Изучаем Spark

Молниеносный анализ данных

Холден Карау, Энди Конвински,
Патрик Венделл, Матей Захария

ДМК ПРЕСС издательство

O'REILLY®

УДК 004.65:004.43 Spark
ББК 32.972.34
К21

Карау Х., Конвински Э., Венделл П., Захария М.
К21 Изучаем Spark: молниеносный анализ данных. – М.: ДМК Пресс, 2015. – 304 с.: ил.

ISBN 978-5-97060-323-9

Объем обрабатываемых данных во всех областях человеческой деятельности продолжает расти быстрыми темпами. Существуют ли эффективные приемы работы с ним? В этой книге рассказывается об Apache Spark, открытой системе кластерных вычислений, которая позволяет быстро создавать высокопроизводительные программы анализа данных. С помощью Spark вы сможете манипулировать огромными объемами данных посредством простого API на Python, Java и Scala.

Написанная разработчиками Spark, эта книга поможет исследователям данных и программистам быстро включиться в работу. Она рассказывает, как организовать параллельное выполнение заданий всего несколькими строчками кода, и охватывает примеры от простых пакетных приложений до программ, осуществляющих обработку потоковых данных и использующих алгоритмы машинного обучения.

УДК 004.65:004.43 Spark
ББК 32.972.34

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

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 978-1-449-35862-4 (анг.)
ISBN 978-5-97060-323-9 (рус.)

Copyright © 2015 Databricks
© Оформление, издание, ДМК Пресс, 2015

Содержание

Глава 1

Введение в анализ данных с помощью Spark

В этой главе приводится обобщенный обзор Apache Spark. Если вы уже знакомы с этим фреймворком и его компонентами, можете сразу перейти к главе 2.

Что такое Apache Spark?

Apache Spark – это *универсальная* и *высокопроизводительная* кластерная вычислительная платформа.

По производительности Spark превосходит популярные реализации модели MapReduce, попутно обеспечивая поддержку более широкого диапазона типов вычислений, включая интерактивные запросы и потоковую обработку (streaming processing). Скорость играет важную роль при обработке больших объемов данных, так как именно скорость позволяет работать в интерактивном режиме, не тратя минуты или часы на ожидание. Одно из важнейших достоинств Spark, обеспечивающих столь высокую скорость, – способность выполнять вычисления в памяти. Но даже при работе с дисковой памятью Spark выполняет операции намного эффективнее, чем известные механизмы MapReduce.

Фреймворк создавался с целью охватить как можно более широкий диапазон рабочих нагрузок, которые прежде требовали создания отдельных распределенных систем, включая приложения пакетной обработки, циклические алгоритмы, интерактивные запросы и потоковую обработку. Поддерживая все эти виды задач с помощью единого механизма, Spark упрощает и удешевляет *объединение* разных видов обработки, которые часто необходимо выполнять в едином конвейере обработки данных. Кроме того, он уменьшает бремя обслуживания, поддерживая отдельные инструменты.

Фреймворк Spark предлагает простой API на языках Python, Java, Scala и SQL и богатую коллекцию встроенных библиотек. Он также легко объединяется с другими инструментами обработки больших данных. В частности, Spark может выполняться под управлением кластеров Hadoop и использовать любые источники данных Hadoop, включая Cassandra.

Унифицированный стек

Проект Spark включает множество тесно связанных компонентов. Ядро фреймворка образует его «вычислительный механизм» (computational engine), отвечающий за планирование, распределение и мониторинг приложений, выполняющих множество вычислительных задач на множестве машин – *вычислительном кластере*. Быстрое и универсальное вычислительное ядро Spark приводит в действие разнообразные высокоуровневые компоненты, специализированные для решения разных задач, таких как выполнение запросов SQL или машинное обучение. Эти компоненты поддерживают тесную интеграцию друг с другом, давая возможность объединять их, подобно библиотекам в программном проекте.

