Посвящается моей жене Элисон и детям.
Спасибо вам за терпение и поддержку.
— Франческо

Посвящается Дули и Эду за то, что учили меня, «как».
А ещё Синди, Райану, Эрин, Эндрю и Джейку за то,
что были той причиной, «зачем».
— Стив

Посвящается Джо, Майку, Роберту за то,
что сделали тот самый телефонный звонок.
— Франческо и Стив

Содержание

Введение

Платформа Erlang/OTP в наши дни выросла в замечательную и полезную систему. Отчасти причиной этого стало то, что её открытый исходный код продолжает привлекать внимание программистов со всего мира. Тот факт, что программисты из разных культур, с разной историей и опытом, получили возможность вносить в неё свои идеи, исправления и свой код, помогает увеличить широту возможностей и гибкость системы. В свою очередь это помогает всему сообществу, собравшемуся вокруг Erlang/OTP. Если вы ещё не знакомы с этим сообществом, то мы уверены, что вы сочтёте его одним из самых приятных, полезных и начитанных сообществ среди всех языков программирования.

Мы рады и гордимся тем, что наша книга теперь стала доступна русскоязычным энтузиастам Erlang. Мы надеемся, что детали и рекомендации, представленные в этой книге, помогут вам на вашем пути к становлению в роли эксперта по Erlang/OTP. Возможно вы станете активным и известным участником сообщества, а ваши идеи и код помогут новичкам разобраться с языком и только улучшат качество и полезность платформы Erlang/OTP.

Об авторах этой книги

Франческо Чезарини является основателем и техническим директором компании Erlang Solutions[1]. Он использовал Erlang ежедневно с 1995 года, с момента начала его карьеры практикантом в компьютерной научной лаборатории Ericsson (CSLab), в которой и появился Erlang. Затем он перешёл в отдел повышения квалификации и консультирования Ericsson, где работал над первой версией R1 платформы OTP, используя её для создания приложений и ключевых продуктов компании. В 1999 году, вскоре после того, как исходный код Erlang был открыт, он основал компанию, которая в наши дни известна под именем Erlang Solutions. С отделениями в семи странах и на трёх континентах другие компании предпочитают её в качестве партнёра по разработке масштабируемых высокодоступных решений.

В роли технического директора Франческо руководит командами разработчиков и консультантов в компании Erlang Solutions и отвечает за стратегии создания продуктов и исследований в компании. Также он является соавтором книги «Программирование на Erlang», опубликованной O'Reilly, перевод которой был выпущен издательством «ДМК Пресс». Более десяти лет Франческо преподавал в IT-университете Гётеборга, а с 2010 года вёл курс параллельного программирования в университете Оксфорда. Его можно найти в Твиттере под ником *@FrancescoC*.

Стив Виноски работает разработчиком в компании Arista Networks. Большую часть своей тридцатилетней карьеры он провёл, работая над межплатформенным ПО (*middleware*) и распределёнными вычислительными системами. В 2006 году, после 20 лет разработки таких систем на Java и C++, он открыл для себя Erlang и с тех пор использует его в качестве основного языка разработки. Стив участвовал во множестве известных проектов, таких как база данных Riak и веб-сервер Yaws. Также он внёс в исходный код Erlang/OTP десятки новых возможностей и исправлений ошибок.

Стив также, как сам, так и совместно с другими авторами, выпустил более сотни статей и публикаций на темы межплатформенного ПО, распределённых систем и веб-разработки, а также пару книг. Он вёл колонку «The Functional Web» («Функциональная интернет-паутина») в журнале «*IEEE Internet Computing*» с 2008 по 2012 год, а до того с 2002 года вёл в том же журнале колонку «Toward Integration» («Путь к интеграции»). Стив является членом редакторской команды журнала. Между 1995 и 2005 годом он был соавтором колонки «Object Interconnections» («Взаимосвязи объектов») для «*C++ Report*» и позднее для «*C/C++ Users Journal*». В течение прошедших лет Стив также принял участие в сотнях конференций, отдельных презентаций и уроков по межплатформенному ПО, распределённым системам, веб-разработке и языкам программирования, а также был участником программного комитета в десятках конференций и семинаров.

