Содержание

Предисловие

Эта книга содержит введение в абстракции распределенного программирования и знакомит с фундаментальными алгоритмами и их реализациями в нескольких распределенных окружениях. Перед читателем будут раскрыты важные проблемы распределенных вычислений и основные алгоритмические приемы их решения. На подробных примерах читатель сможет понять, как с помощью этих приемов конструировать распределенные приложения. Главной темой книги является обеспечение устойчивости к случайным и преднамеренным воздействиям в распределенных системах, которые могут быть обусловлены задержками в сети, ошибками и даже злонамеренными атаками.

Содержание

В современных вычислениях программы нередко объединяют *несколько процессов*. Процесс в данном контексте – это простая абстракция, которая может представлять физический или виртуальный компьютер, процессор в компьютере или конкретный поток выполнения в многозадачной системе. Основная проблема таких распределенных программ состоит в том, чтобы заставить все процессы *вместе работать* над решением *общей* задачи. При этом в каждом процессе все еще приходится решать традиционные алгоритмические проблемы. Распределенные окружения от единичных компьютеров до вычислительных центров и даже глобальных систем, доступных круглосуточно, влекут за собой дополнительные сложности: как добиться надежного решения задач даже в случае аварийного завершения некоторых процессов, потери связи с некоторыми процессами и злонамеренного нападения на некоторые процессы? Распределенные алгоритмы должны быть безотказными, надежными, безопасными и иметь предсказуемое поведение даже при наличии в распределенном окружении отрицательно влияющих факторов.

Если бы не требовалось сотрудничество процессов, распределенные программы просто представляли бы собой множество независимых программ, каждая из которых выполняется в отдельном процессе, и мы были бы лишены преимуществ многозадачности в распределенных системах. Потребность в обеспечении сотрудничества обнажила множество сложных проблем, рассматривающихся в этой книге, которые необходимо было решить, чтобы сделать распределенные вычисления реальностью. Данная книга не просто знакомит читателей с описаниями этих проблем, но также демонстрирует пути их решения в разных контекстах.

Неудивительно, что распределенное программирование можно существенно упростить, если заключить сложности надежного сотрудничества в определенные *абстракции*. Изолируя сложные алгоритмические проблемы, такие абстракции распределенного программирования служат мостом, соединяющим уровни сетевых взаимодействий, которые не дают никаких гарантий надежности, и распределенные прикладные уровни, которые часто требуют высоконадежных примитивов.

Книга знакомит с разными абстракциями распределенного программирования и описывает алгоритмы для их реализации. В некотором смысле мы даем прикладному программисту библиотеку спецификаций абстрактных интерфейсов, а системному программисту – библиотеку алгоритмов для реализации этих спецификаций.

Работая над этой книгой, мы потратили много времени, чтобы сформировать коллекцию упражнений и их решений. Мы настоятельно рекомендуем читателям проработать эти упражнения. Никакие знания и навыки не могут быть приобретены пассивным путем. Это особенно верно в области распределенных вычислений, где человеческий разум часто стремится следовать за привлекательными, но ошибочными представлениями. Книга также включает решения для всех упражнений, чтобы подчеркнуть наше намерение сделать их неотъемлемой частью содержимого книги. Многие упражнения просты и могут решаться коллективно. Другие – сложнее и требуют больше времени. Как правило, такие упражнения решаются индивидуально.

Представление

Эта книга получилась вполне самодостаточной. Это стало возможным, потому что сфера распределенных алгоритмов достигла определенного уровня зрелости, когда отвлекающие детали можно вынести за рамки обсуждения распределенных алгоритмов. К числу таких деталей можно отнести работу сетей связи, разного рода отказы, а также реализации криптографических примитивов; все они подробно рассматриваются в других работах. Безусловно, элементарные знания алгоритмов, логики предикатов, языков программирования, особенностей работы сетей, принципов обеспечения безопасности и операционных систем будут полезны при чтении книги. Но мы считаем, что большинство наших абстракций и алгоритмов будет понятно даже тем, кто обладает минимальными знаниями об этих понятиях.

