cloudera

Developer Training for Apache Spark and Hadoop

201709c

Introduction

Chapter 1

Course Chapters

- **Introduction**
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Trademark Information

- **The names and logos of Apache products mentioned in Cloudera training courses, including those listed below, are trademarks of the Apache Software Foundation**

Apache Accumulo	Apache Lucene
Apache Avro	Apache Mahout
Apache Bigtop	Apache Oozie
Apache Crunch	Apache Parquet
Apache Flume	Apache Pig
Apache Hadoop	Apache Sentry
Apache HBase	Apache Solr
Apache HCatalog	Apache Spark
Apache Hive	Apache Sqoop
Apache Impala (incubating)	Apache Tika
Apache Kafka	Apache Whirr
Apache Kudu	Apache ZooKeeper

- **All other product names, logos, and brands cited herein are the property of their respective owners**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Introduction

- **About This Course**
- About Cloudera
- Course Logistics
- Introductions
- Hands-On Exercise: Starting the Exercise Environment

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Course Objectives

During this course, you will learn

- **How the Apache Hadoop ecosystem fits in with the data processing lifecycle**
- **How data is distributed, stored, and processed in a Hadoop cluster**
- **How to write, configure, and deploy Apache Spark applications on a Hadoop cluster**
- **How to use the Spark shell and Spark applications to explore, process, and analyze distributed data**
- **How to query data using Spark SQL, DataFrames, and Datasets**
- **How to use Spark Streaming to process a live data stream**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Introduction

- About This Course
- **About Cloudera**
- Course Logistics
- Introductions
- Hands-On Exercise: Starting the Exercise Environment

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

About Cloudera

- The leader in Apache Hadoop-based software and services
- Founded by Hadoop experts from Facebook, Yahoo, Google, and Oracle
- Provides support, consulting, training, and certification for Hadoop users
- Staff includes committers to virtually all Hadoop projects
- Many authors of authoritative books on Apache Hadoop projects

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

CDH

CDH (Cloudera's Distribution including Apache Hadoop)

- 100% open source, enterprise-ready distribution of Hadoop and related projects
- The most complete, tested, and widely deployed distribution of Hadoop
- Integrates all the key Hadoop ecosystem projects
- Available as RPMs and Ubuntu, Debian, or SuSE packages, or as a tarball

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Cloudera Express

- **Cloudera Express**
  - Completely free to download and use
- **The best way to get started with Hadoop**
- **Includes CDH**
- **Includes Cloudera Manager**
  - End-to-end administration for Hadoop
  - Deploy, manage, and monitor your cluster

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Cloudera Enterprise (1)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Cloudera Enterprise (2)

- **Subscription product including CDH and Cloudera Manager**
- **Provides advanced features, such as**
  - Operational and utilization reporting
  - Configuration history and rollbacks
  - Rolling updates and service restarts
  - External authentication (LDAP/SAML)
  - Automated backup and disaster recovery
- **Specific editions offer additional capabilities, such as**
  - Governance and data management (Cloudera Navigator)
  - Active data optimization (Cloudera Navigator Optimizer)
  - Comprehensive encryption (Cloudera Navigator Encrypt)
  - Key management (Cloudera Navigator Key Trustee)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Cloudera University

- **The leader in Apache Hadoop-based training**
- **We offer a variety of courses, both instructor-led (in physical or virtual classrooms) and self-paced OnDemand, including:**
  - *Developer Training for Apache Spark and Hadoop*
  - *Cloudera Administrator Training for Apache Hadoop*
  - *Cloudera Data Analyst Training*
  - *Cloudera Search Training*
  - *Cloudera Data Scientist Training*
  - *Cloudera Training for Apache HBase*
- **We also offer private courses:**
  - Can be delivered on-site, virtually, or online OnDemand
  - Can be tailored to suit customer needs

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Cloudera University OnDemand

- **Our OnDemand platform includes**
  - A full catalog of courses, with free updates
  - Free courses such as *Essentials* and *Cloudera Director*
  - Online-only modules such as *Cloudera Director* and *Cloudera Navigator*
  - Searchable content within and across courses
- **Courses include:**
  - Video lectures and demonstrations with searchable transcripts
  - Hands-on exercises through a browser-based virtual cluster environment
  - Discussion forums monitored by Cloudera course instructors
- **Purchase access to a library of courses (365 days) or individual courses (180 days)**
- **See the Cloudera OnDemand information page for more details or to make a purchase, or go directly to the OnDemand Course Catalog**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Accessing Cloudera OnDemand

- **Cloudera OnDemand subscribers can access their courses online through a web browser**

- **Cloudera OnDemand is also available through an iOS app**
  - Search for "Cloudera OnDemand" in the iOS App Store
  - iPad users: change your Store search settings to "iPhone only"

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Cloudera University Certification

- **The leader in Apache Hadoop-based certification**
- **All Cloudera professional certifications are hands-on, performance-based exams requiring you to complete a set of real-world tasks on a working multi-node CDH cluster**
- **We offer two levels of certifications**
  - Cloudera Certified Professional (CCP)
    - The industry's most demanding performance-based certification, CCP Data Engineer evaluates and recognizes your mastery of the technical skills most sought after by employers
    - CCP Data Engineer
  - Cloudera Certified Associate (CCA)
    - To achieve CCA certification successfully, you complete a set of core tasks on a working CDH cluster instead of guessing at multiple-choice questions
    - CCA Spark and Hadoop Developer
    - CCA Data Analyst
    - CCA Administrator

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Introduction

- About This Course
- About Cloudera
- **Course Logistics**
- Introductions
- Hands-On Exercise: Starting the Exercise Environment

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Logistics

- Class start and finish time
- Lunch
- Breaks
- Restrooms
- Wi-Fi access
- Virtual machines

Your instructor will give you details on how to access the course materials for the class

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Introduction

- About This Course
- About Cloudera
- Course Logistics
- **Introductions**
- Hands-On Exercise: Starting the Exercise Environment

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Introductions

- **About your instructor**
- **About you**
  - Where do you work? What do you do there?
  - How much Hadoop and Spark experience do you have?
  - What do you expect to gain from this course?
  - Which language do you plan to use in the course: Python or Scala?

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Introduction

- About This Course
- About Cloudera
- Course Logistics
- Introductions
- **Hands-On Exercise: Starting the Exercise Environment**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Starting the Exercise Environment

- **In this exercise, you will configure and launch your course exercise environment**
  - Please refer to the Hands-On Exercise Manual for instructions

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

cloudera®

Introduction to Apache Hadoop and the Hadoop Ecosystem

Chapter 2

cloudera®

Course Chapters

- Introduction
- **Introduction to Apache Hadoop and the Hadoop Ecosystem**
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Introduction to Apache Hadoop and the Hadoop Ecosystem

In this chapter, you will learn

- **What Apache Hadoop is and what kind of use cases it is best suited for**
- **How the major components of the Hadoop ecosystem fit together**
- **How to get started with the hands-on exercises in this course**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Introduction to Apache Hadoop and the Hadoop Ecosystem

- **Apache Hadoop Overview**
- Data Ingestion and Storage
- Data Processing
- Data Analysis and Exploration
- Other Ecosystem Tools
- Introduction to the Hands-On Exercises
- Essential Points
- Hands-On Exercise: Querying Hadoop Data with Apache Impala

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

What Is Apache Hadoop?

