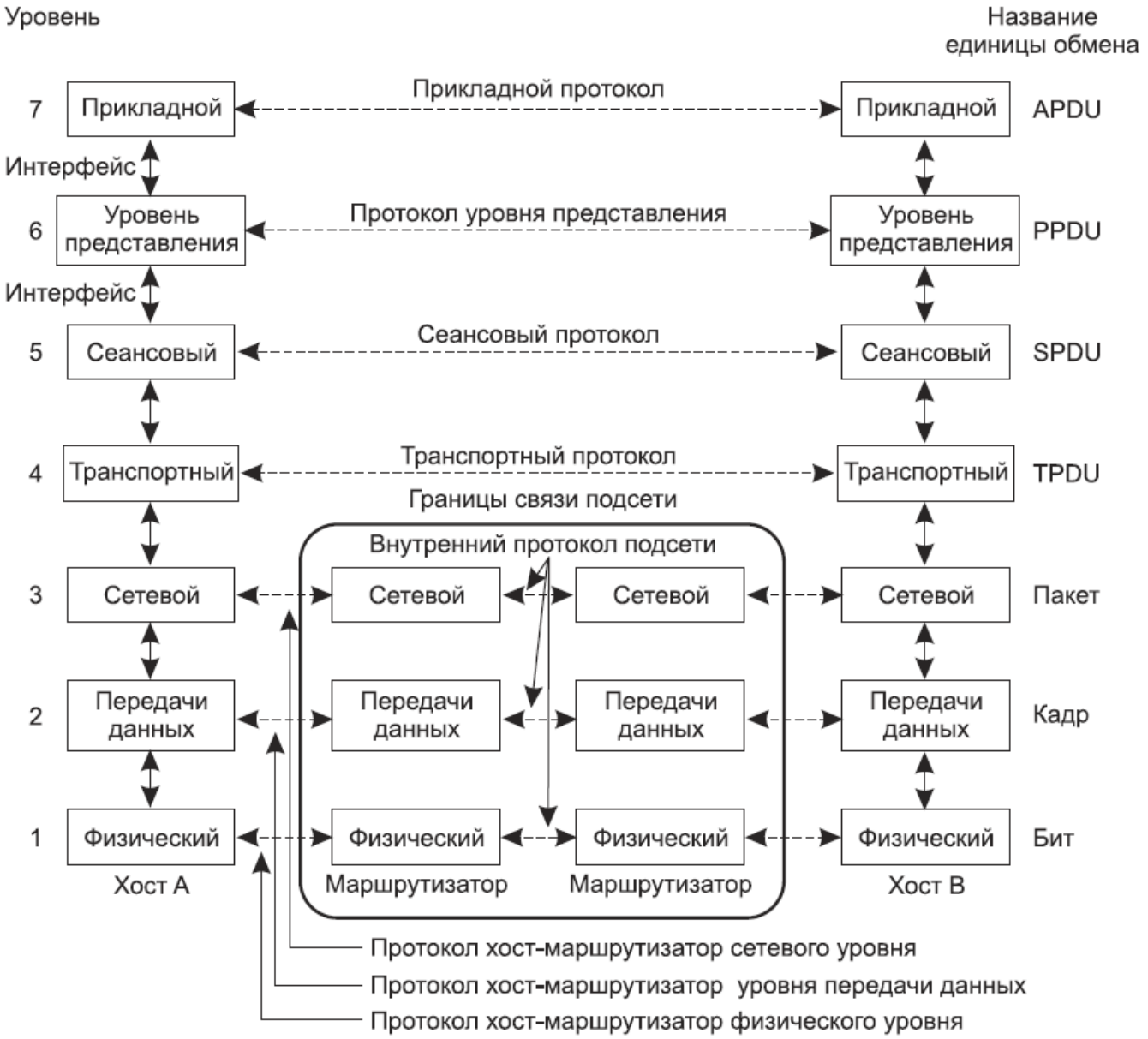


Рис. 5

Тем не менее, сама модель OSI до сих пор весьма актуальна, а свойства ее уровней очень важны. Вследствие ее разработки стало возможным разделить функции стека протоколов так, чтобы независимые группы разработчиков могли работать над различными уровнями. Т.е. общий процесс разработки стал более рациональным.

2.3.3 Эталонная модель TCP/IP

С появлением спутниковых сетей и радиосетей возникла проблема при объединении с ними других сетей с помощью имеющихся протоколов. Решением данной проблемы стало создание эталонной модели TCP/IP (Рис. 6).

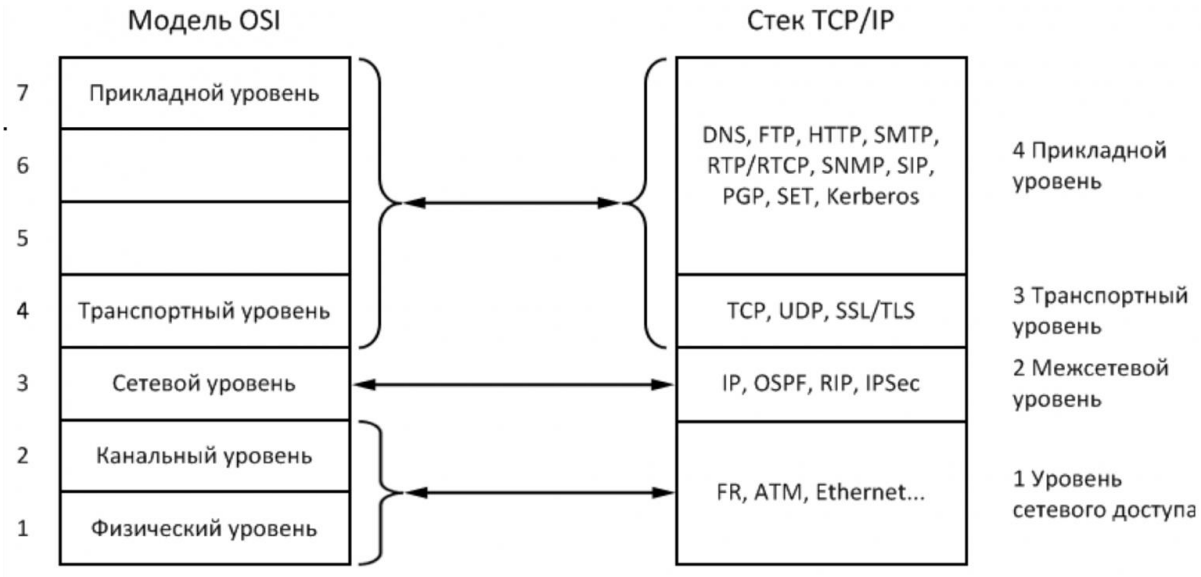


Рис. 6

Она включает в себя четыре уровня:

- ✓ прикладной уровень
- ✓ транспортный уровень
- ✓ межсетевой уровень
- ✓ канальный уровень

Таким образом она описывает базовые принципы работы набора протоколов – логическую передачу и доставку трафика между конечными станциями.

