# ОГЛАВЛЕНИЕ

# ПРЕДИСЛОВИЕ

Уже давно данные очаровывают меня. Еще на младших курсах, изучая электротехнику, я открыл для себя цифровую обработку сигналов и прикипел к ней всей душой. Я понял, что музыку, фотографии и множество других вещей можно рассматривать как данные. Оказалось, что создавать и усиливать эмоциональные переживания можно с помощью вычислений. Ничего интереснее я даже представить себе не мог.

Со временем передо мной раскрылись новые, не менее удивительные аспекты данных. Вот уже несколько лет, как я занимаюсь социальными сетями и большими массивами данных. Именно большие массивы бросают вызов моему интеллекту. Статистический анализ данных я освоил уже давно, а для работы с новыми типами данных были нужны «всего лишь» новые математические методы. Конечно, это не просто, но по крайней мере я этому учился, а уж в ресурсах для пополнения знаний недостатка и вовсе нет. С другой стороны, для работы с большими объемами данных необходим совсем другой системный подход и новые способы программирования. Этому меня не учили, но – и это даже важнее – не один я оказался в таком положении. Знания о практических методах обработки больших массивов данных сродни черной магии. Существует немало инструментов и приемов масштабируемой обработки данных, в частности, кэширование (например, с помощью программы memcached), репликация, секционирование и, конечно, MapReduce/Hadoop. Последние несколько лет я потратил на приобретение и совершенствование навыков в этой области.

Лично для меня наиболее сложной оказалась средняя часть кривой обучения. Поначалу не так уж трудно найти блоги с вводным материалом и презентации, объясняющие, как написать пример уровня «Здравствуй, мир». А приобретя начальные знания, вы уже можете задавать дополнительные вопросы в списках рассылки, общаться со специалистами на различных встречах и конференциях и даже самостоятельно читать исходный код. Но вот в середине зияет гигантский

провал — аппетит уже разыгрался, а как сформулировать последующие вопросы, неясно. Эта проблема стоит особенно остро для таких новейших технологий, как Hadoop. Необходимо некое упорядоченное изложение, которое начиналось бы с примера «Здравствуй, мир» и доводило читателя до точки, в которой он смог бы без особых усилий применить Hadoop на практике. Именно так я и представляю себе эту книгу. К счастью, я обнаружил, что серия «In Action», выходящая в издательстве Manning, отлично согласуется с моей целью, и к тому же там есть прекрасные редакторы, помогавшие мне на всем протяжении работы.

Я получал удовольствие, когда писал книгу, и надеюсь, что она станет для вас началом увлекательного путешествия в мир Hadoop.

# БЛАГОДАРНОСТИ

В эту книгу внесло вклад много людей. Прежде всего, я хочу поблагодарить Джеймса Уоррена (James Warren). Он возглавлял аналитический отдел в компании RockYou и вместе с ним мы стремились популяризировать Hadoop в этой организации. Я многому научился у него, он даже помогал мне писать ранние варианты текста.

На мое счастье нашлось немало людей, познакомивших меня с интересными примерами за рамками индустрии Web 2.0. В частности, я хочу выразить благодарность Чжи Гуо Люо (Zhiguo Luo), Мень Ху (Meng Xu), Шаолинь Сун (Shaoling Sun), Кену Макиннису (Ken MacInnis), Райану Роусону (Ryan Rawson), Вуку Эрчеговачу (Vuk Ercegovac), Раджасекару Кришнамурти (Rajasekar Krishnamurthy), Срираму Рагхавану (Sriram Raghavan), Фредерику Рейсу (Frederick Reiss), Юджину Шекита (Eugene Shekita), Сандипу Тата (Sandeep Tata), Шивакумару Вайтианатхану (Shivakumar Vaithyanathan) и Хуай Ю Чжу (Huaiyu Zhu).

Я также благодарен рецензентам этой книги. Они прислали ценные отзывы на первые варианты. В частности, технический редактор Пол О'Рорк (Paul O'Rorke) вышел далеко за пределы формальных обязанностей и внес ряд крайне полезных предложений о том, как улучшить рукопись. Я надеюсь, что когда-нибудь он сам напишет книгу. Кроме того, я получал истинное удовольствие от долгих бесед с Джонатаном Као (Jonathan Cao). Его опыт работы с базами данных и крупномасштабными системами позволил мне шире взглянуть на возможности Hadoop и лучше понять их.

По ходу работы текст книги многократно читали и другие рецензенты, которых я хотел бы поблагодарить за их бесценный труд: Пол Стусяк (Paul Stusiak), Филипп К. Дженерт (Philipp K. Janert), Амин Мохаммед-Коулмен (Amin Mohammed-Coleman), Джон С. Гриффин (John S. Griffin), Марко Угетти (Marco Ughetti), Рик Вагнер (Rick Wagner), Кеннет Делонг (Kenneth DeLong), Джош Паттерсон (Josh Patterson), Срини Пенчикала (Srini Penchikala), Костантино Кербо (Costantino Cerbo), Стив Логран (Steve Loughran), Ара Абрахамян

(Ara Abrahamian), Бен Холл (Ben Hall), Эндрю Зимер (Andrew Siemer), Роберт Хансон (Robert Hanson), Кит Ким (Keith Kim), Сопан Шивейл (Sopan Shewale), Марион Стуртеван (Marion Sturtevant), Крис Чендлер (Chris Chandler), Эрик Реймонд (Eric Raymond) и Джероен Бенкхейсен (Jeroen Benckhuijsen).