---

[1] http://www.erlang-solutions.com/.

Колофон

Животное на обложке «Проектирования масштабируемых систем с помощью Erlang/OTP» – это европейская морская камбала (*pleuronectes platessa*), известный вид камбалы. Морская камбала живёт вдоль береговой линии Европы от Средиземного моря до Баренцева моря. Обычно их можно найти на глубине от 10 до 50 метров, где они закапываются в песчаное или илистое дно.

Европейская морская камбала имеет тёмно-зелёную или коричневую чешую и покрыта оранжевыми пятнами. Взрослые особи могут достигать одного метра в длину, но большинство дорастает лишь до половины метра. Питается рыба морскими червями, двустворчатыми моллюсками и ракообразными.

Камбала является одним из основных продуктов северогерманской и датской кухни и обычно добывается рыбаками по всей Европе. Особенно она популярна в Дании в жареном виде с картофелем фри под соусом ремулад. Европейская морская камбала была под угрозой истребления в 1970–1980-х годах, но благодаря усилиям сбережения популяции её численность растёт и в 2012 году оценивалась на наивысшем уровне с 1957 года.

Множество животных на обложках книг издательства O'Reilly находится под угрозой, все они важны для нашего мира. Чтобы узнать подробнее о том, как вы можете им помочь, прочтите информацию на сайте animals.oreilly.com.

Предисловие

Эта книга является тем, что вы получите, если посадите вместе Erlang-энтузиаста, который работал над OTP версии R1 в 1996 году, и специалиста по распределенным системам, который спустя десять лет обнаружил то, как Erlang/OTP позволяет вам сосредоточиться на реальных сложностях развития систем, избегая случайных трудностей.

Описывая, как строятся поведения OTP и зачем они нужны, мы покажем вам, как использовать их для проектирования архитектуры автономных узлов. В нашем первоначальном предложении издательству O'Reilly здесь мы и остановились. Однако при написании книги мы решили поднять планку выше, задокументировав наши практики, проектные решения и распространённые ошибки при проектировании распределенных систем. Эти шаблоны посредством ряда проектных решений и компромиссов дают нам те масштабируемость, надёжность и доступность, которыми славится Erlang/OTP. Вопреки распространенному мнению, это не происходит волшебным образом или не появляется из коробки, но, конечно, это гораздо проще, чем с любым другим языком программирования, который не имитирует семантику Erlang и не выполняется на виртуальной машине BEAM.

Франческо: Почему появилась эта книга?

Кто-то однажды сказал мне, что написать книгу – это как завести детей. После того как вы написали одну книгу и уже держите в руках бумажную копию, волнение захлёстывает вас, и вы быстро забываете напряжённую работу и принесённые жертвы и хотите начать писать следующую. Я был намерен написать продолжение «*Программирования Erlang*» (издательство O'Reilly), начиная с того момента, как я получил в руки её первую бумажную копию в июне 2009 года. Я не имел тогда собственных детей, когда начал этот проект, но в конце концов проект оказался настолько длинным, что в момент выхода этой книги на подходе уже мой второй ребёнок. Кто сказал, что хорошие вещи не стоят того, чтобы их ждать?

Как и в случае с первой книгой, для «*Проектирования масштабируемых систем с помощью Erlang/OTP*» мы взяли за основу примеры из учебных материалов по OTP, разработанных компанией Erlang Solutions. Я использовал примеры и начал объяснять их, преобразовывая попутно мои лекции и подход к обучению в слова книги. Когда я закончил одну главу, я вернулся и перепроверил те части, с которыми у студентов нередко возникали трудности. Вопросы, которые часто задавались лучшими студентами, в конечном итоге оказались в боковых сносках, а длинные главы были разделены на более мелкие. Все шло хорошо, пока мы не добрались до глав 11 и 12, потому что не существовало стандартного лучшего способа работы с релизами и обновлениями ПО. Скорее, имелись инструменты, мно-

жество инструментов. Некоторые были интегрированы в цикл сборки и выпуска кода нашего клиента, другие же работали просто из коробки. Некоторые были непригодными для использования. Мы надеемся, что эти главы станут наилучшим руководством для тех, кто хочет понять, как релизы обновления ПО работают за кулисами. Они также объясняют, что вам нужно знать для поиска и устранения проблем в существующих инструментах или для написания своих собственных.