Прикладной уровень содержит все протоколы высокого уровня, например протокол передачи файлов (File Transfer Protocol, FTP), протокол эмуляции терминала telnet, простой протокол передачи почты (Simple Mail Transfer Protocol, SMTP), протокол передачи гипертекста (Hypertext Transfer Protocol, HTTP) и многие другие. Протоколы прикладного уровня развертываются на хостах. На транспортном уровне реализуются протоколы TCP (Transmission Control Protocol – протокол управления передачей) и UDP (User Dataram Protocol – протокол пользовательских дейтограмм).

Межсетевой уровень определяет официальный формат пакета и межсетевой протокол (Internet Protocol, IP). Уровень сетевых интерфейсов отвечает только за организацию взаимодействия с подсетями разных технологий, входящими в составную сеть.

Модель TCP/IP ориентирована на Интернет и является основой для разработки широкого диапазона стандартов в области связи между узлами сети. Часто данную модель называют просто стеком протоколов TCP/IP.

2.3.4 Организация взаимодействия в распределенных системах

Взаимодействие между узлами в распределенной системе допустимо представить в виде абстракции (Рис. 7).

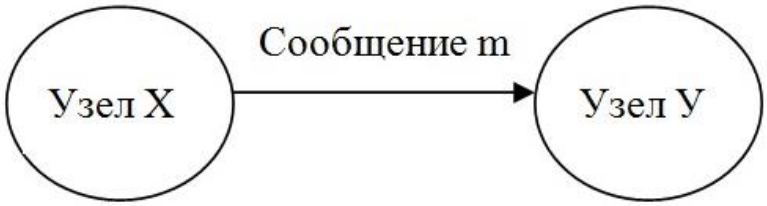


Рис. 7

Такое представление указывает на то, что у одного узла есть способ отправить сообщение другому узлу. В данном случае неважно, как данное сообщение физически представлено или закодировано, то есть какие сетевые протоколы используются. Основной принцип его отправки и получения остается одинаковым, несмотря на изменение в сетевых технологиях.

Предположим, что существует необходимость переслать большой объем данных (десятки терабайт). В этом случае отправка через Интернет будет достаточно медленной. Вариант записи данных на диски и их доставка в место назначения является более рациональным.

Но с точки зрения распределенных систем способ доставки сообщения не важен. Передача происходит по абстрактному каналу с определенной задержкой (задержка от момента отправки сообщения до его получения) и пропускной способностью (объем данных, которые передаются в единицу времени). Целью распределенной системы в данном случае является обеспечение координирования узлов для выполнения некоторой общей

задачи. Алгоритмы, реализованные в распределенной системе, решают, какие сообщения следует отправить и как их обработать при получении.

3 СПОСОБЫ ИСПОЛЬЗОВАНИЯ ИМЕН В РАСПРЕДЕЛЕННЫХ СИСТЕМАХ

Имена применяются для совместного использования ресурсов, идентификации объектов, ссылки на местоположения и т.д. Имя является уникальным, тем самым позволяя процессу получить доступ к названной сущности. Для оперирования именами используется система имен, реализация которой распределяется по нескольким машинам.

3.1 Имя и сущность. Организация доступа к сущности

Имя – это строка битов или символов, которая используется для ссылки на сущность. Сущностью являются конечные устройства, файлы, процессы, пользователи, веб-страницы и т.д.

Типы имен:

- адрес;
- идентификатор;
- символьное имя.

Для получения доступа к сущности и последующей работы с ней необходима точка доступа. Адрес точки доступа сущности называется адресом данной сущности. Например, хост, на котором запущен HTTP-сервер, является точкой доступа. Его адрес – это совокупность IP-адреса и номера порта. Адрес точки доступа может использоваться в качестве имени для конкретной сущности. Однако такой вариант является нерациональным, поскольку сущность может изменить точку доступа, или точка доступа может быть переназначена другой сущности. Гораздо эффективнее использовать имена, которые не зависят от адресов сущностей.

Помимо адреса в качестве имени сущности может использоваться идентификатор. Идентификатор относится только к одной сущности и не используется повторно. Кроме того на каждую сущность ссылается не более одного идентификатора. Использование идентификаторов упрощает процесс ссылки на сущности.

Использование адресов и идентификаторов в двоичной форме часто является неудобным для пользователя. Символьные имена позволяют

решить данную проблему, поскольку представлены в виде осмысленной комбинации букв.

3.2 Пространство имен и реализация службы наименования

Имена организовываются в пространстве, которое называется пространством имен. Оно формирует основу службы имен, которая позволяет пользователям и процессам добавлять, удалять и искать имена. Служба именования реализуется на нескольких серверах имен, количество которых зависит от размера распределенной системы. Она имеет иерархическую древовидную структуру. Разделение имени на части позволяет разделить административную ответственность за назначение уникальных имен между различными людьми или организациями в пределах своего уровня иерархии. В службе именования реализована возможность поиска любой информации, хранящейся в конечном узле, на который ссылается конкретное имя. Процесс поиска имени называется разрешением имени.

3.3 Система доменных имен DNS

Одной из крупнейших служб именования является система доменных имен (DNS), которая отображает символьные имена узлов сети на их IP-адреса (как IPv4, так и IPv6). Причиной возникновения данной системы стал рост масштабов сети Internet и, как следствие, невозможность оперативного обновления таблицы соответствия между символьными именами узлов и их адресами в ручном режиме.

Система доменных имен, как и любая другая служба именования, имеет иерархическую древовидную структуру. Дерево начинается с корня, который обозначается точкой. Затем следует старшая символьная часть имени, вторая по старшинству символьная часть имени и т. д. Младшая часть имени соответствует конечному узлу. Конечный узел обычно хранит информацию о сущности, которую он представляет, например о её адресе.

Совокупность имен, у которых несколько старших составных частей совпадают, образуют домен имен. Например, имена etu.ru и tass.ru входят в домен ru, так как все они имеют одну общую старшую часть – имя ru. Другим примером является домен etu.ru, в который входят имена mail.etu.ru и digital.etu.ru. Этот домен образуют имена, у которых две старшие части

равны etu.ru. Администратор домена etu.ru несет ответственность за уникальность имен следующего уровня, входящих в домен, то есть имен mail и digital. Образованные домены mail.etu.ru и digital.etu.ru являются поддоменами домена etu.ru и, так как имеют общую старшую часть имени.