Книга следует поэтапному подходу и писалась в первую очередь как учебник для высших учебных заведений. Она знакомит читателей с основными элементами распределенных вычислений на наглядных примерах и выстраивает сложные абстракции распределенного программирования постепенно, от простого к сложному. Всякий раз, разрабатывая алгоритм для реализации той или иной абстракции, мы сначала рассматриваем простую модель распределенной системы, а затем адаптируем этот алгоритм для более сложных моделей. Иными словами, мы сначала разрабатываем алгоритм, опираясь на упрощенные предположения об устройстве распределенного окружения, а затем обсуждаем, как избавиться от этих упрощений.

Мы старались сбалансировать простоту и наглядность, с одной стороны, и строгость – с другой. Местами строгости уделяется меньше внимания, и это не всегда делалось специально. Основной упор в этой книге делается на абстракциях и алгоритмах, а не на вычислимости и сложности. На самом деле в этой книге не обсуждаются теоремы. Доказательства приводятся лишь с целью разъяснения алгоритмов: сами по себе они не являются формальными доказательствами.

Организация

Эта книга содержит шесть глав в двух частях. Первая часть строит общую платформу.

В главе 1 мы *обоснуем* необходимость абстракций распределенного программирования, обсудив некоторые приложения, где используются эти абстракции. В этой главе также будут представлены блочная форма записи и псевдокод, используемые для записи алгоритмов в книге.

В главе 2 будут представлены разного рода *допущения и предположения* о распределенной среде. С этой целью мы введем в обиход семейство моделей распределенных систем. В основном модели описывают низкоуровневые абстракции, на которых будут строиться абстракции более высокого уровня. К числу таких абстракций относятся абстракции процесса и связей. Эту главу можно рассматривать как справочник для других глав.

Остальные четыре главы составляют вторую часть книги. Каждая глава посвящена одной проблеме, содержащей широкий класс абстракций, и разным алгоритмам их реализации. Мы будем двигаться от простых абстракций к сложным.

В главе 3 мы введем абстракции обмена данными для распределенного программирования. Они описывают *широковещательную рассылку* сообщений группам процессов и предлагают разные гарантии надежности доставки сообщений процессам. Например, мы обсудим, как гарантировать, что сообщение, доставленное одному процессу, будет доставлено всем другим процессам, даже в случае аварийного завершения процесса-отправителя.

В главе 4 мы обсудим абстракции *разделяемой памяти* как простые формы распределенных хранилищ объектов, доступных для чтения и записи. Это могут быть файлы в распределенной системе хранения или регистры в памяти многопроцессорного компьютера. Мы охватим методы чтения и записи данных клиентами, такие, что значения, хранимые множеством процессов, позднее могут быть получены, даже если некоторые из процессов потерпели аварию, стерли свои значения или записали ошибочные данные.

В главе 5 мы обратимся к абстракциям *консенсуса*, с помощью которых множество процессов может приходить к общему решению, опираясь на исходные данные, изначально предложенные процессами. Они могут приходить к тому же решению, несмотря на отказы в отдельных процессах, которые могут не только завершаться аварийно, но и активно препятствовать принятию общего решения.

В главе 6 мы рассмотрим *варианты консенсуса*, которые могут быть получены расширением или адаптацией абстракции консенсуса в соответствии с потребностями приложений. Сюда входят рассылка в общем порядке (total-order broadcast), надежная рассылка с ограничениями, атомарная (неблокирующая) фиксация, членство в группах и синхронные взаимодействия.

Распределенные алгоритмы, которые мы будем обсуждать, отличаются не только фактическими абстракциями, которые они реализуют, но также набором предположений и допущений о распределенной среде. Начальное множество абстракций, которое алгоритм считает самим собой разумеющимся, мы назвали мо-

делью *распределенной системы*. Существует множество аспектов, оказывающих фундаментальное влияние на проектирование алгоритмов, таких как надежность связей, степень синхронности системы, серьезность отказов и требования к детерминизму решения.

В разных местах в книге одни и те же базовые примитивы распределенного программирования будут реализованы в нескольких моделях распределенных систем. Такой подход имеет перед собой две цели: во-первых, дать представление о конкретной проблеме, свойственной данной модели, и, во-вторых, показать, как выбор модели влияет на реализацию примитива.