Но реальные проблемы начались с главы 13. Не имея примеров или учебных материалов, мне пришлось формализовать те знания, которые были в наших головах, и документировать подходы, которые мы предпринимаем при проектировании систем на Erlang/OTP, пытаясь привести их в соответствие с теорией распределенных вычислений. Глава 13 превратилась в четыре главы, на которые ушло столько же времени, сколько и на первые десять глав. Для тех из вас, кто купил версию в раннем доступе, я надеюсь, что ожидание того стоило. Те, кто мудро ждал окончания работы над книгой, прежде чем купить себе копию, – наслаждайтесь!

Стив: Почему появилась эта книга?

Я впервые открыл для себя Erlang/OTP в 2006 году, исследуя способы, как сделать разработку программного обеспечения интеграции предприятия быстрее, дешевле и лучше. Независимо от того, с какой стороны я смотрел на него, Erlang/OTP явно превосходил языки C++ и Java, которые я и мои коллеги давно использовали на то время. В 2007 году я присоединился к новой компании и начал использовать Erlang/OTP для создания коммерческого продукта, и оказалось, что Erlang исполнил всё, что о нём говорили мои предыдущие расследования. Я обучил языку некоторых моих коллег, и спустя некоторое время горстка разработчиков смогла продолжить разработку ПО, оказавшегося в итоге более функциональным, более надёжным, легче развивающимся и готовым к релизу гораздо быстрее, чем аналогичный код, сделанный значительно большей командой программистов на C++. И по сей день я по-прежнему полностью убежден во впечатляюще практической эффективности Erlang/OTP.

С годами я опубликовал немало технических материалов, и предполагаемой целевой аудиторией всегда были другие практикующие разработчики вроде меня. И эта книга не исключение. В первых 12 главах мы даём глубокую детализацию, которая нужна практикующим разработчикам, для того чтобы полностью понять основные принципы OTP. К этим деталям мы подмешиваем крупицы практических знаний – модули, функции и подходы, которые сэкономят вам значительное время и усилия в вашей повседневной работе по проектированию, разработке и отладке. В заключительных четырёх главах мы переводим дух и фокусируемся на общей картине тех компромиссов, которые встречаются в разработке, развертывании и эксплуатации устойчивых масштабируемых распределенных приложений. Из-за ошеломляющего количества знаний, подходов и компромиссов, на которые приходится идти в распределённых системах, отказоустойчивости и DevOps, написание этих глав в лаконичном стиле оказалось трудным, но я считаю, что нам

удалось нащупать верный баланс, предоставив много полезных советов и не потерявшись в несущественных деталях.

Я надеюсь, что эта книга поможет вам повысить качество и полезность программного обеспечения и систем, которые вы разрабатываете.

Кому следует прочесть эту книгу

Целевая аудитория этой книги включает разработчиков и архитекторов на языках Erlang и Elixir, которые прочитали, по крайней мере, одну вступительную книгу по языку и готовы поднять свои знания на следующий уровень. Это не та книга, с которой можно начать изучение, но скорее такая книга, которая подхватит вас с того места, где предыдущая книга закончилась. В главах с 3 по 12 материал основан на материале предыдущей главы, и эти главы следует читать последовательно, так же и главы 13–16. Если вам не нужны основы Erlang, то не стесняйтесь пропустить главу 2.

Как читать эту книгу

Мы писали эту книгу для Erlang версии 18.2. Большинство функций, которые мы описываем, будет работать и с более ранними выпусками, а те особенности, которые не совместимы, отмечены отдельно. В настоящее время неизвестно о несовместимостях с будущими релизами, если такие появятся, мы опишем их подробно на странице найденных ошибок на сайте книги и поправим код в github-репозитории книги. Вам предлагается скачать примеры к книге из нашего репозитория на github[1] и запускать их самостоятельно для лучшего понимания.