Мне посчастливилось работать с коллективом чудесных людей в издательстве Manning. Отдельное спасибо Трою Мотту (Troy Mott), который побудил меня взяться за это дело и терпеливо дожидался, пока я его закончу. Спасибо также Таре Уолш (Tara Walsh), Карен Тегтмейер (Karen Tegtmeyer), Марьян Бейс (Marjan Bace), Мэри Пирджис (Mary Piergies) Синтии Кэйн (Cynthia Kane), Стивену Хонгу (Steven Hong), Рашель Шредер (Rachel Schroeder), Кэти Теннант (Katie Tennant) и Морин Спенсер (Maureen Spencer). Поддержка с их стороны была просто феноменальной. Лучшей компании для работы я себе даже представить не могу.

Само собой, добрых слов заслуживают все, кто внес свой вклад в разработку каркаса Hadoop и помогает развиваться сформировавшейся вокруг него экосистеме. Начало положил Дуг Каттинг (Doug Cutting), а компания Yahoo оказалась достаточно прозорливой, чтобы поддержать его с самых первых шагов. Теперь компания Cloudera представляет Hadoop вниманию более широкой аудитории корпоративных пользователей. Сейчас самое подходящее время присоединиться к растущему сообществу Hadoop.

И наконец, я хочу сказать спасибо всем своим друзьям, семье и коллегам, которые поддерживали меня на протяжении работы над книгой.

# ОБ ЭТОЙ КНИГЕ

Hadoop – это каркас с открытым исходным кодом, в котором реализован алгоритм распределения и редукции MapReduce, лежащий в основе подхода Google к организации запросов к распределенным наборам данных, которые и составляют Интернет. В связи с таким определением возникает очевидный вопрос: что такое *распределение* (map) и зачем нужна последующая *редукция* (reduce)? Зачастую массивные наборы данных с трудом поддаются анализу и опросу традиционными средствами, особенно когда сами запросы весьма сложны. По существу, алгоритм MapReduce разбивает запрос и набор данных на несколько частей – это этап распределения. Отдельные компоненты запроса можно обработать параллельно, а затем *свести* полученные результаты воедино (*редуцировать*).

Читатель этой книги узнает, как использовать Hadoop и писать MapReduce-программы. Книга предназначена для программистов, архитекторов и руководителей проектов, связанных с обработкой большого объема данных в офлайновом режиме. Будет описано, как получить копию Hadoop, как организовать кластер и как писать программы анализа. Мы начнем с применения Hadoop в конфигурации по умолчанию к решению нескольких простых задач, например, об анализе изменения частоты вхождения слов в корпус документов; это позволит уяснить основные идеи Hadoop и MapReduce. Далее мы перейдем к базовым концепциям MapReduce-приложений, разрабатываемых с помощью Hadoop, и по ходу дела изучим компоненты каркаса, применение Hadoop к широкому спектру задач анализа данных и многочисленным примерам Hadoop в действии.

Алгоритм MapReduce сложен как концептуально, так и в плане реализации, а от пользователей каркаса Hadoop требуется изучить все средства и хитрости, необходимые для работы с ним. В этой книге не просто рассказывается о том, как запустить Hadoop; прочитав ее, вы научитесь писать полезные программы с использованием MapReduce.

Предполагается, что читатель хотя бы немного владеет языком Java, поскольку именно на нем написано большинство примеров.

Знакомство с основами математической статистики (например, гистограммами и корреляцией) поможет разобраться в более сложных примерах обработки данных.

## Структура книги

Эта книга состоит из 12 глав, разбитых на три части.

Часть 1 состоит из трех глав и представляет собой введение в каркас Hadoop. Здесь излагаются те базовые сведения, которые необходимо знать для понимания и использования каркаса. Описывается, из каких аппаратных компонентов состоит кластер Hadoop, рассказывается об установке и конфигурировании системы. Также в части 1 дается общее представление о каркасе MapReduce и приводится пример первой MapReduce-программы.

Часть 2 «Hadoop в действии» состоит из пяти глав, в которых описаны практические навыки, необходимые для составления и запуска программ обработки данных в среде Hadoop. Здесь мы рассмотрим многочисленные примеры применения Hadoop к анализу набора данных о патентах, в том числе и такие нетривиальные алгоритмы, как фильтр Блума. Мы также поговорим о приемах программирования и администрирования, чрезвычайно полезных при работе с Hadoop в производственной среде.