Материал всех глав и комплекс упражнений образуют обширное и исчерпывающее введение в тему распределенного программирования. Каждая глава, описывающая отдельный класс абстракций и алгоритмы их реализации в самом простом представлении, то есть в самой простой модели, предусматривающей только аварийные отказы, могла бы с успехом составить основу более короткого и простого курса. Такие курсы могли бы служить хорошим подспорьем для более практических курсов по распределенному программированию.

Изменения во втором издании

Это издание является полностью пересмотренной версией первого издания. Обновлениям подверглась большая часть книги. Но самое большое изменение произошло в области охватываемых тем, число которых увеличилось и дополнилось темой обеспечения *безопасности против злонамеренных действий*. Абстракции и алгоритмы в модели распределенных вычислений, где допускается вероятность злонамеренных действий, известны как Византийская модель сбоев (Byzantine fault-tolerance), или модель произвольных сбоев.

Первое издание книги называлось «Introduction to Reliable Distributed Programming». Добавление одного слова («secure» – «безопасное») в название и увеличение числа соавторов в жизни книги отражает развитие сферы распределенных систем в современном мире. Еще в 2006 году, когда вышло первое издание книги, было очевидно, что большинство действующих распределенных систем постоянно находится под угрозой вторжения и что штатных сотрудников, имеющих доступ к конфиденциальной информации, нельзя исключать из списка потенциальных источников злонамеренных атак. Для создания надежных распределенных систем в настоящее время требуются серьезные усилия, с привлечением знаний распределенных алгоритмов, методов обеспечения безопасности и из других областей.

С технической точки зрения, были изменены синтаксис модулей и имена некоторых событий, чтобы сделать алгоритмы более структурированными. Теперь модуль может существовать сразу в нескольких экземплярах внутри алгоритма, и в связи с этим каждый экземпляр получает свой уникальный идентификатор. Мы считаем, что это упростило представление некоторых важнейших алгоритмов.

В первое издание был включен вспомогательный набор действующих примеров реализации алгоритмов на языке Java, использующих фреймворк *Appia*. Эти примеры доступны для загрузки на веб-сайте книги.

Ресурсы в Интернете

Дополнительная информация о книге, включая реализацию многих протоколов из первого издания, учебные презентации, слайды и список опечаток, доступна на веб-сайте книги: http://distributedprogramming.net.

Ссылки

Мы занимаемся исследованием абстракций распределенного программирования уже почти два десятилетия. Основой для этой книги послужили работы многих исследователей в области распределенных вычислений. Но особенно нам хотелось бы отметить Лесли Лэмпорта (Leslie Lamport) и Нэнси Линч (Nancy Lynch), сформулировавших интереснейшие проблемы в распределенных вычислениях, и ученых из Корнельского университета, занимавшихся исследованиями надежности распределенных вычислений, включая Озалпа Бабайоглы (Özalp Babaoğlu), Кена Бирмана (Ken Birman), Кейта Марцзулло (Keith Marzullo), Роберта ван Ренеше (Robbert van Rennesse), Рика Шлихтинга (Rick Schlichting), Фреда Шнайдера (Fred Schneider) и Сэма Туега (Sam Toueg).

Прямое или косвенное отношение к этой книге имеют также многие другие исследователи. Мы постарались никого не забыть и добавить ссылки на их работы повсюду в этой книге. Все главы заканчиваются примечаниями, где приводятся информация по теме и исторические ссылки. Цель этих примечаний состоит в том, чтобы подсказать, какие еще работы можно прочитать в дальнейшем, проследить историю представленных концепций, а также отдать должное всем, кто причастен к их созданию и разработке. В конце книги приводится список книг для будущего чтения, посвященных другим аспектам распределенных вычислений.

Благодарности

Мы хотели бы выразить глубокую благодарность нашим студентам и аспирантам Федеральной политехнической школы Лозанны (École Polytechnique Fédérale de Lausanne, EPFL) и Лиссабонского университета (University of Lisboa, UL) за рецензирование рукописей этой книги. На самом деле они не имели выбора и в любом случае должны были сдавать экзамены нам! Но они были весьма снисходительны к ошибкам и опечаткам, которых было немало в первых версиях книги и сопроводительных слайдах, и они представили множество очень ценных замечаний.