Благодарности

Написание этой книги оказалось долгим путём. Пока мы по нему шли, мы получили большую поддержку от множества замечательных людей. Наш редактор Энди Орам был бесконечным источником идей и предложений, терпеливо ведя нас, давая советы и оказывая постоянную поддержку. Спасибо, Энди, мы не смогли бы сделать это без тебя! Саймон Томпсон, соавтор *«Программирования Erlang»*, помог с идеей книги и заложил основу для второй главы. Большое спасибо Роберту Вирдингу за то, что подарил книге некоторые из примеров. Множество читателей, корректоров и добровольцев давали нам советы, по мере того как мы по каплям кормили их содержимым очередных глав.

Рискуя пропустить кого-то, вот эти люди: Ричард Бен Алея, Роберто Алой, Йеспер Луи Андерсен, Боб Баланс, Ева Бихари, Мартин Бодокки, Наталья Чечина, Жан-Франсуа Клутье, Ричард Кроучер, Виктория Фёрдош, Хайнц Гис, Йоаким Хален, Фред Хеберт, Чаба Хош, Торбен Хофманн, Боб Ипполито, Аман Коули, Ян Виллем Луитен, Джей Нельсон, Робби Рашке, Анджей Слива, Дэвид Смит,

[1] https://github.com/francescoc/scalabilitywitherlangotp.

Сэм Таваколи, Премананд Тангамани, Ян Улиг, Джон Уорвик, Дэвид Уэлтон, Ульф Вигер и Александр Йонг.

Если мы вас пропустили, то примите наши искренние извинения! Напишите нам, и вы будете оперативно добавлены. Отдельный привет передаём персоналу Erlang Solutions за чтение глав по мере их написания и всем остальным, кто представил замеченные ошибки в раннем выпуске. Отдельное спасибо всем тем, кто ободрял нас в социальных медиа, особенно другие авторы. Вы знаете, о ком мы! И наконец, но не в последнюю очередь, благодарности командам по производству, маркетингу и конференциям в издательстве O'Reilly, которые продолжали напоминать нам, что путь не закончен, пока вы не держите в руках бумажную копию. Мы действительно ценим вашу поддержку!

Соглашения, используемые по тексту этой книги

Следующие типографические соглашения используются в этой книге:

Курсив
: Указывает новые термины, приложения, URL-ссылки, адреса электронной почты, имена файлов, каталогов и расширения файлов.

`Моноширинный шрифт`
: Используется для листингов программ, а также внутри текста для обозначения элементов программ, таких как имена функций, базы данных, типы данных, переменные окружения, операторы и ключевые слова. Также используется для поведений, команд и параметров командной строки.

`Моноширинный жирный`
: Выделяет команды или прочий текст, который должен быть введён пользователем вручную.

`Моноширинный курсив`
: Выделяет текст, который следует заменить данными пользователя, требующимися по ситуации.


Этот значок указывает на подсказку или совет.


Этот значок означает некоторое замечание.


Этот значок ставится рядом с предупреждением или в опасном месте.

Пользовательский код

Дополнительные материалы (примеры кода, упражнения и т. д.) доступны для скачивания по адресу https://github.com/francescoc/scalabilitywitherlangotp.

Эта книга сделана для того, чтобы помочь вам выполнить вашу работу. В целом вы можете свободно пользоваться кодом из этой книги в ваших программах или документации. Не нужно спрашивать у нас разрешения, кроме тех случаев, когда вы копируете значительную часть кода. Например, для написания программы с использованием нескольких фрагментов кода из этой книги не требуется спрашивать разрешения. Продажа или распространение дисков с примерами из книг O'Reilly требует наличия разрешения. Ответ на чей-то вопрос с помощью цитаты или фрагмента кода из книги не требует разрешения. Копирование значительного количества примеров из этой книги в документацию вашей программы потребует разрешения.

Мы будем благодарны, хотя и не требуем указания нашего авторства. Указание авторства обычно включает название книги, автора, издателя и международный код ISBN. Например: *«Проектирование для масштабируемости с Erlang/OTP»* Франческо Чезарини и Стив Виноски (изд-во O'Reilly). © 2015 Франческо Чезарини и Стивен Виноски.

Если вы считаете, что использование вами примеров из книги выходит за рамки добросовестного использования, обратитесь к представителям издательства по адресу permissions@oreilly.com.