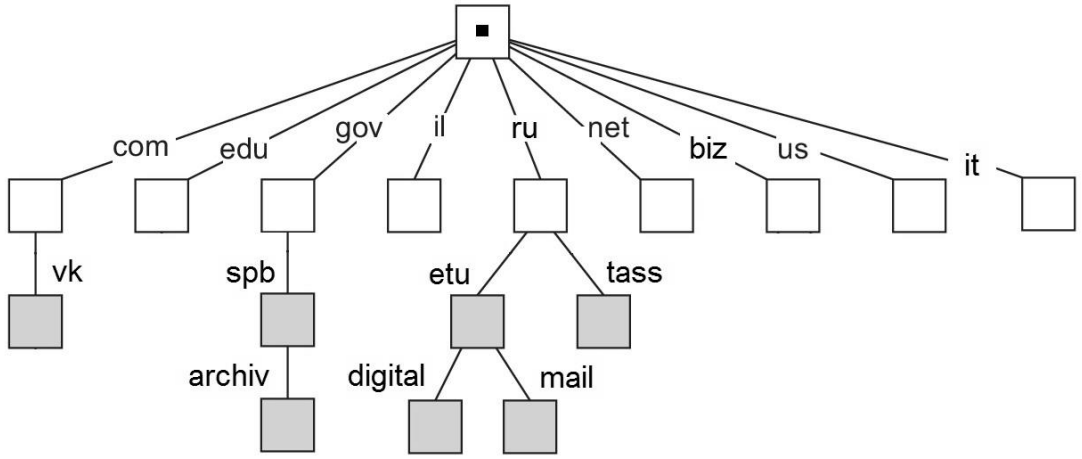


Рис. 8

Корневой домен имен управляется центральными органами Интернета, в том числе ICANN. Домены верхнего уровня назначаются для каждой страны и для различных типов организаций. Например, ru (Россия), il (Израиль), com (коммерческие организации), gov (правительственные организации и т.д. Каждый домен администрирует отдельная организация, которая разделяет свой домен на поддомены и передает функции администрирования этих поддоменов другим организациям.

Система доменных имен включает серверы и клиенты. DNS-серверы поддерживают распределенную базу отображений имен, а DNS-клиенты обращаются к серверам с запросами на получение IP-адреса по известному доменному имени. Описанная процедура называется разрешением доменного имени и реализуется двумя способами. Первый способ является итеративным – DNS-клиент управляет работой по поиску IP-адреса. Он итеративно выполняет последовательность запросов к разным серверам имен. Второй способ является рекурсивным - DNS-клиент перепоручает всю работу по разрешению имени цепочке DNS-серверов.

Пример разрешения доменного имени согласно рекурсивной процедуре приведен далее. Клиент хочет получить доступ к своей корпоративной почте и вводит в строке браузера mail.etu.ru. Сначала производится поиск сервера NS(ru), отвечающий за имена, оканчивающиеся на ru, после чего остальная

часть имени передается на сервер NS(ru). Данный сервер преобразовывает имя etu в сервер NS(etu.ru), отвечающий за имена, оканчивающиеся на etu.ru, который может дополнительно обработать оставшееся имя mail.

NS(.) → NS(ru) → NS(etu.ru) → address of mail.etu.ru,

где NS(.) обозначает сервер, который может вернуть адрес NS(ru), NS(etu.ru) вернет фактический адрес сервера.

3.4 Размещение мобильных сущностей

Традиционные службы именования поддерживают отображение символьных имен в адреса сущностей. Однако, если имя или адрес изменяются, то коррекции должно быть подвергнуто и отображение. Таким образом традиционные службы имен, например DNS, не справляются с мобильными сущностями.

Решение данной проблемы состоит в отделении именования сущностей от их размещения путем ввода идентификатора. При поиске сущности средствами службы именования она возвращает идентификатор. Когда этот идентификатор потребуется снова, его можно будет просто взять с локальной машины, не обращаясь к поиску средствами службы именования.

Размещение сущности определяется посредством отдельной службы локализации. Служба локализации использует в качестве исходных данных идентификатор и возвращает текущий адрес соответствующей ему сущности.

Подходы к локализации мобильных сущностей:

- широковещательная рассылка;
- групповая рассылка;
- передача указателей;
- использование базовой точки;
- построение иерархического дерева поиска.

3.5 Удаление сущностей

В системах именования сущности, на которых нет ссылок, должны удаляться автоматически. Поэтому особое внимание необходимо уделять процессу создания ссылок на сущности.

В распределенных системах существуют механизмы для автоматического удаления сущностей, которые называются

распределенными сборщиками мусора. При сборке мусора выполняется либо подсчет ссылок, либо трассировка.

В случае подсчета ссылок сущность считает количество созданных на нее ссылок. Когда счетчик достигает нуля, сущность удаляется. Вместо подсчета ссылок допустимо создать список ссылок на процессы, ссылающиеся на сущность. Он более стабилен в работе, но имеет проблемы с масштабированием.

В методах трассировки все сущности прямо или косвенно ссылаются на заданный набор корневых сущностей, которые помечаются как доступные. Недоступные сущности удаляются. Распределенная трассировка является трудоемким процессом, поскольку все сущности в системе следует проверить.

4 СИНХРОНИЗАЦИЯ ПРОЦЕССОВ В РАСПРЕДЕЛННЫХ СИСТЕМАХ

Процессы в распределенных системах процессы взаимодействуют, синхронизируя свои действия друг с другом.

В централизованной системе время является однозначным. Процесс А, которому требуется узнать время, обращается к операционной системе. Процесс В, который запрашивает время позже процесса А, никогда не получит время ниже, чем у процесса А. В распределенной системе процесс согласования времени не столь прост.

4.1 Физические часы

В ряде систем (например, в системах реального времени) актуально действительное время часов. В этих условиях необходимы внешние физические часы. Основой для хранения глобального времени выступает универсальное координированное время (Universal Coordinated Time, UTC). UTC является мировым стандартом. Услуги UTC предоставляют 40 коротковолновых радиостанций по всему миру, которые передают короткие импульсы в начале каждой секунды UTC, и несколько спутников Земли. На основе информации от нескольких спутников реализуются наземные серверы времени. Многие компьютеры оснащены приемниками UTC.

4.2 Логические часы

В распределенных системах нет необходимости вести точный учет реального времени. Согласование какого-либо текущего времени каждым из узлов будет являться достаточным.

Американский ученый Лампорт по результатам проведенных исследований определил, что синхронизация часов не обязательно должна быть абсолютной. Если два процесса не взаимодействуют, то синхронизировать их часы нет необходимости. Кроме того, он подчеркнул, что процессам важно согласовать порядок, в котором происходят события, а не точное время.