Философия тесной интеграции имеет несколько преимуществ. Во-первых, все библиотеки и высокоуровневые компоненты стека извлекают определенные выгоды от улучшений в слоях более низкого уровня. Например, когда в ядро Spark вносятся какие-то оптимизации, библиотеки поддержки SQL и машинного обучения автоматически увеличивают производительность. Во-вторых, затраты на сопровождение стека минимальны, потому что вместо 5–10 независимых программных систем организации требуется поддерживать только одну. Эти затраты включают развертывание, сопровождение, тестирование, поддержку и т. д. Это также означает, что при добавлении в стек Spark новых компонентов все организации, использующие Spark, немедленно получают возможность опробовать эти новые компоненты. Это уменьшает затраты на опробование новых видов анализа данных, избавляя организации от необходимости загружать, развертывать и изучать новые программные проекты.

Наконец, одним из самых больших преимуществ тесной интеграции является возможность создавать приложения, прозрачно объединяющие разные модели обработки. Например, используя Spark, можно написать приложение, применяющее модель машинного обучения для классификации данных в масштабе реального времени и потребляю-

щее потоковые данные. Одновременно аналитики имеют возможность запрашивать результаты, также в масштабе реального времени, посредством SQL (например, объединяя данные с неструктурированными файлами журналов). Кроме того, опытные программисты и исследователи могут обращаться к тем же данным посредством командной оболочки на языке Python и выполнять дополнительные виды анализа. Другие могут пользоваться результатами, получаемыми от автономных приложений пакетной обработки. При этом отделу ИТ приходится обслуживать единственную систему.

Ниже мы коротко представим все основные компоненты Spark, изображенные на рис. 1.1.

Рис. 1.1 ❖ Стек Spark

Spark Core

Spark Core реализует основные функциональные возможности фреймворка Spark, включая компоненты, осуществляющие планирование заданий, управление памятью, обработку ошибок, взаимодействие с системами хранения данных и многие другие. Spark Core является также основой API *устойчивых распределенных наборов данных* (*Resilient Distributed Datasets*, *RDD*) – базовой абстракции Spark. Наборы данных RDD представляют собой коллекции элементов, распределенных между множеством вычислительных узлов, которые могут обрабатываться параллельно. Spark Core предоставляет множество функций управления такими коллекциями.

Spark SQL

Spark SQL – пакет для работы со структурированными данными. Позволяет извлекать данные с помощью инструкций на языке SQL и его диалекте Hive Query Language (HQL). Поддерживает множество ис-

точников данных, включая таблицы Hive, Parquet и JSON. В дополнение к интерфейсу SQL компонент Spark SQL позволяет смешивать в одном приложении запросы SQL с программными конструкциями на Python, Java и Scala, поддерживаемыми абстракцией RDD, и таким способом комбинировать SQL со сложной аналитикой. Подобная тесная интеграция с богатыми возможностями вычислительной среды выгодно отличает Spark SQL от любых других инструментов управления данными. Spark SQL был добавлен в стек Spark в версии 1.0.

Первой реализацией поддержки SQL в Spark стал проект Shark, созданный в Калифорнийском университете, Беркли. Этот проект представлял собой модификацию Apache Hive, способную выполняться под управлением Spark. Позднее ему на смену пришел компонент Spark SQL, имеющий более тесную интеграцию с механизмом Spark и API для разных языков программирования.

Spark Streaming

Spark Streaming – компонент Spark для обработки потоковых данных. Примерами источников таких данных могут служить файлы журналов, заполняемые действующими веб-серверами, или очереди сообщений, посылаемых пользователями веб-служб. Spark Streaming имеет API для управления потоками данных, который близко соответствует модели RDD, поддерживаемой компонентом Spark Core, что облегчает изучение самого проекта и разных приложений обработки данных, хранящихся в памяти, на диске или поступающих в режиме реального времени. Прикладной интерфейс (API) компонента Spark Streaming разрабатывался с прицелом обеспечить такую же надежность, пропускную способность и масштабируемость, что и Spark Core.

MLlib

В состав Spark входит библиотека MLlib, реализующая механизм машинного обучения (Machine Learning, ML). MLlib поддерживает множество алгоритмов машинного обучения, включая алгоритмы классификации (classification), регрессии (regression), кластеризации (clustering) и совместной фильтрации (collaborative filtering), а также функции тестирования моделей и импортирования данных. Она также предоставляет некоторые низкоуровневые примитивы ML, включая универсальную реализацию алгоритма оптимизации методом градиентного спуска. Все эти методы способны работать в масштабе кластера.