Safari® Books Online

Safari Books Online – это цифровая библиотека, которая позволяет легко выполнять поиск среди 7500 справочников и видеороликов по технологиям и творчеству. Здесь вы быстро сможете найти нужные ответы.

С подпиской вам станет доступна любая книга или видео из нашей онлайн-библиотеки. Читайте книги на вашем мобильном телефоне или планшете. Получайте ранний доступ к материалам до того, как они попадут в печать, а также эксклюзивный доступ к рукописям в процессе их написания с возможностью обратной связи с их авторами. Копируйте и вставляйте примеры кода, организуйте коллекцию избранного, скачивайте главы, ставьте закладки в ключевых секциях, создавайте заметки, печатайте страницы и наслаждайтесь другими возможностями, которые экономят ваше время.

Издательство O'Reilly Media разместило эту книгу в онлайн-сервисе Safari Books Online. Для полного цифрового доступа к этой книге и другим книгам на подобные темы от O'Reilly и других издателей создайте бесплатную учётную запись на сайте http://my.safaribooksonline.com.

Как с нами связаться

Пожалуйста, направляйте комментарии и вопросы по этой книге в адрес издателя:

O'Reilly Media, Inc.
1005 Gravenstein Highway
North Sebastopol, CA 95472
800-998-9938 (звонок из США или Канады)
либо 707-829-0515, 707-829-0104 (факс)

У нас есть веб-страница, посвящённая этой книге, где мы собираем найденные ошибки и любую дополнительную информацию. Вы можете посетить её по этому адресу: http://bit.ly/designing-for-scalability-with-erlangotp.

Чтобы оставить комментарий или задать технические вопросы по этой книге, отправьте письмо на bookquestions@oreilly.com.

Дополнительную информацию о наших книгах, учебных курсах, конференциях, новостях смотрите на сайте издательства: http://www.oreilly.com.

Наша страничка на Facebook: http://facebook.com/oreilly.

Подпишитесь на наш Twitter: http://twitter.com/oreillymedia.

Наш канал на YouTube: http://www.youtube.com/oreillymedia.

Глава 1

Знакомство

Итак, вам требуется создать устойчивую к сбоям, масштабируемую систему мягкого реального времени с требованиями к её высокой доступности. Она должна работать на событиях и реагировать на внешние раздражители, нагрузку и сбои. Она должна всегда продолжать отвечать на запросы. Вы где-то слышали, конечно же, множество историй успеха, которые утверждают, что Erlang является самым подходящим инструментом для такой работы. В самом деле так и есть – но хотя Erlang и является мощным языком программирования, его одного недостаточно, чтобы собрать все эти возможности вместе и строить сложные реактивные системы. Чтобы корректно, быстро и эффективно выполнить данную задачу, вам ещё понадобятся вспомогательное программное обеспечение (middleware), повторно используемые библиотеки, инструментарий, принципы проектирования и программная модель, которая расскажет вам, как построить вашу систему и сделать её распределённой.

В этой книге нашей целью было изучить многогранность доступности и масштабирования систем, а также затронуть близкие темы параллельного исполнения, распределения и устойчивости к сбоям с точки зрения языка программирования Erlang и его библиотеки OTP. Erlang/OTP был создан в лаборатории компьютерных наук компании Ericsson (CS Lab), и целью проекта по его созданию было исследование эффективной разработки нового на то время поколения телекоммуникационных систем в индустрии, где время до выхода продукта на рынок становилось всё критичнее. Это было до прихода Всемирной паутины, до планшетов и смартфонов, MMO-игр, систем обмена сообщениями и эпохи Интернета вещей (IoT).