4.3 Отметки времени Лампорта

Лампорт определил отношение под названием «происходит перед» для синхронизации логических часов. Выражение a->b читается как «событие a происходит до события b». Оно означает, что все процессы согласны с тем, что первым происходит событие a, а затем – событие b. Отношение «происходит перед» используется в двух случаях.

- Если a и b являются событиями, происходящими в одном и том же процессе, и a происходит перед b, то отношение a->b истинно.
- Если a – это событие отправки сообщения одним процессом, а b – событие получения того же сообщения другим процессом, то отношение a->b также является истинным. Сообщение не может быть получено до отправки или в то же время, когда оно было отправлено, поскольку для его получения необходимо ненулевое количество времени.

Отношение «происходит перед» – это транзитивное отношение, то есть в случае, если a->b и b->c, выполняется условие a->c. Если два события, x и y, происходят в разных процессах, которые не обмениваются сообщениями, то отношение x->y и y->x не истинно. Такие события называются параллельными. Это означает, что никто не может знать о том, где и какое из этих событий произошло.

Далее необходимо найти способ измерения времени каждого события, который позволяет поставить в соответствие каждому событию a отметку времени C(a), которая подошла бы всем процессам. Данные отметки времени должны быть такими, чтобы при a->b соблюдалось соотношение

C(a)<C(b). Т.е., если a и b — два события одного процесса и a происходит перед b, то C(a)<C(b). Время по часам, C, всегда идет увеличивается и никогда не уменьшается. Коррекция времени производится только путем добавления к нему положительного значения, а не его вычитания.

На рис. 9 представлен алгоритм Лампорта, применяемый для присвоения времен событиям. Три процесса запущены на различных машинах, каждая из которых имеет собственные часы и скорость работы. Когда часы процесса 0 поставят отметку времени 6, в процессе 1 они покажут 8, а в процессе 2 — 10. Каждые часы идут с постоянной скоростью, но эти скорости различны.

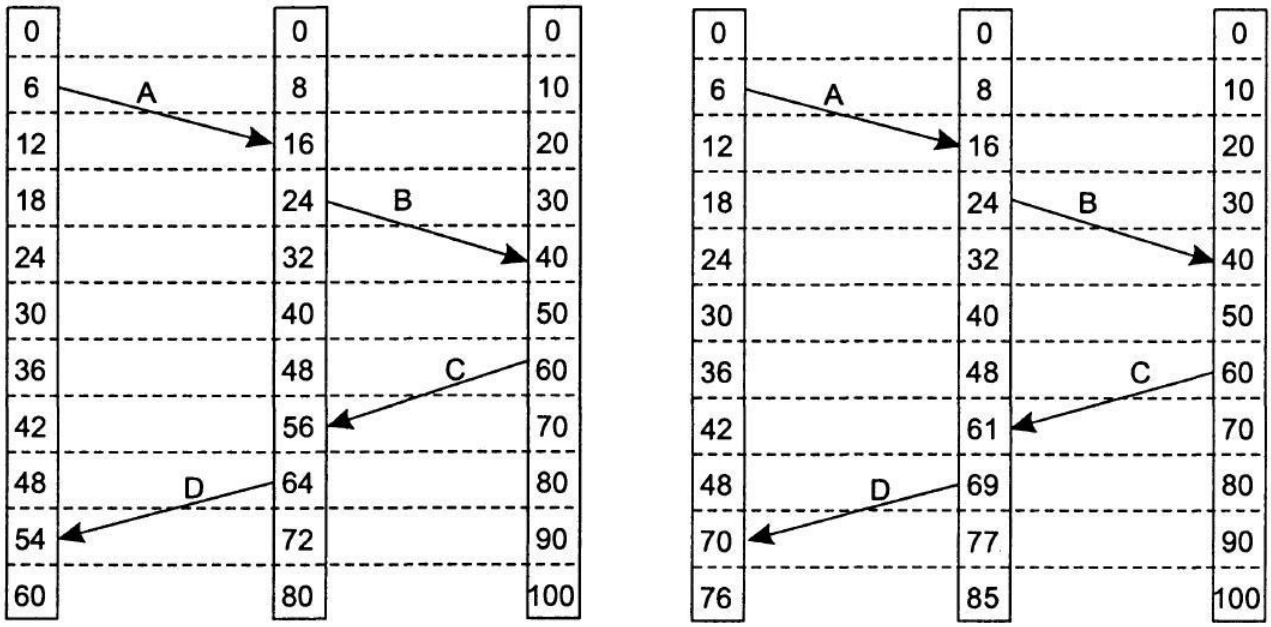


Рис. 9

В момент 6 процесс 0 посылает сообщение A процессу 1. Когда оно придет в процесс 1, его часы будут показывать 16. Если сообщение будет содержать время отправки, 6, процесс 1 сочтет, что на пересылку ушло 10 тиков. Сообщение B от процесса 1 процессу 2 будет доставлено за 16 тиков. Но сообщение C от процесса 2 процессу 1 будет послано на отметке 60 и достигнет цели на отметке 56. Сообщение D от процесса 1 процессу 0 будет послано на отметке 64 и достигнет цели на отметке 54. Данную ситуацию необходимо изменить.

Решение, найденное Лампортом, вытекает из отношения «происходит перед». Поскольку C посылается на отметке 60, оно должно достичь цели на отметке 61 или позднее. Поэтому каждое сообщение содержит время

отправки по часам отправителя. Когда сообщение достигает цели, но часы получателя показывают время, более раннее чем время отправки, получатель переводит свои часы так, чтобы они показывали время на единицу большее времени отправки. На рис. 8 справа сообщение C приходит на отметке 61. Таким же образом сообщение D приходит на отметке 70.

Данный алгоритм удовлетворяет требованиям по глобальному времени с одним дополнением. Между любыми двумя событиями часы должны «протикать» как минимум один раз. Если процесс посылает или принимает в сеансе два сообщения, он должен сделать так, чтобы его часы успели протикать один раз в промежутке между сообщениями.

В некоторых случаях применяется дополнительное требование: никакие два сообщения не должны происходить одновременно. Чтобы выполнить это требование, можно добавить номер процесса, в котором происходят события, справа от метки времени и отделить его от целой части десятичной точкой. Таким образом, если в процессах 1 и 2 в момент времени 40 происходят события, то первый из них будет происходить в 40.1, а второй в 40.2.

Используя этот способ, можно указать время для всех событий в распределенной системе, подпадающих под перечисленные ниже условия, и упорядочить их.