Парта Дутта (Partha Dutta), Корин Хэри (Corine Hari), Михаль Капалка (Michal Kapalka), Петр Кузнецов (Petr Kouznetsov), Рон Леви (Ron Levy), Максим Монод (Maxime Monod), Бастиан Пошон (Bastian Pochon) и Джаспер Спринг (Jesper Spring), аспиранты школы информационных технологий и теории связи (School of Computer and Communication Sciences) университета EPFL, Филипе Араужо (Filipe Araujo) и Хъюго Миранда (Hugo Miranda), аспиранты из группы «Распределенные алгоритмы и сетевые протоколы» (Distributed Algorithms and Network Protocol, DIALNP) кафедры информатики (Departamento de Informatica)

факультета естественных наук Лиссабонского университета (Faculdade de Ciências da Universidade de Lisboa), Лейла Халил (Leila Khalil) и Роберт Басмаджан (Robert Basmadjian), аспиранты из Ливанского университета в Бейруте (Lebanese University), а также Али Годши (Ali Ghodsi), аспирант из Шведского института компьютерных наук (Swedish Institute of Computer Science, SICS) в Стокгольме предложили массу улучшений к алгоритмам, представленным в книге.

Несколько реализаций для «практической» части книги было разработано Александром Пинто (Alexandre Pinto) или с его помощью, ключевым членом проекта *Appia*, при участии некоторых преподавателей и студентов группы DIALNP, включая Нуно Крвальо (Nuno Carvalho), Марию Жоао Монтейро (Maria João Monteiro) и Луиса Сардинья (Luís Sardinha).

Наконец, мы хотели бы поблагодарить всех наших коллег, которые оказали нам любезность, передав свои комментарии к первым рукописям книги. В их числе: Феликс Гартнер (Felix Gaertner), Бенуит Гарбинато (Benoit Garbinato) и Мартен ван Стин (Maarten van Steen).

Благодарности ко второму изданию

Работа над вторым изданием книги началась в 2009 году, когда Кристиан Качин (Christian Cachin), работавший тогда в IBM Research (EPFL), находился в ежегодном отпуске. Мы благодарны EPFL и IBM Research за поддержку.

Мы снова хотим сказать спасибо студентам из EPFL и Лиссабонского университета, которым довелось работать с книгой, за предложенные улучшения к первому изданию. Мы также выражаем свою благодарность студентам из высшего технологического института (Instituto Superior Técnico, IST) Лиссабонского технического университета (Universidade Técnica de Lisboa), федерального технологического института в Цюрихе (ETH Zurich) и EPFL, на чей суд были представлены рукописи и дополнительные материалы, включенные во второе издание, за их ценные замечания.

Мы благодарны многим внимательным читателям первого издания и тем, кто прислал свои комментарии к рукописям второго издания, за подмеченные проблемы и предложения по улучшению. В частности, мы благодарны Зинаиде Бенсон (Zinaida Benenson), Элиссон Бессани (Alysson Bessani), Диего Бьюррун (Diego Biurrun), Филиппе Кристову (Filipe Cristóvão), Дэну Добре (Dan Dobre), Феликсу Фрейлингу (Felix Freiling), Али Годши (Ali Ghodsi), Сейфу Хариди (Seif Haridi), Матусу Харвану (Matus Harvan), Рудигеру Капице (Rüdiger Kapitza), Николу Кнежевичу (Nikola Knežević), Андреасу Кнобелю (Andreas Knobel), Михаю Летиа (Mihai Letia), Томасу Локеру (Thomas Locher), Хейну Мелингу (Hein Meling), Хьюго Миранде (Hugo Miranda), Луису Пина (Luís Pina), Мартину Шобу (Martin Schaub) и Марко Вуколичу (Marko Vukolić).

Кристиан Качин,
Рашид Гуерру,
Луис Родригес

Глава 1

Введение

Я полностью использую свои возможности. По-моему, именно к этому стремится любое разумное существо.

(HAL 9000)

В этой главе сначала будет обоснована необходимость абстракций распределенного программирования. Особое внимание при этом мы уделим абстракциям, охватывающим задачи, которые лежат в основе устойчивых форм взаимодействий множественных процессов в распределенной системе и обычно называются *абстракциями соглашений* (agreement abstractions). Затем будет представлена модульная стратегия разработки распределенных программ с использованием упомянутых абстракций и специализированных прикладных программных интерфейсов (Application Programming Interfaces, API).