- **Scalable and economical data storage, processing, and analysis**
  - Distributed and fault-tolerant
  - Harnesses the power of industry standard hardware
- **Heavily inspired by technical documents published by Google**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Common Hadoop Use Cases

- **Hadoop is ideal for applications that handle data with high**
  - Volume
  - Velocity
  - Variety

- **Extract, Transform, and Load (ETL)**
- **Data analysis**
- **Text mining**
- **Index building**
- **Graph creation and analysis**
- **Pattern recognition**
- **Data storage**
- **Collaborative filtering**
- **Prediction models**
- **Sentiment analysis**
- **Risk assessment**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Distributed Processing with Hadoop

A Hadoop Cluster

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Introduction to Apache Hadoop and the Hadoop Ecosystem

- Apache Hadoop Overview
- **Data Ingestion and Storage**
- Data Processing
- Data Analysis and Exploration
- Other Ecosystem Tools
- Introduction to the Hands-On Exercises
- Essential Points
- Hands-On Exercise: Querying Hadoop Data with Apache Impala

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Data Ingestion and Storage

- **Hadoop typically ingests data from many sources and in many formats**
  - Traditional data management systems such as databases
  - Logs and other machine generated data (event data)
  - Imported files

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Data Storage: HDFS

- **Hadoop Distributed File System (HDFS)**
  - HDFS is the main storage layer for Hadoop
  - Provides inexpensive reliable storage for massive amounts of data on industry-standard hardware
  - Data is distributed when stored

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Data Storage: Apache Kudu

- **Apache Kudu**
  - Distributed columnar (key-value) storage for structured data
  - Allows random access and updating data (unlike HDFS)
  - Supports SQL-based analytics
  - Works directly on native file system; is not built on HDFS
  - Integrates with Spark, MapReduce, and Apache Impala
  - Created at Cloudera, donated to Apache Software Foundation
  - Covered in Cloudera's online OnDemand module *Introduction to Apache Kudu*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Data Ingest Tools (1)

- **HDFS**
  - Direct file transfer
- **Apache Sqoop**
  - High speed import to HDFS from relational database (and vice versa)
  - Supports many data storage systems
    - Examples: Netezza, MongoDB, MySQL, Teradata, and Oracle

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Data Ingest Tools (2)

- **Apache Flume**
  - Distributed service for ingesting streaming data
  - Ideally suited for event data from multiple systems
    - For example, log files
- **Apache Kafka**
  - A high throughput, scalable messaging system
  - Distributed, reliable publish-subscribe system
  - Integrates with Flume and Spark Streaming

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Introduction to Apache Hadoop and the Hadoop Ecosystem

- Apache Hadoop Overview
- Data Ingestion and Storage
- **Data Processing**
- Data Analysis and Exploration
- Other Ecosystem Tools
- Introduction to the Hands-On Exercises
- Essential Points
- Hands-On Exercise: Querying Hadoop Data with Apache Impala

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Apache Spark: An Engine for Large-Scale Data Processing

- **Spark is a large-scale data processing engine**
  - General purpose
  - Runs on Hadoop clusters and processes data in HDFS
- **Supports a wide range of workloads**
  - Machine learning
  - Business intelligence
  - Streaming
  - Batch processing
  - Querying structured data
- **This course uses Spark for data processing**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hadoop MapReduce: The Original Hadoop Processing Engine

- **Hadoop MapReduce is the original Hadoop framework for processing big data**
  - Primarily Java-based
- **Based on the map-reduce programming model**
- **The core Hadoop processing engine before Spark was introduced**
- **Still in use in many production systems**
  - But losing ground to Spark fast
- **Many existing tools are still built using MapReduce code**
- **Has extensive and mature fault tolerance built into the framework**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Introduction to Apache Hadoop and the Hadoop Ecosystem

- Apache Hadoop Overview
- Data Ingestion and Storage
- Data Processing
- **Data Analysis and Exploration**
- Other Ecosystem Tools
- Introduction to the Hands-On Exercises
- Essential Points
- Hands-On Exercise: Querying Hadoop Data with Apache Impala

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Apache Impala (incubating): High-Performance SQL

- **Impala is a high-performance SQL engine**
  - Runs on Hadoop clusters
  - Stores data in HDFS files, or in HBase or Kudu tables
  - Inspired by Google's Dremel project
  - Very low latency—measured in milliseconds
  - Ideal for interactive analysis
- **Impala supports a dialect of SQL (Impala SQL)**
  - Data in HDFS modeled as database tables
- **Impala was developed by Cloudera**
  - Donated to the Apache Software Foundation, where it is incubating
  - 100% open source, released under the Apache software license

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Apache Hive: SQL on MapReduce or Spark

- **Hive is an abstraction layer on top of Hadoop**
  - Hive uses a SQL-like language called HiveQL
    - Similar to Impala SQL
  - Useful for data processing and ETL
  - Impala is preferred for interactive analytics