- Если a происходит раньше b в одном и том же процессе, то $C(a)<C(b)$,
- Если a и b представляют собой отправку и получение сообщения, соответственно $C(a)<C(b)$,
- Для всех различимых событий a и b выполняется соотношение $C(a)\neq C(b)$.

4.4 Взаимоисключающий доступ для процессов

В распределенной системе нескольким процессам может понадобиться доступ к одному и тому же ресурсу одновременно. Это может привести к искажению его содержимого. Специальные распределенные алгоритмы позволяют предоставить взаимоисключающий доступ для процессов.

Классификация распределенных алгоритмов взаимного исключения

- решения на основе токенов;

- подход на основе разрешений.

В первом случае между процессами для взаимного исключения передается специальное сообщение – токен. Токен представлен в единственном экземпляре. Процесс, который имеет токен, получает доступ к общему ресурсу. После завершения токен передается следующему процессу. Если процесс с токеном не заинтересован в доступе к ресурсу, он передает его дальше.

Описанный подход гарантирует, что каждый процесс получит возможность доступа к ресурсу в процессе своей работы, а тупиковых ситуаций, в результате которых несколько процессов бесконечно ждут продолжения друг друга, не возникнет. Однако, если токен будет утерян (например, из-за сбоя использующего его процесса), процедура создания нового токена будет достаточно трудоемкой.

Во втором случае процесс, который хочет получить доступ к ресурсу, сначала требует разрешения от других процессов различными способами.

4.5 Выбор координатора

Многие распределенные алгоритмы требуют, чтобы процесс выполнял определенную роль, например функции координатора, инициатора т.п. При синхронизации требуется, чтобы один процесс действовал как координатор. В тех случаях, когда координатор не зафиксирован, необходимо, чтобы процессы в распределенных вычислениях принимали решение о том, кто будет координатором. Для данной цели используются алгоритмы выборов. Запущенный алгоритм выбора завершается всеми процессами, согласовывающими, кем должен быть новый координатор.

4.6 Распределенные транзакции

В распределенной системе пользователям предоставляется возможность отправить запросы к нескольким серверам в виде одного большого запроса и выполнить его как распределенную транзакцию. Сутью данной операции является выполнение или всех, или ни одного из запросов.

Для работы с транзакциями используют специальные примитивы, примеры которых представлены ниже.

- BEGIN_TRANSACTION – отметить начало транзакции;

- END_TRANSACTION - прекратить транзакцию и попытаться завершить ее;
- ABORT_TRANSACTION - прервать транзакцию и восстановить прежние значения;
- READ - читать данные из файла, таблицы или другого источника;
- WRITE - записать данные в файл, таблицу или другой приемник.

Транзакции обладают следующими свойствами:

- атомарность - транзакция осуществляется неделимой для внешнего мира;
- согласованность - транзакция не нарушает системные инварианты;
- изолированность - параллельные транзакции не мешают друг другу;
- долговечность - изменения после совершения транзакции являются постоянными.

Транзакции в распределенных системах часто строятся как ряд субтранзакций, которые вместе формируют вложенную транзакцию, изображенную на рис. 10. Транзакция верхнего уровня разделяется на потомки, которые работают параллельно один с другим на разных машинах с целью увеличения производительности. Каждый из этих потомков может также выполнить одну или несколько субтранзакций или продолжить разделение.

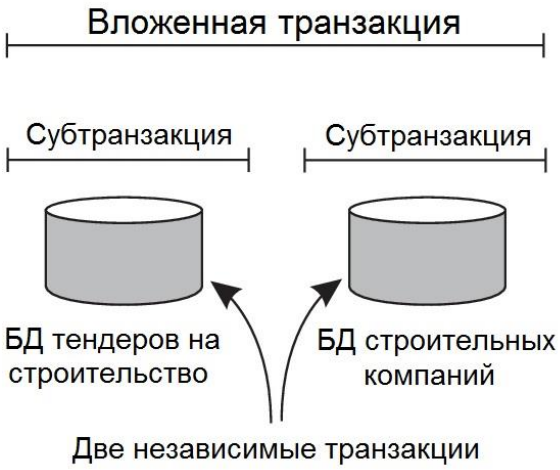


Рис. 10

Если субтранзакция осуществляется, а затем запускается новая субтранзакция, вторая видит результаты, полученные первой. Если вложенная (более высокий уровень) транзакция прерывается, все ее

субтранзакции должны быть также прерваны. Если несколько транзакций запускаются одновременно, результат таков, как будто они запускались последовательно в неустановленном порядке.

Вложенные транзакции предоставляют естественный способ распределения транзакций по нескольким машинам, что является важным в контексте распределенных систем. С их помощью обеспечивается логическое разделение работы исходной транзакции.

5 РЕПЛИЦИРУЕМЫЕ ДАННЫЕ В РАСПРЕДЕЛЕННЫХ СИСТЕМАХ

5.1 Репликация данных и обеспечение их согласованности

Репликация - механизм синхронизации содержимого нескольких копий одних и тех же данных на нескольких узлах, каждый из которых называется репликой. Она является стандартной функцией многих распределенных баз данных, файловых систем и других систем хранения. Репликация выполняется с целью повышения надежности и производительности. Надежность обеспечивается за счет возможности переключения с одной копии данных на другую в случае сбоя, производительность – за счет репликации сервера с последующим распределением рабочей нагрузки между процессами, которые работают с данными, или размещения копии данных в непосредственной близости от процесса, который использует ее. Однако наличие нескольких копий может привести к проблемам согласованности. Чтобы ее избежать, необходимо, чтобы внесенные изменения отражались во всех копиях. Но в крупных распределенных системах использование такого подхода может привести к обратному эффекту - резкому снижению производительности.

Для достижения баланса между необходимостью согласованности реплик и увеличению производительности распределенной системы на практике предлагается смягчать требования к согласованности. Степень ослабления сильно зависит от специфики приложений, поэтому общие правила отсутствуют.

На практике выделяют три независимых варианта для определения несогласованности:

- отклонения между репликами в числовых значениях;
- отклонения между репликами в устаревании;

- отклонение относительно порядка операций обновления.

Первый вариант используется приложениями, которые работают с числовыми данными. Например, можно установить абсолютное и относительное числовые отклонения для реплик. Если обновление данных в обоих случаях происходит без нарушения указанных числовых отклонений, реплики считаются взаимно непротиворечивыми.

Второй вариант актуален для приложений, в которых содержимое реплик может содержать не слишком старые данные. Например, прогноз погоды редко меняется в течении нескольких часов. Поэтому сервер, получая обновления, может принять решение распространять их на реплики время от времени.