Также будет дан конкретный пример API для иллюстрации обозначений и схемы вызовов на основе событий, используемых на протяжении всей книги для описания алгоритмов, реализующих абстракции. Обозначения и схемы вызовов очень близки к тем, что можно найти в практических реализациях распределенных алгоритмов.

1.1. Обоснование

Под распределенными вычислениями понимаются алгоритмы для множества процессов, целью которых является организация некоторой формы сотрудничества. Помимо одновременного выполнения, некоторые процессы в распределенной системе могут прекратить работу, например в результате аварии или разрыва соединения, тогда как другие могут продолжать действовать. Само понятие такого *частичного отказа* является одной из характеристик распределенной системы. В действительности это понятие может пригодиться, когда возникнет необходимость обозначить отличие распределенной системы от параллельной системы. Здесь уместно процитировать Лесли Лэмпорта (Leslie Lamport):

> *«Распределенной называется система, в которой отказ компьютера, о существовании которого вы даже не подозревали, может сделать ваш собственный компьютер бесполезным».*

Когда подмножество процессов терпит аварию или теряет связь, перед процессами, которые продолжают действовать или сохраняют связь с большинством, вста-

ет задача синхронизировать свою деятельность согласованным способом. Иными словами, сотрудничество процессов должно быть достаточно устойчивым, чтобы допускать возможность частичных отказов, а иногда и злонамеренных нападений. Это делает распределенные вычисления сложной, но чрезвычайно интересной задачей. Из-за асинхронности процессов, возможности отказов в инфраструктуре взаимодействий и, может быть, даже преднамеренных действий скомпрометированных процессов порой нельзя точно классифицировать отказы; в частности, невозможно отличить отказ процесса от отказа сети, о чем будет рассказываться далее в книге. Хуже того, процесс, находящийся под контролем злоумышленника, может целенаправленно нарушать связь между остальными процессами. Это делает задачу обеспечения согласованного сотрудничества еще более сложной. Задача распределенных вычислений как раз и заключается в создании алгоритма, который будет обеспечивать остающиеся процессы непротиворечивой информацией, благодаря чему они смогут взаимодействовать и решать общие задачи.

Фактически многие современные программы, которые мы используем, являются распределенными программами. Простые повседневные программы, такие как клиенты электронной почты или веб-браузеры, вовлекают распределенные вычисления в некоторой форме. Однако чаще в таких приложениях используется самая простая разновидность распределенных вычислений: вычисления в архитектуре *клиент-сервер*. В архитектуре клиент-сервер существует центральный процесс – *сервер*, – обслуживающий множество удаленных *клиентов*. Клиенты и сервер взаимодействуют, обмениваясь сообщениями и, как правило, следуя форме взаимодействий типа запрос-ответ. Например, чтобы отобразить веб-страницу, браузер посылает запрос веб-серверу и ожидает получить от него ответ с информацией для отображения. Основная проблема, связанная с достижением непротиворечивой формы сотрудничества при ненулевой вероятности частичных отказов, может проявиться даже в такой простой форме взаимодействий. Вернемся к примеру с веб-браузером. Было бы разумным ожидать, что пользователь сможет продолжить обзор Всемирной паутины, если запрошенный веб-сервер потерпит аварию (пользователь автоматически будет подключен к другому веб-серверу). Еще более разумным было бы ожидать, что серверный процесс продолжит передавать информацию другим клиентским процессам, даже если часть из них потерпит аварию или потеряет соединение с сетью.

Проблемы выше уже не тривиальны, когда распределенные вычисления ограничиваются всего двумя взаимодействующими сторонами, такими как клиент и сервер. Однако распределенные вычисления не ограничиваются схемой взаимодействий клиент-сервер. Очень часто во взаимодействиях участвует более двух процессов, деятельность которых требуется синхронизировать для решения общей задачи. Наличие множества процессов еще больше осложняет организацию распределенных вычислений. Иногда мы будем называть *многосторонние* взаимодействия общим случаем. Фактически оба шаблона могут вполне естественно сосуществовать друг с другом. Более того, многие распределенные приложения имеют части, взаимодействующие по схеме клиент-сервер, а другие – по схеме

многосторонних взаимодействий. Часто определение схемы (шаблона) зависит от точки зрения. Например, когда клиент подключается к серверу, чтобы получить услугу, он может не знать, что для оказания запрошенной услуги серверу придется самому послать запрос одному или нескольким вспомогательным серверам. Иногда, чтобы подчеркнуть наличие центрального сервера, используют термин *вычисления типа точка-точка* (peer-to-peer computing).