Часть 3 «Hadoop в реальной жизни», в которую входят последние четыре главы, посвящена обширной экосистеме, сложившейся вокруг Hadoop. Существуют облачные службы, которые позволяют обойтись без покупки собственного оборудования для создания кластера. Имеются также дополнительные пакеты, предлагающие высокоуровневые абстракции, надстроенные над MapReduce. Наконец, мы рассмотрим несколько примеров практического применения Hadoop к решению реальных задач бизнеса.

В приложении приведен список команд HDFS.

## Графическое выделение и загрузка исходного кода

Исходный код в листингах и в основном тексте набран `моноширинным шрифтом`. Многие листинги сопровождаются аннотациями, в которых излагаются важные концепции. В некоторых случаях в листингах присутствуют нумерованные маркеры, с которыми соотносятся последующие пояснения.

Код всех приведенных в книге примеров можно скачать с сайта издательства по адресу www.manning.com/HadoopinAction.

# АВТОР В СЕТИ

Приобретение книги «*Hadoop в действии*» открывает бесплатный доступ к закрытому форуму, организованному издательством Manning Publications, где вы можете оставить свои комментарии к книге, задать технические вопросы и получить помощь от автора и других пользователей. Получить доступ к форуму и подписаться на список рассылки можно на странице www.manning.com/HadoopinAction. Там же написано, как зайти на форум после регистрации, на какую помощь можно рассчитывать, и изложены правила поведения в форуме.

Издательство Manning обязуется предоставлять читателям площадку для общения с другими читателями и автором. Однако это не означает, что автор обязан как-то участвовать в обсуждениях; его присутствие на форуме остается чисто добровольным (и не оплачивается). Мы советуем задавать автору хитроумные вопросы, чтобы его интерес к форуму не угасал!

Форум автора в сети и архивы будут доступны на сайте издательства до тех пор, пока книга не перестанет печататься.

# ОБ АВТОРЕ

В настоящее время Чак Лэм занят организацией социальной сети для пользователей мобильных устройств под названием *RollCall*. Это будет социальный секретарь для активных людей.

Раньше Чак работал техническим руководителем проекта в компании RockYou. В этой должности он разрабатывал социальные приложения и инфраструктуру обработки данных, рассчитанную на сотни миллионов пользователей. Он применял A/B тестирование и статистический анализ для оптимизации скорости распространения социального приложения среди пользователей (его *виральности*). Ему удавалось значительно – иногда на порядок – повысить частоту посещения страниц.

Обработкой больших объемов данных Чак заинтересовался во время работы над кандидатской диссертацией в Стэнфордском университете. Он узнал о том, какой серьезный эффект большие массивы данных могут оказать на машинное обучение, и начал изучать последствия. В его диссертации «Вычислительные методы сбора данных» впервые были исследованы новые подходы к сбору данных для машинного обучения – с применением идей, заимствованных из программ с открытым исходным кодом и онлайновых игр.

# Об иллюстрации на обложке

Рисунок на обложке книги «Hadoop в действии» называется «Юноша из селения Кистане в Далмации». Репродукция взята из альбома традиционной хорватской одежды середины девятнадцатого века, составленного Николой Арсеновичем и опубликованного этнографическим музеем города Сплит, Хорватия, в 2003 году. Рисунок был получен при содействии библиотекаря этнографического музея Сплита, который расположен в той части средневекового центра города, которая восходит еще к временам римского владычества: рядом с руинами дворца императора Диоклетиана, датируемыми примерно 304 годом до н. э. В альбоме собраны тщательно раскрашенные изображения людей из разных районов Хорватии, сопровождаемые описаниями костюмов и деталей повседневного быта.

Сейчас Кистане – небольшой городок в Буковице, одном из географических регионов Хорватии. Он расположен в северной Далмации, области с богатой историей времен римлян и венецианцев. Хорватское слово «mamok» означает «холостяк», «кавалер», «ухажер» – неженатый молодой человек брачного возраста. Изображенный на обложке юноша выглядит очень щеголевато в выглаженной белой полотняной рубашке и расшитом жилете – вероятно, это его лучший наряд, в котором он ходит в церковь и на праздники, или приоделся для свидания с девушкой.

Манера одеваться и общий уклад жизни за прошедшие 200 лет сильно изменились, и различия между областями, когда-то столь разительные, сгладились. Теперь трудно отличить друг от друга даже выходцев с разных континентов, что уж говорить о деревеньках и городках, разделенных несколькими километрами. Мы обменяли культурное разнообразие на иное устройство личной жизни – основанное на многостороннем и стремительном технологическом развитии.

Издательство Manning откликается на новации и инициативы в компьютерной отрасли обложками своих книг, на которых представлено широкое разнообразие местных укладов быта в позапрошлом веке. Мы возвращаем его в том виде, в каком оно запечатлено на иллюстрациях в старых книгах и собраниях, – примером может служить обложка этой книги.

# Часть 1

# Hadoop – каркас распределенного программирования

В части 1 закладываются основы для понимания и использования Hadoop. Мы расскажем, из каких аппаратных компонентов состоит кластер Hadoop, а также опишем процедуру установки и конфигурирования, необходимую для создания работоспособной системы. Мы дадим общее представление о каркасе MapReduce, после чего вы напишете и запустите свою первую MapReduce-программу.