Третий вариант применяется в приложениях, в которых порядок обновлений в разных репликах может быть различным, если различия остаются ограниченными. Один из способов просмотра таких обновлений состоит в том, что они предварительно применяются к локальной копии, ожидая глобального соглашения со всеми репликами. От ряда обновлений, возможно, придется отказаться и применить их в другом порядке, прежде чем они станут постоянными.

Отдельно следует отметить, что ряд крупномасштабных распределенных и реплицированных баз данных допускают относительно высокую степень несогласованности. Если в течение длительного времени не происходит никаких обновлений, то все реплики постепенно становятся согласованными. Такая форма согласованности называется конечной согласованностью.

5.2 Управление репликами

Перед использованием репликации необходимо разместить реплики в распределенной системе.

Данная задача, как правило, разбивается две подзадачи, а именно: задачу размещения серверов реплики и задачу размещения контента. Размещение сервера-реплики подразумевает поиск лучших мест для размещения сервера, на котором можно развернуть хранилище данных. Размещение контента связано с поиском лучших серверов для размещения контента.

С целью решения первой подзадачи производится анализ свойств клиента и сети. Например, выполняется измерение расстояний между клиентами и возможными местоположениями сервера с точки зрения задержки или пропускной способности. В итоге, выбирается такое местоположение сервера, для которого среднее расстояние между ним и его клиентами являлось минимальным.

При решении второй подзадачи вводят понятия трех типов реплик, согласно рис.11, с различной логической организацией.

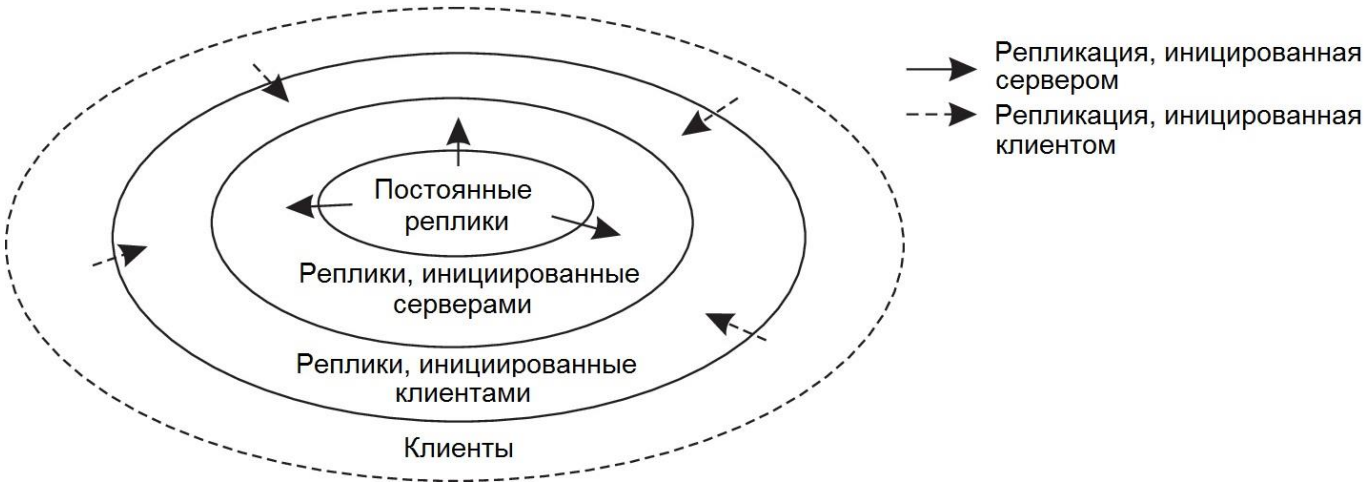


Рис. 11

Постоянные реплики – это начальный набор реплик, составляющих распределенное хранилище данных. Как правило, их количество невелико. Примерами являются зеркала сайта, которые географически распределены по интернету, реплики базы данных, которые расположены на нескольких серверах, образующих кластер.

Реплики, инициированные сервером - это набор реплик, которые являются копиями хранилища данных, которые существуют для повышения производительности и создаются по инициативе владельца хранилища данных. Примерам служит реплики веб-сервера, временно установленные в регионах, из которых наблюдается повышенное число запросов в конкретные месяцы.

Реплики, инициированные клиентом - это набор реплик, представляющих собой клиентские кеши. Кеш – это локальное хранилище, которое используется клиентом для временного хранения копии только что запрошенных данных. Данные хранятся в кеше в течение ограниченного периода времени с целью недопущения использования крайне устаревших

данных. Кеш обычно располагается на клиентской машине и управляется соответствующим клиентом. Разделение кеша между несколькими клиентами актуально в случае, когда запрос данных от клиента C1 также может быть полезен для запроса от другого соседнего клиента C2. Однако, на практике такой метод используется редко.

Управление репликами связано с вопросом распространения обновленного контента на соответствующие серверы реплик.

Характер распространения представлен в следующих вариантах:

1. распространять только уведомление об обновлении;
2. переносить данные из одной копии в другую;
3. распространять операцию обновления на другие копии.

В первом случае другие копии информируются о том, что произошло обновление и что содержащиеся в них данные больше недействительны. Если в дальнейшем запрашивается операция с недействительной копией, то последняя должна обновляться в первую очередь, в зависимости от конкретной модели согласованности.

Во втором случае перенос эффективен, когда отношение чтение-запись относительно велико. Тогда измененные данные с большой долей вероятности будут считаны до следующего обновления.

В третьем случае, который также называется активной репликацией, каждой реплике сообщается, какую операцию обновления ей следует выполнить.

В распределенных системах существуют два подхода, которые касаются механизма распространения обновлений.

- Подход на основе проталкивания или серверные протоколы. Обновления распространяются на другие реплики, даже если эти реплики не запрашивают обновления. Данный подход обеспечивает строгую согласованность.
- Подход на основе извлечения или клиентские протоколы. Обновления, которые сервер имеет на текущий момент, отправляются по запросу от другого сервера или клиента. Такой подход часто востребован кешами клиентов.

В зависимости от того, какой из двух механизмов выбран, используется либо одноадресная, либо групповая рассылка. Клиентские протоколы используют одноадресную рассылку, поскольку при таком подходе только один клиент или сервер запрашивает обновление своей копии. Серверные протоколы используют групповую рассылку с целью отправки обновлений сразу нескольким серверам.

5.3 Согласованность протоколов

Протокол согласованности - это протокол, который описывает реализацию конкретной модели согласованности.

5.3.1 Первичные протоколы

Первичные протоколы - это протоколы, в которых каждый элемент данных x в хранилище данных имеет связанный с ним первичный элемент, который отвечает за координацию операций записи в x.

Типы первичных протоколов:

- протоколы первичного резервного копирования;
- протоколы локальной записи.