- **Hive executes queries using MapReduce or Spark**

```
SELECT zipcode, SUM(cost) AS total
FROM customers
JOIN orders
ON (customers.cust_id = orders.cust_id)
WHERE zipcode LIKE '63%'
GROUP BY zipcode
ORDER BY total DESC;
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Cloudera Search: A Platform for Data Exploration

- **Interactive full-text search for data in a Hadoop cluster**
- **Allows non-technical users to access your data**
  - Nearly everyone can use a search engine
- **Cloudera Search enhances Apache Solr**
  - Integrates Apache Solr with HDFS, MapReduce, HBase, and Flume
  - Supports file formats widely used with Hadoop
  - Includes a dynamic web-based dashboard interface with Hue
- **Cloudera Search is 100% open source**
- **Cloudera Search is discussed in depth in *Cloudera Search Training* course**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Introduction to Apache Hadoop and the Hadoop Ecosystem

- Apache Hadoop Overview
- Data Ingestion and Storage
- Data Processing
- Data Analysis and Exploration
- **Other Ecosystem Tools**
- Introduction to the Hands-On Exercises
- Essential Points
- Hands-On Exercise: Querying Hadoop Data with Apache Impala

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hue: The UI for Hadoop

- **Hue = Hadoop User Experience**
- **Hue provides a web front-end to Hadoop**
  - Upload and browse data in HDFS
  - Query tables in Impala and Hive
  - Run Spark jobs
  - Build an interactive Cloudera Search dashboard
  - And much more
- **Makes Hadoop easier to use**
- **Created by Cloudera**
  - 100% open source
  - Released under Apache license

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Cloudera Manager

- **Cloudera tool for system administrators**
  - Provides an intuitive, web-based UI to manage a Hadoop cluster
- **Automatically installs Hadoop and Hadoop ecosystem tools**
  - Installs required services for each node's role in the cluster
- **Configures services with recommended default settings**
  - Administrators can adjust settings as needed
- **Lets administrators manage nodes and services throughout the cluster**
  - Start and stop nodes and individual services
  - Control user and group access to the cluster
- **Monitors cluster health and performance**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Introduction to Apache Hadoop and the Hadoop Ecosystem

- Apache Hadoop Overview
- Data Ingestion and Storage
- Data Processing
- Data Analysis and Exploration
- Other Ecosystem Tools
- **Introduction to the Hands-On Exercises**
- Essential Points
- Hands-On Exercise: Querying Hadoop Data with Apache Impala

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Introduction to the Hands-On Exercises

- **The best way to learn is to *do!***
- **Most topics in this course have Hands-On Exercises to practice the skills you have learned in the course**
- **The exercises are based on a hypothetical scenario**
  - However, the concepts apply to nearly any organization
- **Loudacre Mobile is a (fictional) fast-growing wireless carrier**
  - Provides mobile service to customers throughout western USA

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Scenario Explanation

- **Loudacre needs to migrate their existing infrastructure to Hadoop**
  - The size and velocity of their data has exceeded their ability to process and analyze their data
- **Loudacre data sources**
  - MySQL database: customer account data (name, address, phone numbers, and devices)
  - Apache web server logs from Customer Service site
  - HTML files: Knowledge Base articles
  - XML files: Device activation records
  - Real-time device status logs
  - Base station files: Cell tower locations

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Introduction to the Exercise Environment (1)

- **Your exercise environment**
  - Your VM automatically logs you into Linux with the username **training** (password **training**)
  - Most of your work is done on the gateway node of your cluster
  - Home directory on the gateway node is `/home/training` (often referred to as ~)
  - The environment includes all the software you need to complete the exercises
    - Spark and CDH (Cloudera's Distribution, including Apache Hadoop)
    - Various tools including Firefox, gedit, Emacs, and Apache Maven

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Introduction to the Exercise Environment (2)

- **The course exercise directory on the gateway node is** `~/training_materials/devsh`
- **This folder includes the files used in the course**
  - `examples`: all the example code in this course
  - `exercises`: starter files, scripts and solutions for the Hands-On Exercises
  - `scripts`: course setup and catch-up scripts
  - `data`: sample data for exercises

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Introduction to Apache Hadoop and the Hadoop Ecosystem

- Apache Hadoop Overview
- Data Ingestion and Storage
- Data Processing
- Data Analysis and Exploration
- Other Ecosystem Tools
- Introduction to the Hands-On Exercises
- **Essential Points**
- Hands-On Exercise: Querying Hadoop Data with Apache Impala

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **Hadoop is a framework for distributed storage and processing**
- **Core Hadoop includes HDFS for storage and YARN for cluster resource management**
- **The Hadoop ecosystem includes many components for**
  - Ingesting data (Flume, Sqoop, Kafka)
  - Storing data (HDFS, Kudu)
  - Processing data (Spark, Hadoop MapReduce, Pig)
  - Modeling data as tables for SQL access (Impala, Hive)
  - Exploring data (Hue, Search)
- **Hands-on exercises will let you practice and refine your Hadoop skills**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Bibliography

The following offer more information on topics discussed in this chapter

- ***Hadoop: The Definitive Guide*** **(published by O'Reilly)**
  - http://tiny.cloudera.com/hadooptdg
- ***Cloudera Essentials for Apache Hadoop*****—free online training**
  - http://tiny.cloudera.com/esscourse
- ***Cloudera Administrator Training for Apache Hadoop***
  - http://tiny.cloudera.com/admin
- ***Cloudera Search Training***
  - http://tiny.cloudera.com/search
- ***Cloudera Training for Apache HBase***
  - http://tiny.cloudera.com/hbase
- ***Cloudera Data Analyst Training***
  - http://tiny.cloudera.com/analyst

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Introduction to Apache Hadoop and the Hadoop Ecosystem

- Apache Hadoop Overview
- Data Ingestion and Storage
- Data Processing
- Data Analysis and Exploration
- Other Ecosystem Tools
- Introduction to the Hands-On Exercises
- Essential Points
- **Hands-On Exercise: Querying Hadoop Data with Apache Impala**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Query Hadoop Data with Apache Impala

- **In this exercise, you will use the Hue Impala Query Editor to explore data in a Hadoop cluster**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Apache Hadoop File Storage

Chapter 3

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- **Apache Hadoop File Storage**
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Apache Hadoop File Storage

In this chapter, you will learn

- **How the Apache Hadoop Distributed File System (HDFS) stores data across a cluster**
- **How to use HDFS using the Hue File Browser or the `hdfs` command**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Hadoop File Storage

- **Apache Hadoop Cluster Components**
- HDFS Architecture
- Using HDFS
- Essential Points
- Hands-On Exercise: Accessing HDFS with the Command Line and Hue

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hadoop Cluster Terminology

- **A *cluster* is a group of computers working together**
  - Provides data storage, data processing, and resource management
- **A *node* is an individual computer in the cluster**
  - *Master* nodes manage distribution of work and data to *worker* nodes
- **A *daemon* is a program running on a node**
  - Each performs different functions in the cluster

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Cluster Components

- There are three main components of a cluster
- They work together to provide distributed data processing

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Hadoop File Storage

- Apache Hadoop Cluster Components
- **HDFS Architecture**
- Using HDFS
- Essential Points
- Hands-On Exercise: Accessing HDFS with the Command Line and Hue

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

HDFS Basic Concepts (1)

- **HDFS is a file system written in Java**
  - Based on Google File System
- **Sits on top of a native file system**
  - Such as ext3, ext4, or xfs
- **Provides redundant storage for massive amounts of data**
  - Using readily available, industry-standard computers

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

HDFS Basic Concepts (2)

- **HDFS performs best with a "modest" number of large files**
  - Millions, rather than billions, of files
  - Each file typically 100MB or more
- **Files in HDFS are "write once"**
  - No random writes to files are allowed
- **HDFS is optimized for large, streaming reads of files**
  - Rather than random reads

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

How Files Are Stored (1)

- **Data files are split into blocks (default 128MB) which are distributed at load time**
- **The actual blocks are stored on cluster worker nodes running the Hadoop HDFS Data Node service**
  - Often referred to as *DataNodes*
- **Each block is replicated on multiple DataNodes (default 3x)**
- **A cluster master node runs the HDFS Name Node service, which stores file metadata**
  - Often referred to as the *NameNode*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

How Files Are Stored (2)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Hadoop File Storage

- Apache Hadoop Cluster Components
- HDFS Architecture
- **Using HDFS**
- Essential Points
- Hands-On Exercise: Accessing HDFS with the Command Line and Hue

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Options for Accessing HDFS

- **From the command line**
  - `$ hdfs dfs`
- **In Spark**
  - By URI:
    `hdfs://nnhost:port/file…`
- **Other programs**
  - Java API
    - Used by Hadoop tools such as MapReduce, Impala, Hue, Sqoop, Flume
  - RESTful interface

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

HDFS Command Line Examples (1)

- **Copy file `foo.txt` from local disk to the user's directory in HDFS**

```
$ hdfs dfs -put foo.txt foo.txt
```

  - This will copy the file to `/user/`*`username`*`/foo.txt`

- **Get a directory listing of the user's home directory in HDFS**

```
$ hdfs dfs -ls
```

- **Get a directory listing of the HDFS root directory**

```
$ hdfs dfs -ls /
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

HDFS Command Line Examples (2)

- **Display the contents of the HDFS file `/user/fred/bar.txt`**

```
$ hdfs dfs -cat /user/fred/bar.txt
```

- **Copy that file to the local disk, named as `baz.txt`**

```
$ hdfs dfs -get /user/fred/bar.txt baz.txt
```

- **Create a directory called `input` under the user's home directory**

```
$ hdfs dfs -mkdir input
```

Note: `copyFromLocal` is a synonym for `put`; `copyToLocal` is a synonym for `get`

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

HDFS Command Line Examples (3)