# ГЛАВА 1.
# Введение в Hadoop

***В этой главе:***

- Основы составления масштабируемых, распределенных программ для обработки данных.
- Что представляют собой Hadoop и MapReduce.
- Создание и запуск простой MapReduce-программы.

В современном мире данные окружают нас со всех сторон. Мы загружаем на серверы видео, делаем фотографии с помощью мобильных телефонов, посылаем текстовые сообщения друзьям, изменяем их статус в системе Facebook, оставляем комментарии на разных сайтах, щелкаем по рекламным баннерам и т. д. Компьютеры со своей стороны также порождают и хранят все больше и больше данных. Возможно, даже эта книга представлена у вас в электронном виде, и читаете вы ее на экране своего компьютера. И уж точно факт покупки этой книги отражен в базе данных какого-нибудь розничного продавца[1].

Проблемы, вызванные экспоненциальным ростом данных, первыми ощутили на себе такие находящиеся на переднем крае технологий компании, как Google, Yahoo, Amazon и Microsoft. Им приходилось просеивать терабайты и петабайты данных, чтобы понять, какие вебсайты популярны, какие книги пользуются спросом, какая реклама находит отклик у пользователей. Имеющиеся инструменты оказались не приспособлены к обработке столь больших объемов данных. Google стала первой компанией, начавшей пропагандировать систему *MapReduce*, которую применила для масштабирования обработки данных. Эта система вызвала большой интерес, так как многие другие компании уже столкнулись с аналогичными проблемами, а заново изобретать собственный инструментарий было нерентабельно.

[1] Вы ведь читаете легальную копию, правда?

Дуг Каттинг разглядел открывающиеся возможности и принялся за разработку версии MapReduce с открытым исходным кодом, которую назвал Hadoop. Вскоре Yahoo и другие компании объединили усилия в поддержку этого начинания. Ныне Hadoop составляет основную часть вычислительной инфраструктуры многих работающих в веб компаний, в частности, Yahoo, Facebook, LinkedIn и Twitter. Примериваются к ней и более традиционные организации, например, средства массовой информации и телекоммуникационные компании. В главе 12 описаны примеры использования Hadoop в таких компаниях, как New York Times, China Mobile и IBM.

Hadoop и, более общо, технологии распределенной обработки больших массивов данных, быстро становятся важным умением для широкого круга программистов. Сегодня программисту для эффективной работы необходимо знать о реляционных базах данных, сетях и безопасности, хотя еще пару десятков лет назад все это считалось факультативными навыками. Так и базовые знания о распределенной обработке данных скоро станут непременной частью багажа любого программиста. Ведущие университеты, например Стэнфордский и Карнеги-Меллона, уже ввели в программу факультетов информатики курс по изучению Hadoop. Эта книга поможет вам, программисту-практику, быстро освоить Hadoop и начать использовать его для обработки собственных наборов данных.

Эта глава содержит несколько более формальное введение в каркас Hadoop, рассматриваемый в контексте распределенных систем и систем обработки данных. Мы дадим обзор модели программирования MapReduce. Простой пример программы подсчета слов с помощью имеющихся инструментов поможет уяснить, какие проблемы возникают при обработке больших массивов данных. Затем вы напишете программу, основанную на использовании Hadoop, что позволит лучше оценить простоту этого каркаса. Мы также поговорим об истории Hadoop и некоторых перспективах парадигмы MapReduce. Но сначала позвольте мне вкратце объяснить, для чего я написал эту книгу и чем она может быть вам полезна.

## 1.1. Зачем написана книга «Hadoop в действии»?

На своем опыте я убедился, что поначалу Hadoop завораживает своими возможностями, но дело с трудом продвигается дальше написания элементарных примеров. Документация на официальном сайте

Hadoop довольна полная, но найти ясные ответы на прямо поставленные вопросы не всегда легко.

Решение данной проблемы и составляет задачу этой книги. Я не стану углубляться в скучные технические подробности, а поделюсь информацией, которая позволит вам быстро создавать полезный код. Попутно я расскажу о более сложных вопросах, часто возникающих на практике.

## 1.2. Что такое Hadoop?

Говоря формально, Hadoop – это каркас с открытым исходным кодом, предназначенный для создания и запуска распределенных приложений, обрабатывающих большие объемы данных. Распределенные вычисления – это широкая и многогранная область, но у Hadoop есть ряд важных отличительных особенностей, а именно:

- *Доступность* – Hadoop работает на крупных кластерах, собранных из стандартных компьютеров, или в вычислительном облаке, например на базе службы Elastic Compute Cloud (EC2), предлагаемой компанией Amazon.
- *Надежность* – поскольку Hadoop должен работать на стандартном оборудовании, его архитектура разработана с учетом возможности частых отказов. Большинство отказов можно обработать так, что характеристики кластера будут ухудшаться постепенно.
- *Масштабируемость* – Hadoop масштабируется линейно, то есть при увеличении объема данных достаточно добавить новые узлы в кластер.
- *Простота* – Hadoop позволяет пользователю быстро создавать эффективный параллельный код.