Протокол первичного резервного копирования – это первичный протокол, в котором все операции записи необходимо перенаправлять на фиксированный одиночный сервер, и операции чтения могут выполняться локально. Механизм его работы представлен на рис.12. На нем приняты следующие обозначения:

- W1 – запрос на запись;
- W2 – пересылка запроса на первичный сервер;
- W3 – сигнал на обновление резервных копий;
- W4 – подтверждение обновления;
- W5 – подтверждение выполнения записи;
- R1 – запрос на чтение;
- R2 – подтверждение выполнения чтения.

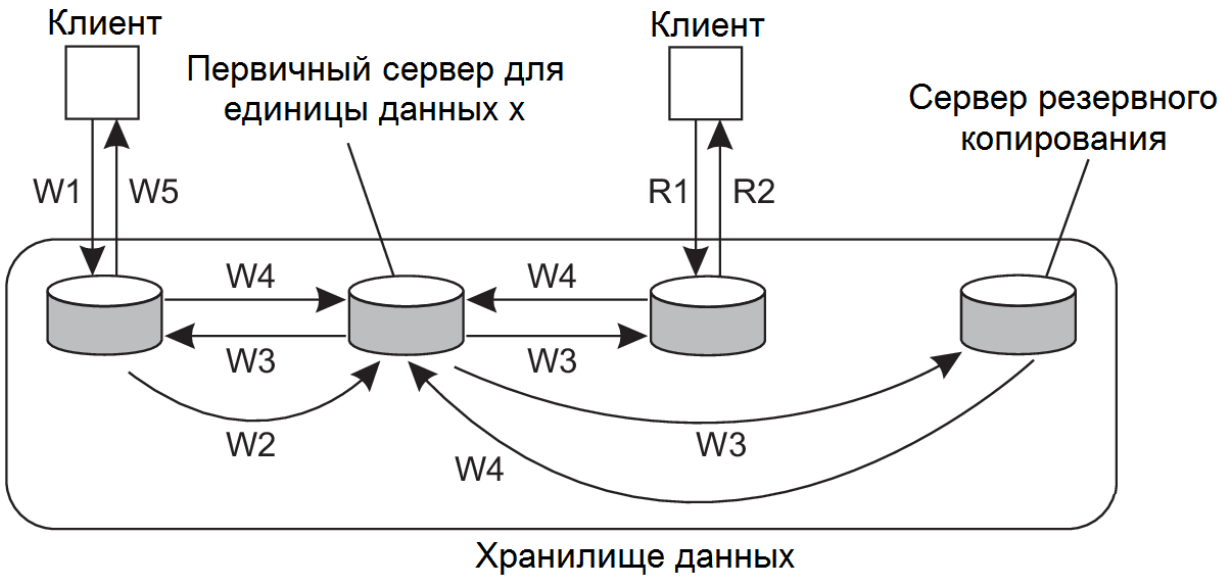


Рис. 12

Процесс, желающий выполнить операцию записи для элемента данных x, перенаправляет эту операцию на основной (первичный) сервер для x. Основной сервер выполняет обновление своей локальной копии x, а затем передает обновление на серверы резервного копирования. Каждый сервер резервного копирования также выполняет обновление и отправляет подтверждение первичному серверу. Когда все резервные копии обновили свою локальную копию, основной сервер отправляет подтверждение начальному процессу, который, в свою очередь, информирует клиента.

Протокол локальной записи - это первичный протокол, в котором первичная копия мигрирует между процессами, которые хотят выполнить операцию записи. Механизм его работы представлен на рис.13. На нем приняты следующие обозначения:

- W1 — запрос на запись;
- W2 — перемещение элемента данных x на новый первичный сервер;
- W3 — подтверждение завершения записи;
- W4 — сигнал на обновление резервных копий;
- W5 — подтверждение обновления;
- R1 — запрос на чтение;
- R2 — подтверждение выполнения чтения.

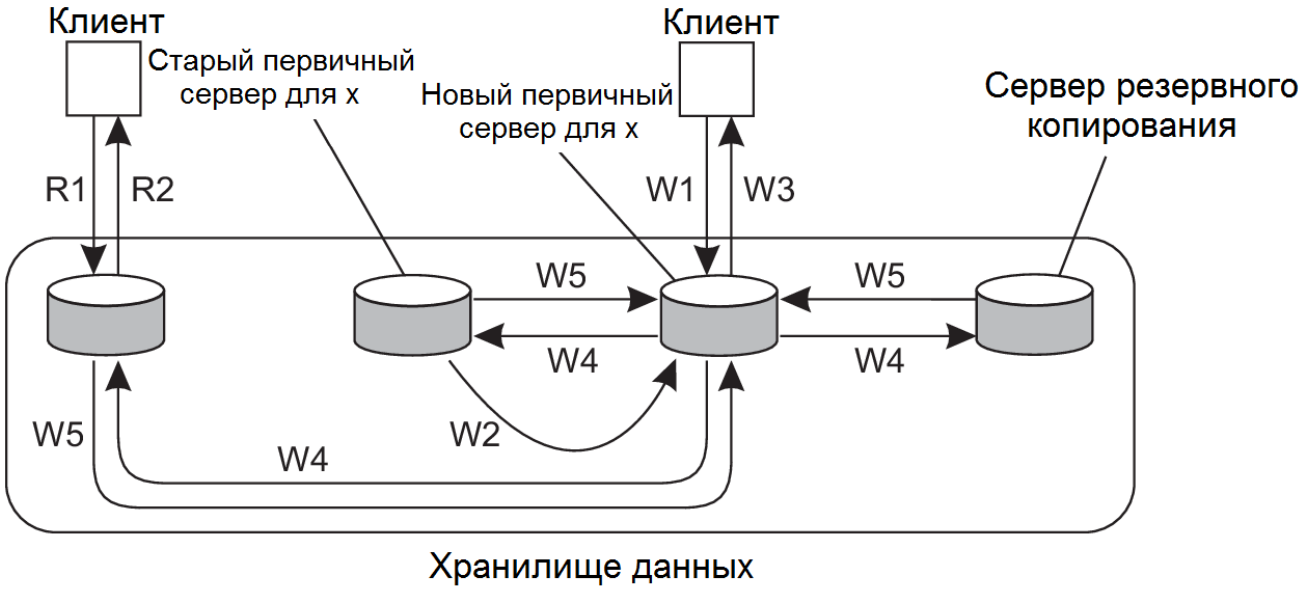


Рис. 13

Процесс, который хочет обновить элемент данных x, находит основную копию x и впоследствии перемещает ее в свое собственное местоположение. Несколько последовательных операций записи могут выполняться локально, тогда как процессы чтения могут по-прежнему получать доступ к своей локальной копии.