1.2. Абстракции распределенного программирования

Подобно улыбке, круг абстракций ограничен очень небольшим числом их естественных разновидностей. Определяя свойства, общие для большого или существенного диапазона систем, абстракции помогают отличать основное от второстепенного и избавляют проектировщиков систем и инженеров от необходимости повторно изобретать одни и те же решения для мало отличающихся вариантов одной и той же задачи.

От простого... Рассуждения о распределенных системах должны начинаться с абстрагирования физической системы, лежащей в основе: описания соответствующих элементов максимально абстрактным способом, описания их внутренних свойств и характеристики взаимодействий между ними, в результате чего должна получиться так называемая *модель системы* (system model). В этой книге в основном используются две абстракции, представляющие физическую систему: *процессы* (processes) и *связи* (links).

Процессы в абстракции распределенной программы – это активные сущности, выполняющие вычисления. Процессом может быть компьютер, процессор в компьютере или конкретное ядро в процессоре. В контексте сетевой безопасности процессом может быть также доверенный домен, или административный модуль. Для совместной работы над некоторой общей задачей процессам обычно требуется возможность обмениваться сообщениями посредством некоторой сети. Связи как раз являются абстракцией физических и логических сетей, обеспечивающих обмен информацией между процессами. Различные реализации распределенных систем можно представить этими двумя абстракциями, описав разные свойства процессов и связей, например как эти элементы могут действовать или отказывать при разных условиях.

В главе 2 будет представлено более глубокое обсуждение разных моделей распределенных систем, используемых в этой книге.

...к сложному. Получив модель системы, следующий шаг состоит в том, чтобы понять, как определить абстракции, описывающие повторяющиеся шаблоны взаимодействий в распределенных приложениях. В этой книге нас интересуют абстракции, решающие проблемы надежности взаимодействий в группах процессов, как наиболее важные. Взаимодействие процессов иногда можно смоделировать как задачу *распределенных соглашений* (distributed agreement). Например, процессам может потребоваться выяснить, произошло ли (или не произошло) неко-

торое событие, чтобы договориться о последовательности дальнейших действий (из числа нескольких альтернатив) или о порядке обработки входных данных. Желательно последовательно устанавливать более сложные формы соглашений, от решений до простых задач соглашений. Рассмотрим, к примеру, следующие ситуации.

Чтобы процессы имели возможность обмениваться информацией, они прежде должны договориться об идентификации себя (например, с использованием IP-адресов в Интернете) и общих форматах представления сообщений. Им может также потребоваться договориться о способах обмена сообщениями (например, использовать для взаимодействий надежный поток данных, такой как TCP в Интернете).

После обмена некоторыми сообщениями процессы могут столкнуться с несколькими альтернативными планами действий. Им может потребоваться добиться *консенсуса* в выборе одной из нескольких альтернатив, и каждый из договаривающихся процессов может первоначально иметь собственный план, отличный от планов других процессов.

В некоторых случаях сотрудничающие процессы могут пойти на некоторый шаг, только если все другие процессы согласятся выполнить его. Если это условие не соблюдается, все процессы должны согласиться не выполнять этого шага. Такая форма соглашений чрезвычайно важна для обработки распределенных транзакций, где данная задача известна под названием *атомарное подтверждение* (atomic commitment).

Процессам может потребоваться не только договориться (согласиться) о каких-то действиях, но и о порядке их выполнения. Эта форма соглашения является основой одного из фундаментальных приемов репликации вычислений для отказоустойчивости, который называется задачей *рассылки в едином порядке* (total-order broadcast).

Данная книга рассказывает, как преодолеть сложности, кроющиеся в этих задачах, и определить *абстракции*, инкапсулирующие эти задачи. Сложность задач объясняется необходимостью координации действий множества процессов; учитывая, что процессы могут терпеть неудачу или даже проявлять злонамеренное поведение, создать такие абстракции не всегда просто. Ниже мы обоснуем уместность некоторых абстракций, рассматриваемых в книге. Мы будем различать случаи, когда абстракции вытекают из естественного распределения приложения, с одной стороны, и когда абстракции являются следствием технического решения о необходимости распределения – с другой.