Доступность и простота Hadoop дают ему конкурентное преимущество в деле написания и запуска больших распределенных программ. Даже студент колледжа может быстро собрать собственный недорогой кластер Hadoop. С другой стороны, надежность и масштабируемость делают Hadoop подходящим средством даже для таких ответственных задач, которые решаются в компаниях Yahoo и Facebook. Поэтому Hadoop популярен как в академических кругах, так и в промышленности.

На рис. 1.1 показано, как пользователь взаимодействует с кластером Hadoop. Как видите, кластер Hadoop представляет собой на-

бор стандартных компьютеров, объединенных в сеть, физически расположенную в одном месте[2]. Как хранение, так и обработка данных происходят внутри этого «облака» машин. Разные пользователи отправляют кластеру Hadoop вычислительные «задания» с клиентских машин, например со своих настольных компьютеров, которые могут находиться далеко от кластера.

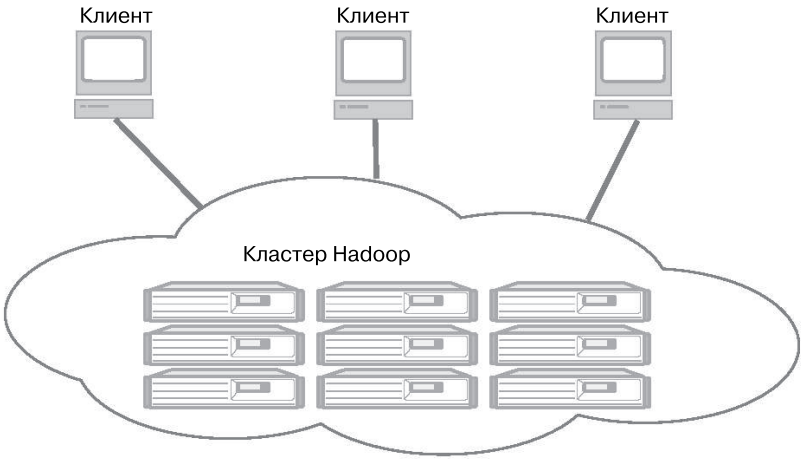


**Рис. 1.1.** Кластер Hadoop состоит из многих машин, которые хранят и параллельно обрабатывают большие наборы данных. Клиентские компьютеры посылают в это вычислительное облако задания и получают результаты.

Не все распределенные системы конфигурируются так, как показано на рис. 1.1. Краткое знакомство с другими распределенными системами поможет лучше понять философию Hadoop.

## 1.3. Сравнение Hadoop с другими распределенными системами

Закон Мура верно служил нам несколько десятилетий, но теперь создание все более и более мощных серверов не обязательно дает оптимальное решение крупномасштабных задач. Быстро набирает

[2] Хотя это и необязательно, обычно кластер Hadoop состоит из более-менее однородных машин на базе процессоров x86 под управлением ОС Linux. И почти всегда они находятся в одном центре обработки данных, часто даже в общем наборе стоек.

популярность альтернативный подход – объединение множества недорогих стандартных машин в функционально единую *распределенную систему*.

Чтобы понять, почему распределенные системы (масштабирование по горизонтали) оказались популярнее монолитных серверов (масштабирование по вертикали), надо принять во внимание производительность и цену современных технологий ввода/вывода. Высокопроизводительному компьютеру с четырьмя каналами ввода/вывода, производительностью 100 МБ/с каждый, потребуется три часа, чтобы только *прочитать* набор данных объемом 4 ТБ! Если же воспользоваться Hadoop, то такой же набор данных можно разбить на меньшие блоки (обычно размером 64 МБ), распределенные по нескольким машинам кластера с помощью распределенной файловой системы Hadoop Distributed File System (HDFS). Благодаря довольно скромной репликации кластерные машины могут читать этот набор данных параллельно, достигая гораздо более высокой пропускной способности. И при этом целый кластер, собранный из стандартных машин, оказывается дешевле одного высокопроизводительного сервера!

Сказанное выше объясняет, почему кластер Hadoop оказывается эффективнее монолитных систем. А теперь сравним Hadoop с другими архитектурами распределенных систем. Одним из примеров может служить система SETI@home, в которой хранители экрана, работающие на компьютерах, разбросанных по всему земному шару, совместно решают задачу поиска внеземных цивилизаций. В этой системе имеется центральный сервер, на котором хранятся полученные из космоса радиосигналы, и этот сервер по Интернету рассылает клиентским компьютерам задачи поиска аномальных признаков. При таком подходе данные перемещаются на места выполнения вычислений (хранители экрана в настольных компьютерах). По завершении вычислений результаты возвращаются в систему хранения данных.

