

Надоор

В ДЕЙСТВИИ

Чак Лэм



MANNING



УДК 004.6:004.42Apache Hadoop

ББК 32.973.26-018.2

Л92

Л92 Чак Лэм

Hadoop в действии. – М.: ДМК Пресс, 2012. – 424с.: ил.

ISBN 978-5-94074-785-7

Обработка больших массивов данных с помощью традиционных СУБД может оказаться трудным делом. Apache Hadoop — это каркас для разработки приложений, предназначенных для выполнения в распределенном кластере, без применения SQL. Такие приложения прекрасно масштабируются и могут обрабатывать гигантские массивы данных. Если вам требуется произвести анализ данных, то Hadoop — как раз то, что надо.

Прочитав эту книгу, вы познакомитесь с предметом и научитесь писать программы в стиле MapReduce. После нескольких простых примеров автор быстро переходит к вопросу об использовании Hadoop для решения более сложных задач анализа данных. Описываются рекомендованные приемы и паттерны проектирования, полезные при программировании для MapReduce.

Для чтения книги требуется знание основ языка Java. Некоторое знакомство с математической статистикой поможет разобраться в более сложных примерах.

УДК 004.6:004.42Apache Hadoop

ББК 32.973.26-018.2

Original English language edition published by Manning Publications 178 South Hill Drive, Westampton NJ 08060 USA, USA. Copyright (c) 2011 by Manning Publications. Russian-language edition copyright (c) 2012 by DMK Press. All rights reserved.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но, поскольку вероятность технических ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 978-1-93518-219-1 (англ.)

© 2011 by Manning Publications Co. All rights reserved.

ISBN 978-5-94074-785-7

© Оформление, перевод на русский язык ДМК Пресс, 2012



ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ	10
БЛАГОДАРНОСТИ	12
ОБ ЭТОЙ КНИГЕ	14
Структура книги.....	15
Графическое выделение и загрузка исходного кода	15
АВТОР В СЕТИ	16
ОБ АВТОРЕ	17
ОБ ИЛЛЮСТРАЦИИ НА ОБЛОЖКЕ	18
ЧАСТЬ 1.	
Hadoop – каркас распределенного программирования.....	19
ГЛАВА 1. Введение в Hadoop	21
1.1. Зачем написана книга «Hadoop в действии»?	22
1.2. Что такое Hadoop?	23
1.3. Сравнение Hadoop с другими распределенными системами.....	24
1.4. Сравнение СУБД на основе SQL с Hadoop.....	26
1.5. Знакомство с MapReduce.....	29
1.5.1. Масштабирование простой программы вручную	30
1.5.2. Масштабирование той же программы с помощью MapReduce	33
1.6. Подсчет слов с помощью Hadoop – ваша первая программа.....	36
1.7. История Hadoop.....	43
1.8. Резюме	44
1.9. Ресурсы	45

ГЛАВА 2. Запуск Hadoop	46
2.1. Структурные элементы Hadoop	46
2.1.1. NameNode	47
2.1.2. DataNode	47
2.1.3. Secondary NameNode	49
2.1.4. JobTracker	49
2.1.5. TaskTracker	50
2.2. Настройка SSH для кластера Hadoop	52
2.2.1. Определение общей учетной записи	52
2.2.2. Проверка правильности установки SSH	52
2.2.3. Генерация пары ключей	53
2.2.4. Распространение открытого ключа и проверка возможности входа в систему	53
2.3. Запуск Hadoop	54
2.3.1. Локальный (автономный) режим	55
2.3.2. Псевдораспределенный режим	56
2.3.3. Полностью распределенный режим	58
2.4. Веб-интерфейс для мониторинга кластера	62
2.5. Резюме	63
ГЛАВА 3. Компоненты Hadoop	65
3.1. Работа с файлами в системе HDFS	65
3.1.1. Основные команды для работы с файлами	66
3.1.2. Чтение и запись в HDFS из программы	71
3.2. Анатомия MapReduce-программы	74
3.2.1. Типы данных в Hadoop	76
3.2.2. Распределитель	78
3.2.3. Редуктор	79
3.2.4. Разбивка — направление выхода распределителя	80
3.2.5. Комбинатор — локальная редукция	81
3.2.6. Подсчет слов с помощью готовых классов распределителя и редуктора	81
3.3. Чтение и запись	83
3.3.1. Интерфейс InputFormat	85
3.3.2. Интерфейс OutputFormat	91

3.4. Резюме	93
-------------------	----

ЧАСТЬ 2.