- **Delete a file**

```
$ hdfs dfs -rm input_old/myfile
```

- **Delete a set of files using a wildcard**

```
$ hdfs dfs -rm input_old/*
```

- **Delete the directory `input_old` and all its contents**

```
$ hdfs dfs -rm -r input_old
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

The Hue HDFS File Browser

- **The File Browser in Hue lets you view and manage your HDFS directories and files**
  - Create, move, rename, modify, upload, download, and delete directories and files
  - View file contents

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Hadoop File Storage

- Apache Hadoop Cluster Components
- HDFS Architecture
- Using HDFS
- **Essential Points**
- Hands-On Exercise: Accessing HDFS with the Command Line and Hue

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- The Hadoop Distributed File System (HDFS) is the main storage layer for Hadoop
- HDFS chunks data into blocks and distributes them across the cluster when data is stored
- HDFS clusters are managed by a single NameNode running on a master node
- Access HDFS using Hue, the `hdfs` command, or the HDFS API

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Bibliography

The following offer more information on topics discussed in this chapter

- **HDFS User Guide**
  - http://tiny.cloudera.com/hdfsuser
- **Hue website**
  - http://gethue.com/

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Hadoop File Storage

- Apache Hadoop Cluster Components
- HDFS Architecture
- Using HDFS
- Essential Points
- **Hands-On Exercise: Accessing HDFS with the Command Line and Hue**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Accessing HDFS with the Command Line and Hue

- **In this exercise, you will practice managing and viewing data files**
  - Please refer to the Hands-On Exercise Manual for instructions

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

cloudera®

Distributed Processing on an Apache Hadoop Cluster

Chapter 4

cloudera®

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- **Distributed Processing on an Apache Hadoop Cluster**
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Distributed Processing on an Apache Hadoop Cluster

In this chapter, you will learn

- **How Hadoop YARN provides cluster resource management for distributed data processing**
- **How to use Hue, the YARN web UI, or the `yarn` command to monitor your cluster**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Distributed Processing on an Apache Hadoop Cluster

- **YARN Architecture**
- Working With YARN
- Essential Points
- Hands-On Exercise: Running and Monitoring a YARN Job

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

What Is YARN?

- **YARN = Yet Another Resource Negotiator**
- **YARN is the Hadoop processing layer that contains**
  - A resource manager
  - A job scheduler

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

YARN Daemons

- **ResourceManager (RM)**
  - Runs on master node
  - Global resource scheduler
  - Arbitrates system resources between competing applications
  - Has a pluggable scheduler to support different algorithms (such as Capacity or Fair Scheduler)

Resource Manager

- **NodeManager (NM)**
  - Runs on worker nodes
  - Communicates with RM
  - Manages node resources
  - Launches containers

Node Manager

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Applications on YARN (1)

- **Containers**
  - Containers allocate a certain amount of resources (memory, CPU cores) on a worker node
  - Applications run in one or more containers
  - Applications request containers from RM
- **ApplicationMaster (AM)**
  - One per application
  - Framework/application specific
  - Runs in a container
  - Requests more containers to run application tasks

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Applications on YARN (2)

- **Each application consists of one or more containers**
  - The ApplicationMaster runs in one container
  - The application's distributed processes (JVMs) run in other containers
    - The processes run in parallel, and are managed by the AM
    - The processes are called executors in Apache Spark and tasks in Hadoop MapReduce
- **Applications are typically submitted to the cluster from an *edge* or *gateway* node**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Running an Application on YARN (1)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Running an Application on YARN (2)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Running an Application on YARN (3)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Running an Application on YARN (4)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Running an Application on YARN (5)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Running an Application on YARN (6)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Running an Application on YARN (7)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Running an Application on YARN (8)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Running an Application on YARN (9)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Distributed Processing on an Apache Hadoop Cluster

- YARN Architecture
- **Working With YARN**
- Essential Points
- Hands-On Exercise: Running and Monitoring a YARN Job

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Working with YARN

- **Developers need to be able to**
  - Submit jobs (applications) to run on the YARN cluster
  - Monitor and manage jobs
- **There are three major YARN tools for developers**
  - The Hue Job Browser
  - The YARN ResourceManager web UI
  - The YARN command line
- **YARN administrators can use Cloudera Manager**
  - May also be helpful for developers
  - Included in Cloudera Express and Cloudera Enterprise

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

The Hue Job Browser

- **The Hue Job Browser allows you to**
  - Monitor the status of a job
  - View a job's logs
  - Kill a running job

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

The YARN Resource Manager Web UI

- **The ResourceManager UI is the main entry point**
  - Runs on the RM host on port 8088 by default
- **Provides more detailed view than Hue**
- **Does not provide any control or configuration**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

ResourceManager UI: Nodes

Logged in as: dr.who

Nodes of the cluster

- Cluster
  - About
  - Nodes
  - Applications
    - NEW
    - NEW_SAVING
    - SUBMITTED
    - ACCEPTED
    - RUNNING
    - FINISHED
    - FAILED
    - KILLED
  - Scheduler
- Tools

Cluster Overview

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
4	0	0	4	0	0 B	4 GB	0 B	0	6	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
3	0	0	0	0	0

User Metrics for dr.who

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Containers Pending	Containers Reserved	Memory Used	Memory Pending	Memory Reserved	VCores Used	VCores Pending	VCores Reserved
0	0	0	0	0	0	0	0 B	0 B	0 B	0	0	0

Show 20 entries Search:

Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Mem Used	Mem Avail	VCores Used	VCores Avail	Version
	/default	RUNNING	worker-1:8041	worker-1:8042	Tue Apr 11 13:15:25 -0700 2017		0	0 B	1.50 GB	0	2	2.6.0-cdh5.10.0
	/default	RUNNING	worker-3:8041	worker-3:8042	Tue Apr 11 13:15:26 -0700 2017		0	0 B	1.50 GB	0	2	2.6.0-cdh5.10.0
	/default	RUNNING	worker-2:8041	worker-2:8042	Tue Apr 11 13:15:26 -0700 2017		0	0 B	1 GB	0	2	2.6.0-cdh5.10.0

Link to Node Manager UI

List of all nodes in the cluster

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

ResourceManager UI: Applications

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

ResourceManager UI: Application Detail

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

YARN Command Line (1)

- **Command to configure and view information about the YARN cluster**

```
$ yarn command
```

- **Most YARN commands are for administrators rather than developers**
- **Some helpful commands for developers**
  - List running applications

```
$ yarn application -list
```

  - Kill a running application

```
$ yarn application -kill app-id
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

YARN Command Line (2)

- **View the logs of the specified application**

```
$ yarn logs -applicationId app-id
```

- **View the full list of command options**

```
$ yarn -help
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Cloudera Manager

- **Cloudera Manager providers a greater ability to monitor and configure a cluster from a single location**
  - Covered in *Cloudera Administrator Training for Apache Hadoop*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Distributed Processing on an Apache Hadoop Cluster

- YARN Architecture
- Working With YARN
- **Essential Points**
- Hands-On Exercise: Running and Monitoring a YARN Job

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- YARN manages resources in a Hadoop cluster and schedules applications
- Worker nodes run NodeManager daemons, managed by a ResourceManager on a master node
- Applications running on YARN consist of an ApplicationMaster and one or more executors
- Use Hue, the YARN ResourceManager web UI, the `yarn` command, or Cloudera Manager to monitor applications

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Bibliography

The following offer more information on topics discussed in this chapter

- ***Hadoop Application Architectures: Designing Real-World Big Data Applications*** **(published by O'Reilly)**
  - http://tiny.cloudera.com/archbook
- **YARN documentation**
  - http://tiny.cloudera.com/yarndocs
- **Cloudera Engineering Blog YARN articles**
  - http://tiny.cloudera.com/yarnblog

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Distributed Processing on an Apache Hadoop Cluster

- YARN Architecture
- Working With YARN
- Essential Points
- **Hands-On Exercise: Running and Monitoring a YARN Job**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Running and Monitoring a YARN Job

- **In this exercise, you will submit an application to the cluster and monitor it using the available tools**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

cloudera®

Apache Spark Basics

Chapter 5

cloudera®

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- **Apache Spark Basics**
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Apache Spark Basics

In this chapter, you will learn

- **How Spark SQL fits into the Spark stack**
- **How to start and use the Python and Scala Spark shells**
- **What a DataFrame is and how to perform simple queries**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Basics

- **What is Apache Spark?**
- Starting the Spark Shell
- Using the Spark Shell
- Getting Started with Datasets and DataFrames
- DataFrame Operations
- Essential Points
- Hands-On Exercise: Exploring DataFrames Using the Apache Spark Shell

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

What Is Apache Spark?

- **Apache Spark is a fast and general engine for large-scale data processing**
- **Written in Scala**
  - Functional programming language that runs in a JVM
- **Spark shell**
  - Interactive—for learning, data exploration, or ad hoc analytics
  - Python or Scala
- **Spark applications**
  - For large scale data processing
  - Python, Scala, or Java

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

The Spark Stack

- **Spark provides a stack of libraries built on core Spark**
  - Core Spark provides the fundamental Spark abstraction: Resilient Distributed Datasets (RDDs)
  - Spark SQL works with structured data
  - MLlib supports scalable machine learning
  - Spark Streaming applications process data in real time
  - GraphX works with graphs and graph-parallel computation

cloudera®
Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark SQL

- **Spark SQL is a Spark library for working with structured data**
- **What does Spark SQL provide?**
  - The DataFrame and Dataset API
    - The primary entry point for developing Spark applications
    - DataFrames and Datasets are abstractions for representing structured data
  - Catalyst Optimizer—an extensible optimization framework
  - A SQL engine and command line interface

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Basics

- What is Apache Spark?
- **Starting the Spark Shell**
- Using the Spark Shell
- Getting Started with Datasets and DataFrames
- DataFrame Operations
- Essential Points
- Hands-On Exercise: Exploring DataFrames Using the Apache Spark Shell

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

The Spark Shell

- **The Spark shell provides an interactive Spark environment**
  - Often called a *REPL*, or Read/Evaluate/Print Loop
  - For learning, testing, data exploration, or ad hoc analytics
  - You can run the Spark shell using either Python or Scala
- **You typically run the Spark shell on a gateway node**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Starting the Spark Shell

- **On a Cloudera cluster, the command to start the Spark 2 shell is**
  - `pyspark2` for Python
  - `spark2-shell` for Scala
- **The Spark shell has a number of different start-up options, including**
  - `master`: specify the cluster to connect to
  - `jars`: Additional JAR files (Scala only)
  - `py-files`: Additional Python files (Python only)
  - `name`: the name the Spark Application UI uses for this application
    - Defaults to `PySparkShell` (Python) or `Spark shell` (Scala)
  - `help`: Show all the available shell options