Hadoop отличается от систем, подобных SETI@home, взглядом на данные. SETI@home основана на повторяющихся операциях пересылки данных между клиентами и серверами. Это хорошо работает для задач, требующих большого объема вычислений, но не годится, когда нужно обработать много данных, поскольку их объем настолько велик, что перемещение становится неэффективным. В основу Hadoop положена идея приближения кода к данным, а не наоборот. Взгляните еще раз на рис. 1.1 – как видите, хранение данных и выполнение вычислений происходит внутри кластера Hadoop. Клиенты

только посылают подлежащие исполнению MapReduce-программы, а они сравнительно невелики (часто порядка нескольких килобайтов). И, что еще важнее, идея приближения кода к данным применяется и в самом кластере Hadoop. Данные распределяются по кластеру, и обработка блока данных по возможности производится на той же машине, где этот блок находится.

Философия приближения кода к данным имеет смысл именно для того типа массивной обработки данных, для которого проектировался Hadoop. Размер исполняемых программ («кода») на много порядков меньше размера данных, поэтому перемещать их проще. Кроме того, на перемещение данных по сети ушло бы куда больше времени, чем на применение к ним вычислений. Так что пусть данные остаются там, где находятся, а на хост-машину мы будем пересылать исполняемый код.

Теперь, когда вы понимаете, какое место Hadoop занимает в ряду распределенных систем, посмотрим, как он соотносится с системами обработки данных, под которыми обычно понимаются СУБД на основе SQL.

## 1.4. Сравнение СУБД на основе SQL с Hadoop

Итак, Hadoop – это каркас для обработки данных. Так чем же он лучше стандартных реляционных СУБД, являющихся рабочей лошадкой в современных приложениях для обработки данных? Одна из причин заключается в том, что SQL (*структурированный* язык запросов) по природе своей ориентирован на работу со структурированными данными. А многие приложения Hadoop имеют дело с неструктурированными данными, например текстовыми. С этой точки зрения, Hadoop предлагает более общую парадигму, чем SQL.

Что касается структурированных данных, то тут сравнение оказывается более тонким. В принципе, SQL и Hadoop могут дополнять друг друга, поскольку SQL – это язык запросов, который можно реализовать поверх Hadoop, выступающего в роли подсистемы исполнения[3]. Однако на практике под словами «СУБД на основе SQL» принято понимать целый спектр унаследованных технологий, предлагаемых несколькими доминирующими производителями, причем эти технологии оптимизированы для исторически сложившегося на-

[3] На самом деле, в сообществе Hadoop эта тема активно обсуждается, и некоторые из передовых проектов в этой области мы рассмотрим в гл. 11.

бора приложений. Многие из существующих коммерческих СУБД не соответствуют требованиям, под которые проектировался Hadoop.

Помня об этом, попробуем все же провести более детальное сравнение Hadoop с типичными СУБД на основе SQL по некоторым параметрам.

### Масштабирование по горизонтали, а не по вертикали

Масштабирование коммерческих реляционных баз данных обходится дорого. По своей природе, они более приспособлены для вертикального масштабирования. Чтобы развернуть более крупную базу данных, вы покупаете более мощный сервер. Вообще, стало уже привычным делом, что производители серверов позиционируют свои высокопроизводительные модели как «серверы класса базы данных». К сожалению, неизбежно настает такой момент, когда набор данных оказывается настолько большим, что достаточно мощного сервера для его обслуживания просто не существует. Но еще важнее тот факт, что высокопроизводительные серверы слишком дороги для многих приложений. Например, машина, которая в четыре раза мощнее стандартного ПК, обойдется гораздо дороже, чем объединение четырех ПК в кластер. Hadoop проектировался в расчете на горизонтально масштабируемую архитектуру, построенную на базе кластера стандартных ПК. Увеличение объема ресурсов сводится к добавлению новых машин в кластер Hadoop. Кластеры Hadoop, состоящие из десятков и сотен машин, считаются стандартными. На самом деле, запускать Hadoop на одном сервере имеет смысл только для разработки.

### Пары ключ/значение вместо реляционных таблиц

Основополагающим принципом реляционных СУБД является размещение данных в таблицах, имеющих реляционную структуру, определяемую схемой. Реляционная модель обладает рядом замечательных формальных свойств, но многие современные приложения имеют дело с типами данных, которые плохо в эту модель укладываются. В качестве широко известных примеров упомянем текстовые документы, изображения и XML-файлы. Кроме того, большие наборы данных часто вообще не структурированы или слабо структурированы. В Hadoop в качестве основной единицы данных используется пара ключ/значение, и это достаточно гибкое решение для работы со слабо структурированными типами данных. Исходные данные в Hadoop могут быть представлены в любом формате, но в конечном итоге они преобразуются в пары ключ/значение, к которым и применяются функции обработки.

### Функциональное программирование (MapReduce) вместо декларативных запросов (SQL)

По сути своей SQL является высокоуровневым декларативным языком. Запрашивая данные, вы говорите, какой результат хотели бы получить, и предоставляете СУБД решать, как добиться желаемого. В парадигме MapReduce предполагается, что вы сами описываете конкретные шаги обработки данных, что в какой-то мере напоминает порождаемый СУБД план выполнения SQL-запроса В SQL вы формулируете команды-запросы, в MapReduce пишете скрипты и программы. MapReduce допускает более общие способы обработки данных, чем SQL. Например, на основе данных можно строить сложные статистические модели или изменять формат изображений. SQL для таких задач приспособлен плохо.