GraphX

GraphX – библиотека для обработки графов (примером графа может служить граф друзей в социальных сетях) и выполнения параллельных вычислений. Подобно компонентам Spark Streaming и Spark SQL, GraphX дополняет Spark RDD API возможностью создания ориентированных графов с произвольными свойствами, присваиваемыми каждой вершине или ребру. Также GraphX поддерживает разнообразные операции управления графами (такие как `subgraph` и `mapVertices`) и библиотеку обобщенных алгоритмов работы с графами (таких как алгоритмы ссылочного ранжирования PageRank и подсчета треугольников).

Диспетчеры кластеров

Внутренняя реализация Spark обеспечивает эффективное масштабирование от одного до многих тысяч вычислительных узлов. Для достижения такой гибкости Spark поддерживает большое многообразие *диспетчеров кластеров* (*cluster managers*), включая Hadoop YARN, Apache Mesos, а также простой диспетчер кластера, входящий в состав Spark, который называется Standalone Scheduler. При установке Spark на чистое множество машин на начальном этапе с успехом можно использовать Standalone Scheduler. При установке Spark на уже имеющийся кластер Hadoop YARN или Mesos можно пользоваться встроенными диспетчерами этих кластеров. Подробнее о разных диспетчерах кластеров и их использовании рассказывается в главе 7.

Кто и с какой целью использует Spark?

Так как Spark относится к категории универсальных фреймворков поддержки вычислений в кластерах, он применяется для реализации широкого круга приложений. Во вступлении мы определили две группы читателей, на которых ориентирована наша книга: специалисты в области анализа данных и инженеры-программисты. Давайте теперь поближе познакомимся с обеими группами и с тем, как они используют Spark. Неудивительно, что специалисты в этих двух группах используют Spark по-разному, но мы можем примерно разбить случаи использования на две основные категории – *исследование данных* и *обработка данных*.

Разумеется, это весьма условное разделение, и многие профессионалы обладают обоими навыками, иногда выступая в роли исследо-

вателей данных, а иногда создавая приложения обработки данных. Тем не менее имеет смысл в отдельности рассмотреть эти две группы и соответствующие им случаи использования.

Исследование данных

Наука о данных (data science) – относительно новая дисциплина, появившаяся несколько лет тому назад и специализирующаяся на анализе данных. Несмотря на отсутствие точного определения, мы будем пользоваться термином *специалист в области анализа данных*, или *исследователь*, для обозначения людей, основной задачей которых являются анализ и моделирование данных. Специалисты в области анализа данных могут иметь опыт использования SQL, статистических методов, прогнозирования (машинного обучения) и программирования, как правило, на Python, Matlab или R. Они также владеют приемами преобразования данных в форматы, в которых они могут быть проанализированы для проникновения в их суть (иногда это называют *выпасом данных* – *data wrangling*).

Исследователи используют свои навыки для анализа данных с целью ответить на определенные вопросы или вскрыть их суть. Часто они прибегают к специализированным методам анализа, для чего используют интерактивные оболочки (вместо создания сложных приложений), позволяющие им получать результаты запросов в кратчайшие сроки. Благодаря быстродействию и простоте API фреймворк Spark прекрасно подходит для этой цели, а его встроенные библиотеки предоставляют множество готовых алгоритмов.

Благодаря большому числу компонентов Spark поддерживает большое многообразие видов анализа данных. Командная оболочка Spark упрощает проведение анализа в интерактивном режиме, с применением Python или Scala. Spark SQL также имеет отдельную интерактивную оболочку SQL, которую можно использовать для исследования данных. Компонент Spark SQL можно также использовать в обычных программах на основе Spark или из командной оболочки Spark. Технологии машинного обучения и анализа данных поддерживаются также библиотеками MLlib. Кроме того, имеется поддержка внешних программ Matlab или на языке R. Spark позволяет исследователям данных заниматься задачами, основанными на обработке огромных объемов данных, которые прежде были недоступны при использовании простых инструментов, таких как R или Pandas.

Иногда, вслед за начальным этапом исследования данных, исследователю необходимо оформить анализ в виде законченного продук-

та, то есть создать надежное приложение, позволяющее выполнять данный анализ и способное стать частью промышленного приложения. Например, начальные исследования данных могли бы привести исследователя к мысли о необходимости создания рекомендательной системы (recommender system), интегрированной в веб-приложение и генерирующей предложения для пользователей. Нередко созданием такого законченного продукта занимается другой человек – инженер-программист.