В то время единственными системами, которые требовали таких уровней масштабируемости и устойчивости к сбоям, которые сегодня мы считаем чем-то само собой разумеющимся, были скучные телефонные коммутаторы. Им приходилось обслуживать массивные всплески объёма звонков в канун Нового года и Рождества, выполнять требования регулятора по обеспечению звонков экстренным службам и избегать болезненных штрафов, которые накладывались на поставщиков, чьё оборудование привело к отказу в обслуживании. С точки зрения простого горожанина, если вы подняли трубку и не услышали сигнала готовности телефонной станции, вы можете быть точно уверены в двух вещах: управляющие

телефонной компании получат серьёзные неприятности, а сбой в обслуживании попадёт на первые страницы утренних газет. Вне зависимости от того, что произошло, этим коммутаторам было запрещено ломаться. Даже когда их компоненты и инфраструктура вокруг них приходили в негодность, приходящие по другим линиям запросы обязаны были быть обслужены. Сегодня регуляторы и штрафы заменились нетерпеливыми пользователями без чувства лояльности, которые, не раздумывая, сменят провайдера, а заголовки на первых страницах газет заменились на массовую истерию в социальных медиа. Но основные проблемы доступности и масштабируемости остаются.

В результате этого коммутаторы связи и подобные им современные системы обязаны реагировать на сбои аналогично тому, как им приходится реагировать на нагрузку или внутренние события. И в то время, пока лаборатория компьютерных наук Ericsson не была готова изобрести новый язык программирования, найденное ими решение этой проблемы оказалось таким новым языком. Это прекрасный пример изобретения языка и модели программирования, которые предназначены для решения конкретного чётко определённого класса задач.

Определение проблемы

Как мы показываем по ходу этой книги, Erlang/OTP уникален среди языков программирования и библиотек и отличается шириной охвата, глубиной и согласованностью возможностей, которые он обеспечивает для масштабируемых систем, устойчивых к сбоям и имеющих требования высокой доступности. Проектирование, реализация, эксплуатация и обслуживание таких системам требуют отдачи всех сил. Команды, достигшие успеха в их построении и запуске, пришли к этому путём перебора этих четырёх фаз, используя подсказки от систем мониторинга и производственных метрик, чтобы найти области в коде, в разработке и эксплуатации, требующие улучшения. Успешные команды также учатся улучшать масштабируемость другими способами, такими как тестирование, экспериментирование и сравнение производительности, а ещё они нажимают на исследования и разработку нужных им характеристик системы. Нетехнические моменты, такие как корпоративные ценности и культура, также могут играть значительную роль в том, соответствуют ли команды требованиям их собственных проектов или даже превосходят эти требования.

Мы использовали термины *распределённый*, *устойчивый к сбоям*, *масштабируемый*, *мягкое реальное время* и *высокодоступный*, чтобы описать системы, которые мы планируем построить с помощью OTP. Но что эти слова обозначают на самом деле?

Масштабируемый ссылается на то, как хорошо система способна адаптироваться к изменениям нагрузки или количества доступных ресурсов. Масштабируемые сайты, к примеру, способны смягчить и обслужить всплески посещений, не теряя запросов клиентов, даже при сбоях в оборудовании. Масштабируемая система чата может быть способна принять тысячи новых пользователей в день, при этом не ухудшив качества обслуживания текущих клиентов.

Термин *распределённый* ссылается на то, как системы группируются вместе и взаимодействуют друг с другом. Кластеры могут проектироваться для горизонтального роста, с добавлением оборудования, доступного в массовой продаже (другими словами, обычного), или когда запускаются дополнительные копии системы на той же или других машинах для лучшего использования доступных ядер процессора. Одиночные машины также могут быть виртуализованы, так что копии операционной системы будут работать поверх другой операционной системы или делить друг с другом доступ к ресурсам оборудования. Добавление вычислительной мощности в кластер баз данных могло бы позволить масштабировать его по количеству данных, которое он сможет хранить, или по количеству запросов в секунду, которые он сможет обработать. Масштабирование в сторону уменьшения часто тоже важно; например, веб-приложение, построенное на облачных технологиях, может задействовать дополнительные мощности в пиковые часы и освободить неиспользуемые серверы, как только волна посетителей спадёт.

Системы, являющиеся *устойчивыми к сбоям*, продолжают вести себя предсказуемо в то время, когда компоненты в их окружении отказывают. Устойчивость к сбоям должна быть заложена в систему с самого начала; даже не думайте добавлять её в конце разработки. А что, если в ваш код вкралась ошибка или состояние программы оказалось повреждено? Или что будет, если ваша сеть отключится или произойдёт аппаратный сбой? Если пользователь посылает сообщение, которое приводит к аварийной остановке процесса, он может быть уведомлён о том, было доставлено сообщение или нет, и может быть уверен, что это уведомление соответствует действительности.