С другой стороны, многие считают, что при работе с данными, которые хорошо укладываются в реляционные структуры, применение MapReduce выглядит неестественно. Программистам, привыкшим к парадигме SQL, бывает трудно перестроиться и мыслить так, как требует MapReduce. Надеюсь, что примеры и упражнения, приведенные в этой книге, сделают программирование в рамках MapReduce интуитивно более понятным. Отметим, однако, что существует много расширений, позволяющих получить все выгоды масштабируемости Hadoop, программируя в более привычных парадигмах. Есть даже такие, что позволяют писать запросы на SQL-подобном языке и затем автоматически компилируют их в исполняемый код для MapReduce. С некоторыми инструментами такого рода мы познакомимся в главах 10 и 11.

### Автономная пакетная обработка вместо оперативных транзакций

Hadoop проектировался для автономной обработки и анализа больших объемов данных. Он не предназначен для произвольного считывания и обновления нескольких записей, то есть не может служить заменой системам оперативной обработки транзакций. На самом деле, сейчас и в обозримом будущем Hadoop лучше всего использовать для работы с хранилищами данных, в которых запись производится однократно, а чтение многократно. В этом смысле он напоминает «хранилища данных» в мире SQL.

Мы рассмотрели в общих чертах, как Hadoop соотносится с распределенными системами и с СУБД на основе SQL. Теперь научимся писать для него программы. Для этого необходимо понимать парадигму MapReduce, лежащую в основе Hadoop.

# 1.5. Знакомство с MapReduce

Вы, наверное, знакомы с такими моделями обработки данных, как конвейеры и очереди сообщений. Эти модели ориентированы на вполне определенные виды приложений. Наиболее хорошо известны конвейеры операционной системы Unix. Конвейер позволяет *повторно использовать* примитивы обработки; простое соединение существующих модулей в цепочку порождает новые модули. Очереди сообщений обеспечивают *синхронизацию* примитивов обработки. Программист создает задание для обработки данных в виде примитива, который может играть роль производителя или потребителя. Синхронизацию выполнения примитивов берет на себя система.

MapReduce также представляет собой модель обработки данных. Ее основное достоинство состоит в простоте масштабирования при наличии нескольких вычислительных узлов. В модели MapReduce примитивы обработки данных называются *распределителями* (mapper) и *редукторами* (reducer). Декомпозиция приложения обработки данных на распределители и редукторы иногда оказывается нетривиальной задачей. Но, коль скоро приложение представлено в форме, пригодной для MapReduce, его масштабирование на сотни, тысячи и даже десятки тысяч машин, объединенных в кластер, сводится к простому изменению конфигурации. Именно эта простота масштабирования и привлекла внимание многих программистов к модели MapReduce.

### Есть много способов сказать MapReduce

Хотя о парадигме MapReduce написано немало, само название разные люди пишут по-разному. В оригинальной работе, опубликованной Google, и в статье в википедии фигурирует название *MapReduce*. Однако на некоторых страницах сайта Google встречается также написание *Map Reduce* (например, на странице http://research.google.com/roundtable/MR.html). На сайте официальной документации по Hadoop можно встретить ссылки на *Map-Reduce Tutorial*. Перейдя по такой ссылке, вы окажетесь на странице руководства, озаглавленного *Hadoop Map/Reduce Tutorial* (http://hadoop.apache.org/core/docs/current/mapred_tutorial.html), в котором объясняется, что такое каркас *Map/Reduce*. Различные варианты написания встречаются и для других компонентов Hadoop, например *NameNode* (*name node*, *namenode* и *namenode*), *DataNode*, *JobTracker* и *TaskTracker*. Для единообразия мы в этой книге всюду будем придерживаться ВерблюжьейНотации (то есть писать *MapReduce*, *NameNode*, *DataNode*, *JobTracker* и *TaskTracker*).

### 1.5.1. Масштабирование простой программы вручную

Прежде чем переходить к формальному рассмотрению MapReduce, попробуем масштабировать простенькую программу для обработки большого набора данных. Вы увидите, какие проблемы при этом возникают, и, следовательно, сможете по достоинству оценить преимущества использования каркасов типа MapReduce, которые берут на себя черную работу.

Нашей задачей будет подсчитать, сколько раз каждое слово встречается в наборе документов. В этом упражнении набор будет состоять всего из одного документа, содержащего единственного предложение:

*Do as I say, not as I do.*

(Делай, как я говорю, а не как я делаю.)