```
$ pyspark2 --name "My Application"
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark Cluster Options (1)

- **Spark applications can run on these types of clusters**
  - Apache Hadoop YARN
  - Apache Mesos
  - Spark Standalone
- **They can also run locally instead of on a cluster**
- **The default cluster type for the Spark 2 Cloudera add-on service is YARN**
- **Specify the type or URL of the cluster using the `master` option**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark Cluster Options (2)

- **The possible values for the `master` option include**
  - `yarn`
  - `spark://masternode:port` (Spark Standalone)
  - `mesos://masternode:port` (Mesos)
  - `local[*]` runs locally with as many threads as cores (default)
  - `local[n]` runs locally with *n* threads
  - `local` runs locally with a single thread

```
$ pyspark2 --master yarn
```

Language: *Python*

```
$ spark2-shell --master yarn
```

Language: *Scala*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Basics

- What is Apache Spark?
- Starting the Spark Shell
- **Using the Spark Shell**
- Getting Started with Datasets and DataFrames
- DataFrame Operations
- Essential Points
- Hands-On Exercise: Exploring DataFrames Using the Apache Spark Shell

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark Session

- The main entry point for the Spark API is a Spark session
- The Spark shell provides a preconfigured `SparkSession` object called `spark`

```
...
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.1.0.cloudera1
      /_/

Using Python version 2.7.8 (default, Nov 14 2016 05:07:18)
SparkSession available as 'spark'.
>>> spark
<pyspark.sql.session.SparkSession object at 0x1928b90>
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Working with the Spark Session

- **The `SparkSession` class provides functions and attributes to access all of Spark functionality**
- **Examples include**
  - `sql`: execute a Spark SQL query
  - `catalog`: entry point for the Catalog API for managing tables
  - `read`: function to read data from a file or other data source
  - `conf`: object to manage Spark configuration settings
  - `sparkContext`: entry point for core Spark API

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Log Levels (1)

- Spark logs messages using Apache Log4J
- Messages are tagged with their level

```
…
17/04/03 11:30:01 WARN Utils: Service 'SparkUI' …
17/04/03 11:30:02 INFO SparkContext: Created broadcast 0 …
17/04/03 11:30:02 INFO FileInputFormat: Total input …
17/04/03 11:30:02 INFO SparkContext: Starting job: …
17/04/03 11:30:02 INFO DAGScheduler: Got job 0 …
…
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Log Levels (2)

- **Available log levels are**
  - `TRACE`
  - `DEBUG`
  - `INFO` (default level in Spark applications)
  - `WARN` (default level in Spark shell)
  - `ERROR`
  - `FATAL`
  - `OFF`

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Setting the Log Level

- Set the log level for the current Spark shell using the Spark context `setLogLevel` method

```
spark.sparkContext.setLogLevel("INFO")
```

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Basics

- What is Apache Spark?
- Starting the Spark Shell
- Using the Spark Shell
- **Getting Started with Datasets and DataFrames**
- DataFrame Operations
- Essential Points
- Hands-On Exercise: Exploring DataFrames Using the Apache Spark Shell

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrames and Datasets

- **DataFrames and Datasets are the primary representation of data in Spark**
- **DataFrames represent structured data in a tabular form**
  - DataFrames model data similar to tables in an RDBMS
  - DataFrames consist of a collection of loosely typed `Row` objects
  - Rows are organized into columns described by a schema
- **Datasets represent data as a collection of objects of a specified type**
  - Datasets are strongly-typed—type checking is enforced at compile time rather than run time
  - An associated schema maps object properties to a table-like structure of rows and columns
  - Datasets are only defined in Scala and Java
  - `DataFrame` is an alias for `Dataset[Row]`—Datasets containing `Row` objects

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrames and Rows

- **DataFrames contain an ordered collection of Row objects**
  - Rows contain an ordered collection of values
  - Row values can be basic types (such as integers, strings, and floats) or collections of those types (such as arrays and lists)
  - A schema maps column names and types to the values in a row

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Creating a DataFrame (1)

- **The `users.json` text file contains sample data**
  - Each line contains a single JSON record that can include a name, age, and postal code field

```
{"name":"Alice", "pcode":"94304"}
{"name":"Brayden", "age":30, "pcode":"94304"}
{"name":"Carla", "age":19, "pcode":"10036"}
{"name":"Diana", "age":46}
{"name":"Étienne", "pcode":"94104"}
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Creating a DataFrame (2)

- Create a DataFrame using `spark.read`
- Returns the Spark session's `DataFrameReader`
- Call `json` function to create a new DataFrame

```
> usersDF = \
  spark.read.json("users.json")
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Creating a DataFrame (3)

- DataFrames always have an associated schema
- `DataFrameReader` can infer the schema from the data
- Use `printSchema` to show the DataFrame's schema

```
> usersDF = \
  spark.read.json("users.json")

> usersDF.printSchema()
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)
 |-- pcode: string (nullable = true)
```

Language: *Python*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Creating a DataFrame (4)

- The show method displays the first few rows in a tabular format

```
> usersDF = \
  spark.read.json("users.json")

> usersDF.printSchema()
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)
 |-- pcode: string (nullable = true)

> usersDF.show()
+----+-------+-----+
| age|   name|pcode|
+----+-------+-----+
|null|  Alice|94304|
|  30|Brayden|94304|
|  19|  Carla|10036|
|  46|  Diana| null|
|null|Etienne|94104|
+----+-------+-----+
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Basics

- What is Apache Spark?
- Starting the Spark Shell
- Using the Spark Shell
- Getting Started with Datasets and DataFrames
- **DataFrame Operations**
- Essential Points
- Hands-On Exercise: Exploring DataFrames Using the Apache Spark Shell

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrame Operations

- **There are two main types of DataFrame operations**
  - *Transformations* create a new DataFrame based on existing one(s)
    - Transformations are executed in parallel by the application's executors
  - *Actions* output data values from the DataFrame
    - Output is typically returned from the executors to the main Spark program (called the *driver*) or saved to a file

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrame Operations: Actions

- **Some common DataFrame actions include**
  - `count`: returns the number of rows
  - `first`: returns the first row (synonym for `head()`)
  - `take(n)`: returns the first *n* rows as an array (synonym for `head(n)`)
  - `show(n)`: display the first *n* rows in tabular form (default is 20 rows)
  - `collect`: returns all the rows in the DataFrame as an array
  - `write`: save the data to a file or other data source

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: take Action

```
> usersDF = spark.read.json("users.json")