1.2.1. Естественно распределенные системы

Приложения, требующие совместного использования или распределения информации между несколькими сотрудничающими процессами, являются благодатной почвой для проблем, решение которых требует применения абстракций распределенного программирования. Примерами таких приложений могут служить механизмы распределения информации, многопользовательские коллективные систе-

мы, распределенные разделяемые пространства, системы управления процессами (технологическими), коллективные редакторы, распределенные базы данных и распределенные системы хранения.

Распределение информации. В распределенных системах, опирающихся на распространение информации, процессы могут играть одну из следующих ролей: источники (производители) информации, которых также называют *издателями* (publishers), и потребители информации, которых также называют *подписчиками* (subscribers). Парадигма взаимодействий таких процессов часто называется *публикация–подписка* (publish–subscribe).

Издатели производят информацию в форме извещений. Подписчики регистрируют свою заинтересованность в получении определенных извещений. Существуют разные варианты парадигмы публикация–подписка, по-разному удовлетворяющие интересы подписчиков, в том числе на основе каналов, на основе тем, на основе содержимого или на основе типа подписки. Независимо от метода подписки весьма вероятно, что в получении одних и тех же извещений будет заинтересовано несколько подписчиков. В этом случае мы обычно заинтересованы в доставке всем подписчикам одной и той же информации – одного и того же множества сообщений. Иначе система будет несправедлива в оказании услуг, так как одни подписчики получат больше информации, чем другие.

Если только это не является свойством надежности, обеспечиваемым инфраструктурой по умолчанию (что встречается крайне редко), издатель и подписчики должны договориться о том, какие сообщения доставлять, а какие нет. Например, когда распространяется аудиопоток, процессы обычно заинтересованы в получении большей части информации и терпимо относятся к потере ограниченного числа сообщений, особенно если это позволит системе добиться большей пропускной способности. Соответствующую абстракцию обычно называют *негарантированной рассылкой, или рассылкой с максимально доступным качеством* (best-effort broadcast).

Для распространения информации, производимой фондовой биржей, может потребоваться более надежный способ доставки, который называют *надежной рассылкой* (reliable broadcast), так как желательно, чтобы все активные процессы приняли одну и ту же информацию. От инфраструктуры рассылки биржевой информации можно было бы также потребовать рассылки информации в определенном порядке. В некоторых приложениях типа издатель–подписчик источники и потребители информации взаимодействуют опосредованно, через промежуточных брокеров. В таких случаях абстракции соглашений (agreement abstractions) могут использоваться для организации взаимодействий между несколькими брокерами.

Управление процессами. Приложения управления (технологическими) процессами – это такие приложения, в которых несколько программных процессов управляют технологическими процессами. Программные процессы могут, например, управлять параметрами движения авиалайнера или поезда, температурой в ядерном реакторе или автоматизировать работу конвейера автомобильного завода.

Обычно каждый программный процесс подключен к некоторому датчику. Процессам может потребоваться, например, обмениваться значениями, прочитанны-

ми с датчиков, и выводить некоторое общее значение, допустим координаты самолета, на экран дисплея пилота, даже при том что из-за погрешностей или отказов локальных датчиков они могут получать немного разные значения. Такое сотрудничество должно осуществляться всегда, несмотря на то что некоторые датчики (или соответствующие им программные процессы) вышли из строя или не получают никаких данных. Подобного рода сотрудничество можно упростить, если все процессы договорятся о едином множестве входных данных для управляющего алгоритма, – требование, зафиксированное абстракцией *консенсуса*.

Коллективные системы. Пользователи на разных узлах сети могут совместно работать над созданием общего программного обеспечения или документа, или просто участвовать в распределенном диалоге, например в чате или виртуальной конференции. Поддержку совместного труда в таких системах очень эффективно обеспечивает абстракция *общего рабочего пространства*. Такого рода абстракции общей распределенной памяти обычно предоставляют пользователям операции *чтения/записи* для эффективного обмена информацией и ее хранения. В простейшей форме общее рабочее пространство можно представить как единый виртуальный неструктурированный объект-хранилище. Более сложные реализации общих рабочих пространств могут добавлять структурные компоненты для создания пространств отдельных пользователей. Примеров таких реализаций можно привести множество: от разнообразных Вики (Wiki) до сложных многопользовательских распределенных файловых систем. Для поддержания целостности таких совместных рабочих пространств процессы должны договариваться об относительном порядке операций *чтения* и *записи* в пространстве.