Мы должны будем получить счетчики слов, показанные в таблице.

| Слово | Счетчик |
|---|---|
| as | 2 |
| do | 2 |
| I | 2 |
| not | 1 |
| say | 1 |

Назовем это упражнение *подсчетом слов*. Если набор документов мал, то для решения задачи достаточно прямолинейной программы. Напишем ее на псевдокоде:

```
define wordCount as Multiset;
for each document in documentSet {
    T = tokenize(document);
    for each token in T {
        wordCount[token]++;
    }
}
display(wordCount);
```

Эта программа в цикле обходит все документы. Каждый документ разбивается на слова, и для каждого слова на единицу увеличивается соответствующий ему счетчик в мультимножестве `wordCount`. В конце функция `display()` распечатывает все элементы `wordCount`.

**Примечание.** Мультимножеством называется множество, в котором для каждого элемента хранится также счетчик. Создаваемая нами структура данных для подсчета слова – канонический пример мультимножества. На практике оно обычно реализуется в виде хеш-таблицы.

Эта программа прекрасно работает, пока интересующий нас набор документов не слишком велик. Но допустим, что нужно построить

фильтр спама, для чего требуется знать, какие слова часто встречаются в миллионах собранных вами «мусорных» почтовых сообщений. Перебор всех этих документов на одном компьютере займет чрезвычайно много времени. Сократить время можно, переписав программу так, чтобы она распределяла работу между несколькими машинами. Каждая машина будет обрабатывать небольшую часть всего набора документов. Когда все документы будут обработаны, наступит черед второй фазы – объединения полученных на каждой машине результатов. Псевдокод первой фазы в случае распределенной обработки выглядит так:

```
define wordCount as Multiset;
for each document in documentSubset {
    T = tokenize(document);
    for each token in T {
        wordCount[token]++;
    }
}
sendToSecondPhase(wordCount);
```

А псевдокод второй фазы так:

```
define totalWordCount as Multiset;
for each wordCount received from firstPhase {
    multisetAdd (totalWordCount, wordCount);
}
```

Не сложно, правда? Но есть несколько нюансов, из-за которых эта программа может работать не так, как ожидается. Прежде всего, мы не подумали о требовании к производительности чтения документов. Если все документы хранятся на одном центральном сервере, то узким местом станет пропускная способность этого сервера. Выделение нескольких машин для обработки поможет, но только до определенного предела – пока сервер хранения справляется с нагрузкой. Необходимо еще распределить сами документы между машинами, так чтобы каждая машина обрабатывала только хранящиеся на ней документы. Вот тогда мы сможем устранить узкое место, связанное с наличием центрального сервера хранения. Это еще раз подтверждает сделанное ранее замечание о тесной связи между хранением и обработкой в распределенных приложениях для обработки больших массивов данных.

Еще один недостаток приведенной выше программы заключается в том, что структуры `wordCount` (и `totalWordCount`) хранятся в памяти. Если набор документов велик, то количество уникальных

слов может превысить объем доступной оперативной памяти. В английском языке около миллиона слов, и они вообще-то могут легко уместиться в память iPod, но нашей программе придется иметь дело со словами, которых нет в стандартных словарях английского языка. Например, могут встречаться названия типа *Hadoop*. Мы должны будем подсчитывать и слова, написанные с грамматическими ошибками (например, *exampel*), и по отдельности считать различные формы слова (например, *eat*, *ate*, *eaten* и *eating*). И даже если для хранения уникальных слов в наборе документов памяти хватает, небольшое изменение в постановке задачи может привести к резкому увеличению потребности в памяти. Например, что если вместо слов мы захотим подсчитывать IP-адреса, встречающиеся в файле журнале, или частоту биграмм. В последнем случае нам придется работать с мультимножеством, содержащим миллиарды записей, а это уже превышает объем памяти большинства стандартных компьютеров.

**Примечание**. Биграммой называется пара соседних слов. В предложении «Do as I say, not as I do» можно выделить следующие биграммы: *Do as*, *as I*, *I say*, *say not*, *not as*, *as I*, *I do*. Аналогично триграммами называются группы из трех соседних слов. Биграммы и триграммы находят важные применения в обработке естественных языков.

Структура `wordCount` может не поместиться в память; нам придется переписать программу так, чтобы эта хеш-таблица хранилась на диске. А для реализации дисковой хеш-таблицы нужно будет написать достаточно сложный и объемный код.

Вспомним еще, что для второй фазы предусмотрена только одна машина, которой предстоит обработать структуры `wordCount`, передаваемые *всеми* машинами, участвующими в первой фазе. Сама по себе обработка одной структуры `wordCount` довольно громоздкая. После выделения достаточно большого количества машин для первой фазы наличие единственной машины для второй фазы становится узким местом. Возникает очевидный вопрос: можно ли переписать вторую фазу, так чтобы она тоже была распределенной и допускала масштабирование за счет добавления новых машин?

Ответ положительный. Чтобы сделать вторую фазу распределенной, необходимо как-то разделить нагрузку между несколькими машинами, работающими независимо. Нужно будет *разбить* таблицу `wordCount`, получившуюся после завершения первой фазы, на части, так чтобы каждая машина, принимающая участие во второй фазе, обрабатывала только одну часть. Предположим, что в нашем примере для