Hadoop в действии 95

Глава 4. Создание простых MapReduce-программ . 97

4.1. Получение набора данных о патентах.....	98
4.1.1. Данные о цитировании патентов	99
4.1.2. Данные об описаниях патентов	101
4.2. Определение шаблона MapReduce-программы	102
4.3. Подсчет всякой всячины	108
4.4. Адаптация к изменениям в API Hadoop	114
4.5. Интерфейс Hadoop Streaming	118
4.5.1. Интерфейс Streaming и команды Unix	119
4.5.2. Streaming и скрипты	120
4.5.3. Интерфейс Streaming и пары ключ/значение	126
4.5.4. Интерфейс Streaming и пакет Aggregate	131
4.6. Повышение производительности с помощью комбинаторов.....	137
4.7. Упражнения	142
4.8. Резюме	144
4.9. Дополнительные ресурсы	145

ГЛАВА 5. Углубленное изучение MapReduce..... 147

5.1. Сцепление задач MapReduce	148
5.1.1. Последовательное сцепление задач MapReduce.....	148
5.1.2. Сцепление задач MapReduce со сложными зависимостями.....	148
5.1.3. Включение в цепочку шагов пред- и постобработки	149
5.2. Соединение данных из разных источников.....	154
5.2.1. Соединение на стороне редуктора	155
5.2.2. Построение реплицированных соединений с помощью класса DistributedCache	166
5.2.3. Полусоединение: соединение на стороне редуктора с фильтрацией на стороне распределителя	171
5.3. Создание фильтра Блума	173

5.3.1. Что делает фильтр Блума?.....	173
5.3.2. Реализация фильтра Блума	176
5.3.3. Фильтр Блума в Hadoop версии 0.20+	184
5.4. Упражнения	184
5.5. Резюме	187
5.6. Дополнительные ресурсы	187
ГЛАВА 6. Практическое программирование	189
6.1. Разработка MapReduce-программ	190
6.1.1. Локальный режим.....	191
6.1.2. Псевдораспределенный режим.....	197
6.2. Мониторинг и отладка в производственном кластере	203
6.2.1. Счетчики	203
6.2.2. Пропуск плохих записей	205
6.2.3. Перезапуск сбойных заданий с помощью IsolationRunner	210
6.3. Оптимизация производительности	211
6.3.1. Уменьшение сетевого трафика с помощью комбинатора.....	212
6.3.2. Уменьшение объема выходных данных.....	212
6.3.3. Использование сжатия.....	213
6.3.4. Повторное использование JVM	216
6.3.5. Наблюдаемое исполнение.....	217
6.3.6. Переработка кода и модификация алгоритмов.....	219
6.4. Резюме	221
ГЛАВА 7. Сборник рецептов	222
7.1. Передача нестандартных параметров задаче	222
7.2. Получение информации о конкретном задании.....	226
7.3. Разбиение на несколько выходных файлов	227
7.4. Ввод и вывод в базу данных	234
7.5. Сортировка выходных данных	236
7.6. Резюме	238
ГЛАВА 8. Администрирование Hadoop	239
8.1. Практическая настройка параметров	240

8.2. Проверка состояния системы	243
8.3. Установка прав доступа	245
8.4. Управление квотами	246
8.5. Включение корзины	247
8.6. Удаление узлов DataNode.....	247
8.7. Добавление узлов DataNode	249
8.8. Управление узлами NameNode и Secondary NameNode...	250
8.9. Восстановление после сбоя узла NameNode.....	252
8.10. Проектирование топологии сети и осведомленность о стойках	254
8.11. Планирование задач, поступающих от нескольких пользователей.....	257
8.11.1. Организация нескольких узлов JobTracker.....	257
8.11.2. Справедливый планировщик	258
8.12. Резюме	261

ЧАСТЬ 3.

Hadoop в реальной жизни 263

ГЛАВА 9. Эксплуатация Hadoop в облаке 265

9.1. Введение в Amazon Web Services.....	266
9.2. Настройка AWS	267
9.2.1. Получение учетных данных для аутентификации в AWS.....	268
9.2.2. Получение командных утилит	271
9.2.3. Подготовка пары ключей для работы с SSH	273
9.3. Настройка Hadoop в EC2	275
9.3.1. Задание параметров защиты	275
9.3.2. Конфигурирование типа кластера	276
9.4. Запуск MapReduce-программ в среде EC2.....	278
9.4.1. Перенос своего кода в кластер Hadoop	279
9.4.2. Доступ к данным из кластера Hadoop	279
9.5. Очистка и останов экземпляров EC2	285
9.6. Amazon Elastic MapReduce и другие службы AWS.....	285
9.6.1. Amazon Elastic MapReduce.....	286