5.3.2 Протоколы реплицируемой записи

Протоколы реплицированной записи – это протоколы, в которых операции записи могут выполняться в нескольких репликах, а не только в одной, как в случае первичных реплик.

Типы протоколов реплицируемой записи:

- активная репликация;
- протоколы на основе кворума.

Активная репликация - это репликация, в которой каждая реплика имеет связанный процесс, выполняющий операции обновления. Обновления распространяются посредством операции записи, которая посылается каждой реплике. Поскольку все операции должны выполняться везде в одном и том же порядке, необходимо наличие центрального координатора - секвенсора. Каждая операция сначала пересылается в секвенсор, который присваивает ей уникальный порядковый номер. Далее секвенсор перенаправляет эту операцию всем репликам. Операции выполняются в соответствии с их порядковым номером.

Протоколы на основе кворума - это протоколы реплицируемой записи, которые используют механизм голосования. Перед чтением или записью реплицированного элемента данных от клиентов требуется запрашивать и получать разрешение нескольких серверов.

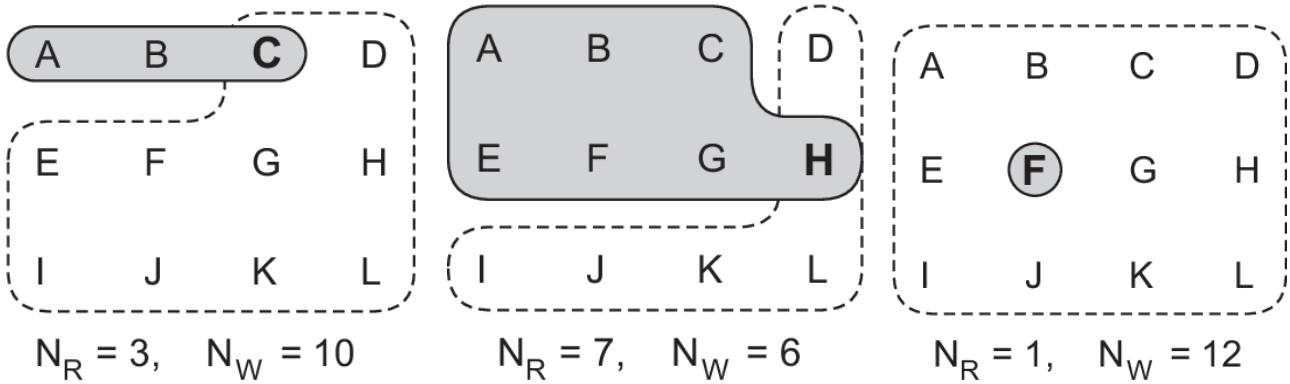
Схема работы протокола на основе кворума описана ниже. Для чтения файла, в котором существует N реплик, клиент должен собрать кворум чтения, в который входят N_R серверов или более. Для редактирования файла необходим кворум записи минимум из N_W серверов. Значения N_R и N_W имеют следующие ограничения:

1) $N_R + N_W > N$;

2) $N_W > N/2$.

Первое ограничение используется для предотвращения конфликтов чтения-записи, второе ограничение - для предотвращения конфликтов записи-записи. После получения согласий необходимого количества серверов файл может быть прочитан или записан.

На рис. 14а проиллюстрирован пример работы протокола на основе кворума, где $N_R = 3$ и $N_W = 10$. Последний кворум записи состоял из 10 серверов от C до L. Они получили новую версию и новое число версии. Любой последующий кворум чтения из трех серверов должен содержать хотя бы одного члена данного набора. Выбор самой новой версии клиентом осуществляется на основании её номера.



$N_R = 3$, $N_W = 10$ (а)

$N_R = 7$, $N_W = 6$ (б)

$N_R = 1$, $N_W = 12$ (в)

Рис. 14

На рис. 14б описана ситуация возможного возникновения конфликта запись-запись, поскольку $N_W < N/2$. Например, если один клиент выберет {A,

B, C, E, F, G} в качестве своего набора записи, а другой клиент выберет {D, H, I, J, K, L}, то оба обновления будут приняты без обнаружения фактического конфликта.

На рис. 14в представлена схема, которая называется ROWA (Read One, Write All; читай один, пиши все). Количество N_R равно единице. Это значит, что реплицированный файл может быть прочитан путем обнаружения и использования любой копии. Однако запись возможна только при наличии всех копий.

СПИСОК ЛИТЕРАТУРЫ

1. Ghosh S. Distributed Systems. An Algorithmic Approach. Second Edition. - USA, Florida, Boca Raton: Taylor & Francis Group, 2015. - 512 p.
2. Kleppmann M. Distributed Systems. - United Kingdom, Cambridge: University of Cambridge, 2021. - 90 p. Course web page: https://www.cst.cam.ac.uk/teaching/2122/ConcDisSys
3. Бабичев, С. Л., Коньков К. А. Распределенные системы: учебное пособие для вузов.— Москва : Издательство Юрайт, 2019. — 507 с.
4. Бёрнс Б. Распределенные системы. Паттерны проектирования. — СПб.: Питер, 2019. — 224 с.
5. Гольдштейн Б.С., Соколов Н.А., Яновский Г.Г. Сети связи: учебник для ВУЗов. СПб.: БХВ- Петербург, 2010. – 400 с.
6. Дэвис К. Шаблоны проектирования для облачной среды / пер. с анг. Д. А. Беликова. – М.: ДМК Пресс, 2020. – 388 с.
7. Закер З. Компьютерные сети. Модернизация и поиск неисправностей. - СПб.: БХВ- Петербург, 2000. – 1008 с.
8. Олифер В., Олифер Н. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 5-е изд. — СПб.: Питер, 2016. — 992 с.
9. Палмер М., Синклер Р.Б. Проектирование и внедрение компьютерных сетей. 2-е изд., перераб. и доп. - СПб.: БХВ-Петербург, 2004. – 752 с.
10. Сергеев А.Н. Основы локальных компьютерных сетей: учебное пособие. — СПб.: Издательство «Лань», 2016. - 184 с.
11. Стин ван М., Таненбаум Э. С. Распределенные системы / пер. с англ. В. А. Яроцкого. – М.: ДМК Пресс, 2021. – 584 с.
12. Столлингс В. Современные компьютерные сети. - СПб.: Питер, 2003. - 783 с.
13. Таненбаум Э., Уэзеролл Д. Компьютерные сети. 5-е изд. — СПб.: Питер, 2012. — 960 с.
14. Хант К. TCP/IP. Сетевое администрирование, 3-е издание. - СПб: Символ-Плюс, 2007. - 816 с.

ОГЛАВЛЕНИЕ