Под *мягким реальным временем* следует понимать предсказуемость ответов и задержек, обслуживание с постоянной пропускной способностью и гарантирование ответа в течение приемлемого времени. Пропускная способность должна оставаться стабильной, несмотря на всплески трафика или количества пришедших одновременно запросов. Не важно, сколько запросов проходит через систему в единицу времени, её пропускная способность не должна падать ниже проектной при тяжёлой нагрузке. Её время ответа, также известное как задержка, должно быть пропорционально количеству параллельно входящих запросов, избегая при этом разброса результатов, возникающего по причине сборки мусора «с остановкой мира» или других узких мест последовательного исполнения. Если пропускная способность вашей системы, например, равняется миллиону сообщений в секунду и внезапно придёт миллион сообщений одновременно, то время обработки должно оказаться равным одной секунде, после чего все получат результат. Но если в момент всплеска придут два миллиона запросов, то не допускается снижение пропускной способности; не некоторые, но все запросы без исключения должны быть обработаны в течение следующих двух секунд.

Высокая доступность минимизирует или совершенно исключает остановку сервиса в результате программных ошибок, обновлений кода или другой операционной деятельности. А что, если процесс упадёт? Что, если система питания вашего центра обработки данных откажет? Есть ли у вас запасной блок питания

или система батарей, дающая вам достаточное время, чтобы перенести ваш кластер или штатно завершить работу серверов, оказавшихся в этой ситуации? Или резервная сеть и резервное оборудование?

Измерили ли вы свою систему и убедились ли, что даже после потери части кластера оставшееся оборудование имеет достаточно ёмкости CPU, чтобы выдержать пиковую нагрузку? Не важно, потеряете ли вы часть вашей инфраструктуры, либо ваш облачный провайдер переживёт стеснительный момент отказа в сервисе, или вы будете заняты работами по обслуживанию, но пользователь, пославший сообщение в чат, желает быть уверенным, что сообщение дойдёт до правильного получателя. Пользователи системы ожидают, что она просто будет работать. Это отличается от *устойчивости к сбоям*, когда пользователю сообщают о том, что отправка не сработала, но вся система при этом не прекращает работу, и ошибка на неё не влияет. Способность Erlang выполнять обновления кода во время выполнения этому помогает. Но если вы задумаетесь о том, что требуется для изменения схемы базы данных или обновления протокола на обратно несовместимый в потенциально распределённом окружении, в то время как на систему продолжают приходить запросы, то простота испаряется. Когда вы работаете со своим онлайн-банком ночью или на выходных, вы хотите уверенности, что не появится это ненавистное сообщение – «закрыто на техническое обслуживание».

Erlang в самом деле способствует решению многих из перечисленных проблем. Но в сухом остатке это всего лишь язык программирования. Для сложных систем, которые вы собираетесь реализовать, вам понадобятся готовые приложения и библиотеки, которые можно использовать сразу из коробки. Вам также понадобятся принципы и шаблоны проектирования, которые наделяют архитектуру вашей системы новой целью – создавать распределённые надёжные кластеры. Вам будут нужны рекомендации по проектированию вашей системы, а также инструменты для реализации, установки, мониторинга, эксплуатации и обслуживания. В этой книге мы постараемся покрыть библиотеки и инструменты, которые позволяют изолировать сбой на уровне узла системы, создать и распределить множество узлов для масштабируемости и высокой доступности.

Вам следует серьёзно задуматься о ваших требованиях и свойствах и постараться выбрать подходящие библиотеки и шаблоны проектирования, которые гарантируют то, что готовая система будет себя вести так, как вам нужно, и будет делать то, что было задумано. В ваших поисках вам придется принять компромиссные решения, которые являются взаимозависимыми, – компромиссы по времени, ресурсам и поддержке функций, компромиссы по доступности, масштабируемости и надежности. Никакая готовая библиотека не сможет вам помочь, если вы сами не знаете, что должна делать ваша система. В этой книге мы проводим вас по шагам к пониманию этих требований и к принятию проектных решений и компромиссов, которые потребуются для их достижения.