Распределенные базы данных. Базы данных составляют еще один класс приложений, где могут пригодиться абстракции соглашений, чтобы гарантировать получение всеми диспетчерами транзакций непротиворечивого представления выполняющихся транзакций и помочь принять решение об упорядочении этих транзакций.

Кроме того, такие транзакции можно использовать для координации действий диспетчеров транзакций, когда решается вопрос о прямых результатах транзакций. То есть серверы баз данных, выполняющие данную распределенную транзакцию, должны координировать свои действия и решать, подтвердить транзакцию или прервать ее. Они могут решить прервать транзакцию, если какой-либо из серверов баз данных обнаружит угрозу целостности данных, несогласованное управление параллельным выполнением, дисковую ошибку или просто аварию какого-то другого сервера баз данных. Как уже отмечалось выше, такое распределенное взаимодействие обеспечивает абстракция *атомарного подтверждения* (atomic commit).

Распределенные системы хранения. Системы хранения огромной емкости распределяют данные между несколькими узлами хранения, каждый из которых является частью общего пространства хранилища. Доступ к хранимым данным обычно сопряжен с обращениями к нескольким узлам, потому что даже единственный элемент данных может оказаться разбросанным по нескольким узлам. Элемент данных может подвергаться сложным преобразованиям программными

компонентами определения и/или исправления ошибок, которые сами могут быть разбросаны по множеству узлов для обеспечения стойкости системы хранения против потери или повреждения некоторых узлов. Такие системы распределяют данные не только из-за ограниченной емкости отдельных узлов, но также для увеличения отказоустойчивости всей системы и снижения нагрузки на отдельные узлы.

Концептуально распределенная система хранения напоминает абстракцию общей памяти, доступ к которой осуществляется посредством операций *чтения* и *записи*, подобно общему рабочему пространству, как упоминалось выше. Но, так как распределенность используется также с целью увеличения общей надежности, распределенные системы хранения сочетают в себе черты естественно распределенных систем с чертами искусственно распределенных систем, о которых рассказывается в следующем разделе.

1.2.2. Искусственно распределенные системы

Часто для реализации приложений, не имеющих естественных предпосылок к распределению, также используются сложные абстракции распределенного программирования. Иногда это является следствием технического решения с целью удовлетворить такие требования, как *отказоустойчивость*, *балансировка нагрузки* или обеспечение *быстрого доступа* при большом числе клиентов.

Проиллюстрируем эту идею на примере *репликации автоматов* (state-machine replication) – мощного средства достижения отказоустойчивости в распределенных системах. Если говорить кратко, репликация автоматов используется для создания служб с высокой доступностью, реализованных в виде нескольких действующих копий службы на разных машинах, которые, как предполагается, не могут выйти из строя одновременно. За счет этого обеспечивается непрерывность обслуживания, даже в случае выхода из строя нескольких машин. Никакого специального аппаратного обеспечения для этого не требуется: отказоустойчивость репликации достигается исключительно программными средствами. В действительности прием репликации можно также использовать внутри информационной системы для ускорения доступа к данным, помещая действующие копии службы как можно ближе к процессам, которые наиболее вероятно будут выполнять запросы. Для служб, доступных для атак через Интернет, например, тот же подход может повысить устойчивость к злонамеренным вторжениям, компрометирующим ограниченное число узлов, участвующих в репликации.

Для большей эффективности репликации разные копии должны поддерживать состояние непротиворечивости. Если состояния копий могут отклоняться произвольно, бессмысленно говорить о репликации. Иллюзия *единой* службы с высокой степенью доступности быстро развалится, и взамен ее появится несколько распределенных служб, каждая из которых может терпеть неудачу независимо от других. Если перечень состояний копий четко определен, один из самых простых способов гарантировать целостность и непротиворечивость всех узлов заключается в передаче им всем одного и того же набора запросов в одном и том же порядке.