Обработка данных

Еще один основной случай использования фреймворка Spark можно описать в контексте работы инженера-программиста. В данном случае под инженерами-программистами мы подразумеваем разработчиков программного обеспечения, использующих Spark для создания приложений обработки данных. Обычно эти разработчики знакомы с принципами создания программ, такими как инкапсуляция, дизайн интерфейса и объектно-ориентированное программирование. Они часто имеют специальное образование и используют свои знания и навыки для проектирования и создания программных систем, реализующих бизнес-логику.

Программистам Spark предоставляет простой способ распараллеливания создаваемых ими приложений в рамках кластера и скрывает сложность программирования распределенных систем, сетевых взаимодействий и устойчивости к ошибкам. Система дает им достаточно высокий уровень контроля для организации мониторинга и настройки приложений, а также быстрого создания реализаций типичных задач. Модульная природа API (на основе передачи распределенных коллекций объектов) упрощает создание библиотек многократного использования и их тестирование на локальном компьютере.

Пользователи часто выбирают фреймворк Spark в качестве основы для своих приложений обработки данных, потому что он предоставляет широкое разнообразие функциональных возможностей, простых в изучении и применении, а также благодаря его зрелости и надежности.

Краткая история развития Spark

Spark – это проект с открытым исходным кодом, поддерживаемый многочисленным сообществом разработчиков. Если вы или ваша организация пробует применить Spark впервые, вам может быть ин-

тересно узнать немного об истории этого проекта. Проект Spark начинался в 2009 году как исследовательский, в лаборатории систем быстрой разработки приложений RAD Lab Калифорнийского университета (Беркли), позднее переименованной в AMPLab. Прежде сотрудники лаборатории использовали Hadoop MapReduce и пришли к выводу, что модель MapReduce неэффективна для реализации итеративных и интерактивных вычислительных задач. Поэтому с самого начала фреймворк Spark проектировался с прицелом на достижение максимальной производительности в интерактивном режиме и при выполнении итеративных алгоритмов, что, в свою очередь, повлекло реализацию идей хранения данных в памяти и эффективной обработки ошибок.

Вскоре после начала работы над проектом в 2009 году в академических кругах появились первые статьи о Spark. Уже тогда фреймворк показывал 10–20-кратное превосходство в скорости над MapReduce на некоторых задачах.

В числе первых пользователей Spark были только лаборатории Калифорнийского университета. К их числу, например, относятся исследователи в области машинного обучения из проекта Mobile Millennium – они использовали Spark для мониторинга и прогнозирования пробок на дорогах Сан-Франциско. Однако в очень короткое время Spark стали использовать другие организации, и на сегодняшний день более 50 организаций указывают себя на странице Spark PoweredBy[1] и десятки других заявляют об использовании Spark в списках сообществ, таких как Spark Meetups[2] и Spark Summit[3]. Помимо Калифорнийского университета, в разработке Spark участвуют также Databricks, Yahoo! и Intel.

В 2011 году лаборатория AMPLab приступила к разработке высокоуровневых компонентов для Spark, таких как Shark (Hive on Spark)[4] и Spark Streaming. Эти и другие компоненты иногда называют «Стек анализа данных из Беркли» (Berkeley Data Analytics Stack, BDAS)[5].

Исходный код Spark был открыт в марте 2010 года и в июне 2013-го передан в фонд Apache Software Foundation, где продолжает развиваться и по сей день.

---

[1] http://bit.ly/1yx195p.
[2] http://www.meetup.com/spark-users/.
[3] http://spark-summit.org/.
[4] Позднее проект Shark заменил проект Spark SQL.
[5] https://amplab.cs.berkeley.edu/software/.

Версии Spark

С момента создания проект Spark активно развивается сообществом, которое продолжает разрастаться с каждым выпуском. В создании версии Spark 1.0 участвовало более 100 отдельных разработчиков. Несмотря на рост активности, сообщество продолжает выпускать обновленные версии Spark на регулярной основе. Версия Spark 1.0 вышла в мае 2014-го. Эта книга в основном охватывает версии Spark 1.1.0 и выше, хотя большинство примеров будет работать и с более ранними версиями.