> users = usersDF.take(3)
[Row(age=None, name=u'Alice', pcode=u'94304'),
 Row(age=30, name=u'Brayden', pcode=u'94304'),
 Row(age=19, name=u'Carla', pcode=u'10036')]
```

Language: *Python*

```
> val usersDF = spark.read.json("users.json")

> val users = usersDF.take(3)
usersDF: Array[org.apache.spark.sql.Row] =
  Array([null,Alice,94304],
        [30,Brayden,94304],
        [19,Carla,10036])
```

Language: *Scala*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrame Operations: Transformations (1)

- **Transformations create a new DataFrame based on an existing one**
  - The new DataFrame may have the same schema or a different one
- **Transformations do not return any values or data to the driver**
  - Data remains distributed across the application's executors
- **DataFrames are immutable**
  - Data in a DataFrame is never modified
  - Use transformations to create a new DataFrame with the data you need

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrame Operations: Transformations (2)

- **Common transformations include**
  - `select`: only the specified columns are included
  - `where`: only rows where the specified expression is true are included (synonym for `filter`)
  - `orderBy`: rows are sorted by the specified column(s) (synonym for `sort`)
  - `join`: joins two DataFrames on the specified column(s)
  - `limit(n)`: creates a new DataFrame with only the first `n` rows

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: `select` and `where` Transformations

```
> nameAgeDF = usersDF.select("name","age")
> nameAgeDF.show()
+-------+----+
|   name| age|
+-------+----+
|  Alice|null|
|Brayden|  30|
|  Carla|  19|
|  Diana|  46|
|Etienne|null|
+-------+----+

> over20DF = usersDF.where("age > 20")
> over20DF.show()
+---+-------+-----+
|age|   name|pcode|
+---+-------+-----+
| 30|Brayden|94304|
| 46|  Diana| null|
+---+-------+-----+
```

Language: *Python*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Defining Queries

- **A sequence of transformations followed by an action is a *query***

```
> nameAgeDF = usersDF.select("name","age")
> nameAgeOver20DF = nameAgeDF.where("age > 20")
> nameAgeOver20DF.show()
+---+-------+
|age|   name|
+---+-------+
| 30|Brayden|
| 46|  Diana|
+---+-------+
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chaining Transformations (1)

- **Transformations in a query can be chained together**
- **These two examples perform the same query in the same way**
  - Differences are only syntactic

```
> nameAgeDF = usersDF.select("name","age")
> nameAgeOver20DF = nameAgeDF.where("age > 20")
> nameAgeOver20DF.show()
```

Language: *Python*

```
> usersDF.select("name","age").where("age > 20").show()
```

Language: *Python*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chaining Transformations (2)

- **This is the same example with Scala**
  - The two code snippets are equivalent

```
> val nameAgeDF = usersDF.select("name","age")
> val nameAgeOver20DF = nameAgeDF.where("age > 20")
> nameAgeOver20DF.show
```

Language: *Scala*

```
> usersDF.select("name","age").where("age > 20").show
```

Language: *Scala*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Basics

- What is Apache Spark?
- Starting the Spark Shell
- Using the Spark Shell
- Getting Started with Datasets and DataFrames
- DataFrame Operations
- **Essential Points**
- Hands-On Exercise: Exploring DataFrames Using the Apache Spark Shell

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **Apache Spark is a framework for analyzing and processing big data**
- **The Python and Scala Spark shells are command line REPLs for executing Spark interactively**
  - Spark applications run in batch mode outside the shell
- **DataFrames represent structured data in tabular form by applying a schema**
- **Types of DataFrame operations**
  - Transformations create new DataFrames by transforming data in existing ones
  - Actions collect values in a DataFrame and either save them or return them to the Spark driver
- **A query consists of a sequence of transformations followed by an action**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Basics

- What is Apache Spark?
- Starting the Spark Shell
- Using the Spark Shell
- Getting Started with Datasets and DataFrames
- DataFrame Operations
- Essential Points
- **Hands-On Exercise: Exploring DataFrames Using the Apache Spark Shell**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Introduction to Spark Exercises: Choose Your Language

- **Your choice: Python or Scala**
  - For the Spark-based exercises in this course, you may choose to work with either Python or Scala
- **Solution and example files**
  - `.pyspark`: Python commands that can be copied into the PySpark shell
  - `.scalaspark`: Scala commands that can be copied into the Scala Spark shell
  - `.py`: Complete Python Spark applications
  - `.scala`: Complete Scala Spark applications

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Exploring DataFrames using the Apache Spark Shell

- **In this exercise, you will start the Python or Scala Spark shell and practice creating and querying a DataFrame**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Working with DataFrames and Schemas

Chapter 6

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- **Working with DataFrames and Schemas**
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Working with DataFrames and Schemas

In this chapter, you will learn

- **How to create DataFrames from a variety of sources**
- **How to specify format and options to save DataFrames**
- **How to define a DataFrame schema through inference or programmatically**
- **The difference between lazy and eager query execution**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Working with DataFrames and Schemas

- **Creating DataFrames from Data Sources**
- Saving DataFrames to Data Sources
- DataFrame Schemas
- Eager and Lazy Execution
- Essential Points
- Hands-On Exercise: Working with DataFrames and Schemas

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrame Data Sources

- **DataFrames read data from and write data to *data sources***
- **Spark SQL supports a wide range of data source types and formats**
  - Text files
    - CSV
    - JSON
    - Plain text
  - Binary format files
    - Apache Parquet
    - Apache ORC
  - Tables
    - Hive metastore
    - JDBC
- **You can also use custom or third-party data source types**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrames and Apache Parquet Files

- **Parquet is a very common file format for DataFrame data**
- **Features of Parquet**
  - Optimized binary storage of structured data
  - Schema metadata is embedded in the file
  - Efficient performance and size for large amounts of data
  - Supported by many Hadoop ecosystem tools
    - Spark, Hadoop MapReduce, Hive, Impala, and others
- **Use `parquet-tools` to view Parquet file schema and data**
  - Use `head` to display the first few records

```
$ parquet-tools head mydatafile.parquet
```

  - Use `schema` to view the schema

```
$ parquet-tools schema mydatafile.parquet
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Creating a DataFrame from a Data Source

- `spark.read` **returns a** `DataFrameReader` **object**
- **Use** `DataFrameReader` **settings to specify how to load data from the data source**
  - `format` indicates the data source type, such as `csv`, `json`, or `parquet` (the default is `parquet`)
  - `option` specifies a key/value setting for the underlying data source
  - `schema` specifies a schema to use instead of inferring one from the data source
- **Create the DataFrame based on the data source**
  - `load` loads data from a file or files
  - `table` loads data from a Hive table

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Examples: Creating a DataFrame from a Data Source

- **Example: Read a CSV text file**
  - Treat the first line in the file as a header instead of data

```
myDF = spark.read. \
  format("csv"). \
  option("header","true"). \
  load("/loudacre/myFile.csv")
```

Language: *Python*

- **Example: Read a table defined in the Hive metastore**

```
myDF = spark.read.table("my_table")
```

Language: *Python*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrameReader Convenience Functions

- **You can call a format-specific load function**
  - A shortcut instead of setting the format and using `load`
- **The following two code examples are equivalent**

```
spark.read.format("csv").load("/loudacre/myFile.csv")
```

```
spark.read.csv("/loudacre/myFile.csv")
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Specifying Data Source File Locations

- **You must specify a location when reading from a file data source**
  - The location can be a single file, a list of files, a directory, or a wildcard
  - Examples
    - `spark.read.json("myfile.json")`
    - `spark.read.json("mydata/")`
    - `spark.read.json("mydata/*.json")`
    - `spark.read.json("myfile1.json","myfile2.json")`
- **Files and directories are referenced by absolute or relative URI**
  - Relative URI (uses default file system)
    - `myfile.json`
  - Absolute URI
    - `hdfs://`*`nnhost`*`/loudacre/myfile.json`
    - `file:/home/training/myfile.json`

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Creating DataFrames from Data in Memory