9.6.2. AWS Import/Export.....	287
9.7. Резюме	288
ГЛАВА 10. Программирование с помощью Pig	289
10.1. Научитесь думать по-свински.....	290
10.1.1. Язык описания потоков данных.....	290
10.1.2. Типы данных	291
10.1.3. Определенные пользователем функции	291
10.2. Установка Pig	291
10.3. Запуск Pig	293
10.3.1. Управление оболочкой Grunt	294
10.4. Изучение языка Pig Latin с помощью Grunt	295
10.5. Учимся говорить на Pig Latin.....	302
10.5.1. Типы данных и схемы.....	302
10.5.2. Выражения и функции	304
10.5.3. Реляционные операторы	307
10.5.4. Оптимизация исполнения.....	317
10.6. Определяемые пользователем функции	317
10.6.1. Использование UDF.....	318
10.6.2. Создание UDF	319
10.7. Работа со скриптами.....	322
10.7.1. Комментарии	322
10.7.2. Подстановка параметров	323
10.7.3. Режим многозапросного исполнения	324
10.8. Pig в действии: отыскание похожих патентов.....	326
10.9. Резюме	332
ГЛАВА 11. Hive и другие	333
11.1. Hive	334
11.1.1. Установка и настройка Hive.....	335
11.1.2. Примеры запросов	338
11.1.3. Детали языка HiveQL	342
11.1.4. Hive: подводя итоги	352
11.2. Другие проекты, связанные с Hadoop.....	353
11.2.1. HBase	353

11.2.2. ZooKeeper	353
11.2.3. Cascading	354
11.2.4. Cloudera	354
11.2.5. Katta	355
11.2.6. CloudBase.....	355
11.2.7. Aster Data и Greenplum.....	356
11.2.8. Hama и Mahout	356
11.2.9. search-hadoop.com.....	356
11.3. Резюме	357
ГЛАВА 12. Примеры применения	358
12.1. Преобразование 11 миллионов изображений из архива газеты New York Times	358
12.2. Добыча данных в компании China Mobile	360
12.3. Рекомендование лучших веб-сайтов на StumbleUpon....	367
12.3.1. Как мы пришли к распределенной обработке в StumbleUpon	368
12.3.2. HBase и StumbleUpon	369
12.3.3. Другие применения Hadoop на сайте StumbleUpon.....	379
12.4. Построение аналитической системы для внутрикорпоративного поиска – проект IBM ES2	381
12.4.1. Архитектура ES2	386
12.4.2. Робот ES2.....	387
12.4.3. Аналитические средства в ES2	390
12.4.4. Выводы	400
12.4.5. Библиография.....	401
ПРИЛОЖЕНИЕ. Команды HDFS.....	403
ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ	408



Часть 1

Нadoop – 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 продавца¹.

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 веб-сайты 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 неременной 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 довольно 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 месте². Как хранение, так и обработка данных происходят внутри этого «облака» машин. Разные пользователи управляют кластеру Hadoop вычислительные «задания» с клиентских машин, например со своих настольных компьютеров, которые могут находиться далеко от кластера.

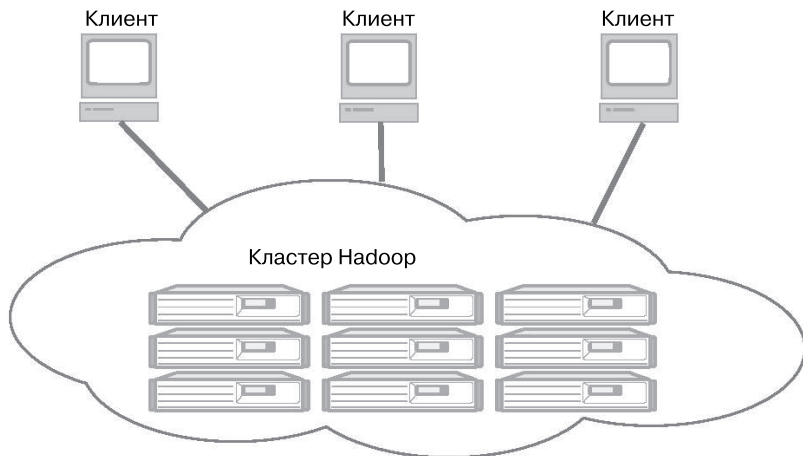


Рис. 1.1. Кластер Hadoop состоит из многих машин, которые хранят и параллельно обрабатывают большие наборы данных. Клиентские компьютеры посылают в это вычислительное облако задания и получают результаты.

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. Высокпроизводительному 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 исполнения³. 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 Верблюжьей Нотации (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.

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);
```

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

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