Механизмы хранения данных для Spark

Spark может создавать распределенные наборы данных из любых файлов, хранящихся в распределенной файловой системе Hadoop (HDFS) или в других системах хранения данных, поддерживающих Hadoop API (включая локальную файловую систему, Amazon S3, Cassandra, Hive, HBase и др.). Важно помнить, что Spark не требует наличия Hadoop; он просто поддерживает взаимодействие с системами хранения данных, реализующих Hadoop API. Spark поддерживает текстовые файлы, файлы SequenceFile, Avro, Parquet и любые другие форматы, поддерживаемые Hadoop. Приемы использования этих источников данных будут рассматриваться в главе 5.

Глава 2

Загрузка и настройка Spark

В этой главе мы рассмотрим процесс загрузки и настройки Spark для работы в локальном режиме на единственном компьютере. Эта глава написана для всех, кто только приступает к изучению Spark, включая исследователей данных и инженеров.

Функциональные возможности Spark можно использовать в программах на Python, Java или Scala. Для успешного усвоения сведений из этой книги необязательно быть опытным программистом, но все же знание базового синтаксиса хотя бы одного из этих языков будет как нельзя кстати. Мы будем включать примеры на всех этих языках, где только возможно.

Сам фреймворк Spark написан на Scala и выполняется под управлением виртуальной машины Java (Java Virtual Machine, JVM). Чтобы запустить Spark на ноутбуке или в кластере, достаточно лишь установить Java версии 6 или выше. Если вы предпочитаете Python API, вам потребуется интерпретатор Python (версии 2.6 или выше). В настоящее время Spark не поддерживает Python 3.

Загрузка Spark

Первым шагом к использованию Spark являются его загрузка и распаковка. Давайте начнем с загрузки последней скомпилированной версии Spark. Откройте в браузере страницу http://spark.apache.org/downloads.html, выберите тип пакета Pre-built for Hadoop 2.4 and later (Скомпилированная версия с поддержкой Hadoop 2.4) и тип загрузки Direct Download (Непосредственная загрузка). Затем щелкните на ссылке ниже, чтобы загрузить сжатый TAR-файл, или *тарболл*, с именем *spark-1.2.0-bin-hadoop2.4.tgz*.

Пользователи Windows могут столкнуться с проблемой при установке Spark в каталог, имя которого содержит пробелы. Поэтому устанавливайте Spark в каталог с именем без пробелов (например, C:\spark).

Для установки Spark необязательно иметь Hadoop, но если у вас уже имеется настроенный кластер Hadoop или установлена поддержка HDFS, выбирайте для загрузки соответствующую версию. На странице http://spark.apache.org/downloads.html можно также выбрать другой тип пакета, но имена файлов будут отличаться незначительно. Возможна также сборка из исходных текстов; последнюю версию исходных текстов можно получить из репозитория GitHub или выбрав тип пакета Source Code (Исходный код) на странице загрузки.

Большинство версий Unix и Linux, включая Mac OS X, уже имеют предустановленный инструмент командной строки `tar` для распаковки TAR-файлов. Если в вашей операционной системе отсутствует команда `tar`, попробуйте найти в Интернете свободный инструмент распаковки TAR-архивов, например 7-Zip для Windows.

Загрузив файл архива с фреймворком Spark, давайте распакуем его и посмотрим, что входит в состав дистрибутива по умолчанию. Для этого откройте окно терминала, перейдите в каталог, куда была выполнена загрузка Spark, и распакуйте файл. В результате будет создан новый каталог с тем же именем, но без расширения *.tgz*. Перейдите в этот каталог и посмотрите, что в нем содержится. Для этого можно выполнить следующие команды:

```
cd ~
tar -xf spark-1.2.0-bin-hadoop2.4.tgz
cd spark-1.2.0-bin-hadoop2.4
ls
```

Флаг `x` в команде `tar` сообщает архиватору, что он должен извлечь файлы из архива, а флаг `f` определяет имя тарболла. Команда `ls` выводит содержимое каталога. Рассмотрим назначение некоторых наиболее важных файлов и каталогов:

- *README.md* – содержит краткие инструкции по настройке Spark;
- *bin* – содержит выполняемые файлы, которые можно использовать для взаимодействий с фреймворком Spark (среди них, например, командная оболочка Spark, о которой рассказывается далее в этой главе);
- *core*, *streaming*, *python*, ... – содержат исходный код основных компонентов проекта Spark;
- *examples* – содержит примеры реализации некоторых распространенных задач, которые можно опробовать и использовать для изучения Spark API.