- **You can also create DataFrames from a collection of in-memory data**
  - Useful for testing and integrations

```
val mydata = List(("Josiah","Bartlett"),
                  ("Harry","Potter"))

val myDF = spark.createDataFrame(mydata)
myDF.show
+------+--------+
|    _1|      _2|
+------+--------+
|Josiah|Bartlett|
| Harry|  Potter|
+------+--------+
```

Language: *Scala*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Working with DataFrames and Schemas

- Creating DataFrames from Data Sources
- **Saving DataFrames to Data Sources**
- DataFrame Schemas
- Eager and Lazy Execution
- Essential Points
- Hands-On Exercise: Working with DataFrames and Schemas

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Key `DataFrameWriter` Functions

- **The DataFrame `write` function returns a `DataFrameWriter`**
  - Saves data to a data source such as a table or set of files
  - Works similarly to `DataFrameReader`
- **`DataFrameWriter` methods**
  - `format` specifies a data source type
  - `mode` determines the behavior if the directory or table already exists
    - `error`, `overwrite`, `append`, or `ignore` (default is `error`)
  - `partitionBy` stores data in partitioned directories in the form *`column=value`* (as with Hive/Impala partitioning)
  - `option` specifies properties for the target data source
  - `save` saves the data as files in the specified directory
    - Or use `json`, `csv`, `parquet`, and so on
  - `saveAsTable` saves the data to a Hive metastore table
    - Uses default table location (`/user/hive/warehouse`)
    - Set `path` option to override location

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Examples: Saving a DataFrame to a Data Source

- **Example: Write data to a Hive metastore table called `my_table`**
  - Append the data if the table already exists
  - Use an alternate location

```
myDF.write. \
  mode("append"). \
  option("path","/loudacre/mydata"). \
  saveAsTable("my_table")
```

- **Example: Write data as Parquet files in the `mydata` directory**

```
myDF.write.save("mydata")
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Saving Data to Files

- **When you save data from a DataFrame, you must specify a directory**
  - Spark saves the data to one or more `part-` files in the directory

```
myDF.write.json("mydata")
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Working with DataFrames and Schemas

- Creating DataFrames from Data Sources
- Saving DataFrames to Data Sources
- **DataFrame Schemas**
- Eager and Lazy Execution
- Essential Points
- Hands-On Exercise: Working with DataFrames and Schemas

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DataFrame Schemas

- **Every DataFrame has an associated schema**
  - Defines the names and types of columns
  - Immutable and defined when the DataFrame is created

```
myDF.printSchema()
root
 |-- lastName: string (nullable = true)
 |-- firstName: string (nullable = true)
 |-- age: integer (nullable = true)
```

- **When creating a new DataFrame from a data source, the schema can be**
  - Automatically inferred from the data source
  - Specified programmatically
- **When a DataFrame is created by a transformation, Spark calculates the new schema based on the query**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Inferred Schemas

- **Spark can infer schemas from structured data, such as**
  - Parquet files—schema is embedded in the file
  - Hive tables—schema is defined in the Hive metastore
  - Parent DataFrames
- **Spark can also attempt to infer a schema from semi-structured data sources**
  - For example, JSON and CSV

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Inferring the Schema of a CSV File (No Header)

```
02134,Hopper,Grace,52
94020,Turing,Alan,32
94020,Lovelace,Ada,28
87501,Babbage,Charles,49
02134,Wirth,Niklaus,48
```

```
spark.read.option("inferSchema","true").csv("people.csv"). \
  printSchema()
root
 |-- _c0: integer (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: integer (nullable = true)
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Inferring the Schema of a CSV File (with Header)

```
pcode,lastName,firstName,age
02134,Hopper,Grace,52
94020,Turing,Alan,32
94020,Lovelace,Ada,28
87501,Babbage,Charles,49
02134,Wirth,Niklaus,48
```

```
spark.read.option("inferSchema","true"). \
  option("header","true").csv("people.csv"). \
  printSchema()
root
 |-- pcode: integer (nullable = true)
 |-- lastName: string (nullable = true)
 |-- firstName: string (nullable = true)
 |-- age: integer (nullable = true)
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Inferred Schemas versus Manual Schemas

- **Drawbacks to relying on Spark's automatic schema inference**
  - Inference requires an initial file scan, which may take a long time
  - The schema may not be correct for your use case
- **You can define the schema manually instead**
  - A schema is a `StructType` object containing a list of `StructField` objects
  - Each `StructField` represents a column in the schema, specifying
    - Column name
    - Column data type
    - Whether the data can be null (optional—the default is true)

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Incorrect Schema Inference

- **Example: This inferred schema for the CSV file is incorrect**
  - The `pcode` column should be a string

```
spark.read.option("inferSchema","true"). \
  option("header","true").csv("people.csv"). \
  printSchema()
root
 |-- pcode: integer (nullable = true)
 |-- lastName: string (nullable = true)
 |-- firstName: string (nullable = true)
 |-- age: integer (nullable = true)
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Defining a Schema Programmatically (Python)

```
from pyspark.sql.types import *

columnsList = [
  StructField("pcode", StringType()),
  StructField("lastName", StringType()),
  StructField("firstName", StringType()),
  StructField("age", IntegerType())]

peopleSchema = StructType(columnsList)
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Defining a Schema Programmatically (Scala)

```
import org.apache.spark.sql.types._

val columnsList = List(
  StructField("pcode", StringType),
  StructField("lastName", StringType),
  StructField("firstName", StringType),
  StructField("age", IntegerType))

val peopleSchema = StructType(columnsList)
```

Language: *Scala*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Applying a Schema Manually

```
spark.read.option("header","true").
  schema(peopleSchema).csv("people.csv").printSchema()
root
 |-- pcode: string (nullable = true)
 |-- lastName: string (nullable = true)
 |-- firstName: string (nullable = true)
 |-- age: integer (nullable = true)
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Working with DataFrames and Schemas

- Creating DataFrames from Data Sources
- Saving DataFrames to Data Sources
- DataFrame Schemas
- **Eager and Lazy Execution**
- Essential Points
- Hands-On Exercise: Working with DataFrames and Schemas

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Eager and Lazy Execution

- **Operations are *eager* when they are executed as soon as the statement is reached in the code**
- **Operations are *lazy* when the execution occurs only when the result is referenced**
- **Spark queries execute both lazily and eagerly**
  - DataFrame schemas are determined eagerly
  - Data transformations are executed lazily
- **Lazy execution is triggered when an action is called on a series of transformations**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Eager and Lazy Execution (1)

```
> usersDF = \
  spark.read.json("users.json")
```

Language: *Python*

users.json

name	age	pcode

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Eager and Lazy Execution (2)

```
> usersDF = \
  spark.read.json("users.json")
> nameAgeDF = \
  usersDF.select("name","age")
```

Language: *Python*

users.json

name	age	pcode

name	age

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Eager and Lazy Execution (3)

```
> usersDF = \
  spark.read.json("users.json")
> nameAgeDF = \
  usersDF.select("name","age")
> nameAgeDF.show()
```

Language: *Python*

users.json

name	age	pcode
Alice	(null)	94304
Brayden	30	94304
…	…	…

name	age
Alice	(null)
Brayden	30
…	…

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Eager and Lazy Execution (4)