Пусть вас не волнует большое число файлов и каталогов в проекте Spark; мы познакомимся с большинством из них далее в этой книге. А теперь давайте сразу же попробуем воспользоваться командными оболочками Spark для Python и Scala. Для начала попытаемся запустить некоторые примеры, входящие в состав дистрибутива. Затем напишем, скомпилируем и выполним собственную простенькую задачу.

Все операции в этой главе будут выполняться в Spark, действующем в *локальном режиме*, то есть в нераспределенном режиме – на единственном компьютере. Spark может работать в нескольких разных режимах, или окружениях. Помимо локального режима, Spark может также выполняться под управлением Mesos, YARN и собственного автономного планировщика Standalone Scheduler. Подробнее о разных режимах выполнения рассказывается в главе 7.

Введение в командные оболочки Spark для Python и Scala

В состав дистрибутива Spark входят интерактивные командные оболочки (shell), позволяющие выполнять специализированные виды анализа. Командные оболочки Spark напоминают любые другие командные оболочки, такие как, например, командные оболочки R, Python и Scala или даже командные оболочки операционных систем, допустим Bash или «Командная строка» в Windows.

Однако, в отличие от большинства других оболочек, позволяющих манипулировать данными на диске и в памяти единственного компьютера, оболочки Spark дают возможность оперировать данными, распределенными по нескольким компьютерам, при этом все сложности, связанные с распределенным доступом, берет на себя Spark.

Так как Spark может загружать данные в память на множестве рабочих узлов, многие распределенные вычисления, даже на массивах данных, занимающих терабайты, распределенных между десятками компьютеров, выполняются всего несколько секунд. Это делает командную оболочку Spark вполне пригодной для исследования данных в интерактивном режиме. Spark предоставляет оболочки для обоих языков, Python и Scala, которые с успехом могут использоваться в кластерах.

Большинство примеров в этой книге написаны на всех языках, поддерживаемых фреймворком Spark, но интерактивные командные оболочки доступны только для Python и Scala. Так как оболочку очень удобно

использовать для изучения API, мы рекомендуем изучить один из этих двух языков, даже если вы занимаетесь разработкой на Java, потому что для всех языков поддерживаются похожие API.

Проще всего продемонстрировать мощь командной оболочки Spark на примере выполнения одного из простых видов анализа. Давайте рассмотрим пример из начального руководства «Quick Start Guide»[1] в официальной документации к Spark.

Сначала запустите одну из командных оболочек Spark. Чтобы запустить командную оболочку для Python, которую также называют PySpark Shell, перейдите в каталог установки Spark и выполните команду:

```
bin/pyspark
```

(Или `bin\pyspark` в Windows.) Чтобы запустить командную оболочку для Scala, выполните:

```
bin/spark-shell
```

В течение нескольких секунд в окне терминала должно появиться приглашение к вводу. Когда оболочка запустится, вы должны увидеть множество сообщений. Вам может понадобиться нажать клавишу Enter, чтобы очистить окно и вывести приглашение к вводу. На рис. 2.1 показано, как выглядит приглашение к вводу в оболочке PySpark Shell.

Кому-то такие начальные сообщения могут показаться излишними или даже раздражающими. Вы можете избавиться от них, создав в каталоге *conf* файл с именем *log4j.properties*. Разработчики Spark уже включили в дистрибутив шаблон этого файла с именем *log4j.properties.template*. Чтобы уменьшить число выводимых сообщений, скопируйте содержимое файла *conf/log4j.properties.template* в файл *conf/log4j.properties* и найдите в нем следующую строку:

```
log4j.rootCategory=INFO, console
```

Уменьшите уровень подробности так, чтобы выводились только сообщения с уровнем `WARN`:

```
log4j.rootCategory=WARN, console
```

Если после этого вновь запустить командную оболочку, вы увидите, что число сообщений уменьшилось, как показано на рис. 2.2.

---

[1] http://spark.apache.org/docs/latest/quick-start.html.