```
> usersDF = \
  spark.read.json("users.json")
> nameAgeDF = \
  usersDF.select("name","age")
> nameAgeDF.show()
+-------+----+
|   name| age|
+-------+----+
|  Alice|null|
|Brayden|  30|
|  Carla|  19|
…
```

Language: *Python*

users.json

name	age	pcode
Alice	(null)	94304
Brayden	30	94304
…	…	…

name	age
Alice	(null)
Brayden	30
…	…

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Working with DataFrames and Schemas

- Creating DataFrames from Data Sources
- Saving DataFrames to Data Sources
- DataFrame Schemas
- Eager and Lazy Execution
- **Essential Points**
- Hands-On Exercise: Working with DataFrames and Schemas

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **DataFrames can be loaded from and saved to several different types of data sources**
  - Semi-structured text files like CSV and JSON
  - Structured binary formats like Parquet and ORC
  - Hive and JDBC tables
- **DataFrames can infer a schema from a data source, or you can define one manually**
- **DataFrame schemas are determined *eagerly* (at creation) but queries are executed *lazily* (when an action is called)**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Working with DataFrames and Schemas

- Creating DataFrames from Data Sources
- Saving DataFrames to Data Sources
- DataFrame Schemas
- Eager and Lazy Execution
- Essential Points
- **Hands-On Exercise: Working with DataFrames and Schemas**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Working with DataFrames and Schemas

- **In this exercise, you will create DataFrames and define schemas based on different data sources**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Analyzing Data with DataFrame Queries

Chapter 7

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- **Analyzing Data with DataFrame Queries**
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Analyzing Data with DataFrame Queries

In this chapter, you will learn

- **How to use column names, column references, and column expressions when querying**
- **How to group and aggregate values in a column**
- **How to join two DataFrames**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Analyzing Data with DataFrame Queries

- **Querying DataFrames Using Column Expressions**
- Grouping and Aggregation Queries
- Joining DataFrames
- Essential Points
- Hands-On Exercise: Analyzing Data with DataFrame Queries

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Columns, Column Names, and Column Expressions

- **Most DataFrame transformations require you to specify a column or columns**
  - `select(column1, column2, …)`
  - `orderBy(column1, column2, …)`
- **For many simple queries, you can just specify the column name as a string**
  - `peopleDF.select("firstName","lastName")`
- **Some types of transformations use *column references* or *column expressions* instead of column name strings**

```
def select(col: String, cols: String*): DataFrame
    Selects a set of columns.

def select(cols: Column*): DataFrame
    Selects a set of column based expressions.
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Column References (Python)

- **In Python, there are two equivalent ways to refer to a column**

```
peopleDF = spark.read.option("header","true").csv("people.csv")

peopleDF['age']
Column<age>

peopleDF.age
Column<age>

peopleDF.select(peopleDF.age).show()
+---+
|age|
+---+
| 52|
| 32|
| 28|
…
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Column References (Scala)

- **In Scala, there are two ways to refer to a column**
  - The first uses the column name with the DataFrame
  - The second uses the column name only, and is not fully resolved until used in a transformation

```
val peopleDF = spark.read.
  option("header","true").csv("people.csv")

peopleDF("age")
org.apache.spark.sql.Column = age

$"age"
org.apache.spark.sql.ColumnName = age

peopleDF.select(peopleDF("age")).show()
+---+
|age|
+---+
| 52|
| 32|
…
```

Language: *Scala*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Column Expressions

- **Using column references instead of simple strings allows you to create *column expressions***
- **Column operations include**
  - Arithmetic operators such as `+`, `-`, `%`, `/`, and `*`
  - Comparative and logical operators such as `>`, `<`, `&&` and `||`
    - The equality comparator is `===` in Scala, and `==` in Python
  - String functions such as `contains`, `like`, and `substr`
  - Data testing functions such as `isNull`, `isNotNull`, and `NaN` (not a number)
  - Sorting functions such as `asc` and `desc`
    - Work only when used in `sort`/`orderBy`
- **For the full list of operators and functions, see the API documentation for `Column`**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Examples: Column Expressions (Python)

```
peopleDF.select("lastName", peopleDF.age * 10).show()
+--------+----------+
|lastName|(age * 10)|
+--------+----------+
|  Hopper|       520|
|  Turing|       320|
…

peopleDF.where(peopleDF.firstName.startswith("A")).show()
+-----+--------+---------+---+
|pcode|lastName|firstName|age|
+-----+--------+---------+---+
|94020|  Turing|     Alan| 32|
|94020|Lovelace|      Ada| 28|
+-----+--------+---------+---+
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Examples: Column Expressions (Scala)

```
peopleDF.select($"lastName", $"age" * 10).show
+--------+----------+
|lastName|(age * 10)|
+--------+----------+
|  Hopper|       520|
|  Turing|       320|
…

peopleDF.where(peopleDF("firstName").startsWith("A")).show
+-----+--------+---------+---+
|pcode|lastName|firstName|age|
+-----+--------+---------+---+
|94020|  Turing|     Alan| 32|
|94020|Lovelace|      Ada| 28|
+-----+--------+---------+---+
```

Language: *Scala*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Column Aliases (1)

- **Use the column `alias` function to rename a column in a result set**
  - `name` is a synonym for `alias`
- **Example (Python): Use column name `age_10` instead of `(age * 10)`**

```
peopleDF.select("lastName",
   (peopleDF.age * 10).alias("age_10")).show()
+--------+------+
|lastName|age_10|
+--------+------+
|  Hopper|   520|
|  Turing|   320|
…
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Column Aliases (2)

- **Example (Scala): Use column name `age_10` instead of `(age * 10)`**

```
 peopleDF.select($"lastName",
   ($"age" * 10).alias("age_10")).show
+--------+------+
|lastName|age_10|
+--------+------+
|  Hopper|   520|
|  Turing|   320|
…
```

Language: *Scala*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Analyzing Data with DataFrame Queries

- Querying DataFrames Using Column Expressions
- **Grouping and Aggregation Queries**
- Joining DataFrames
- Essential Points
- Hands-On Exercise: Analyzing Data with DataFrame Queries

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Aggregation Queries

- Aggregation queries perform a calculation on a set of values and return a single value
- To execute an aggregation on a set of grouped values, use `groupBy` combined with an aggregation function
- Example: How many people are in each postal code?

```
peopleDF.groupBy("pcode").count().show()
+-----+-----+
|pcode|count|
+-----+-----+
|94020|    2|
|87501|    1|
|02134|    2|
+-----+-----+
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

The `groupBy` Transformation

- **`groupBy` takes one or more column names or references**
  - In Scala, returns a `RelationalGroupedDataset` object
  - In Python, returns a `GroupedData` object
- **Returned objects provide aggregation functions, including**
  - `count`
  - `max` and `min`
  - `mean` (and its alias `avg`)
  - `sum`
  - `pivot`
  - `agg` (aggregates using additional aggregation functions)

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Additional Aggregation Functions

- **The `functions` object provides several additional aggregation functions**
- **Aggregate functions include**
  - `first/last` returns the first or last items in a group
  - `countDistinct` returns the number of unique items in a group
  - `approx_count_distinct` returns an approximate counts of unique items
    - Much faster than a full count
  - `stddev` calculates the standard deviation for a group of values
  - `var_sample/var_pop` calculates the variance for a group of values
  - `covar_samp/covar_pop` calculates the sample and population covariance of a group of values
  - `corr` returns the correlation of a group of values

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Using the `functions` object

```
import pyspark.sql.functions as functions

peopleDF.groupBy("pcode").agg(functions.stddev("age")).show()
+-----+------------------+
|pcode|  stddev_samp(age)|
+-----+------------------+
|94020|0.7071067811865476|
|87501|               NaN|
|02134|2.1213203435596424|
+-----+------------------+
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Analyzing Data with DataFrame Queries

- Querying DataFrames Using Column Expressions
- Grouping and Aggregation Queries
- **Joining DataFrames**
- Essential Points
- Hands-On Exercise: Analyzing Data with DataFrame Queries

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Joining DataFrames

- **Use the `join` transformation to join two DataFrames**
- **DataFrames support several types of joins**
  - `inner` (default)
  - `outer`
  - `left_outer`
  - `right_outer`
  - `leftsemi`
- **The `crossJoin` transformation joins every element of one DataFrame with every element of the other**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: A Simple Inner Join (1)

people-no-pcode.csv

```
pcode,lastName,firstName,age
02134,Hopper,Grace,52
,Turing,Alan,32
94020,Lovelace,Ada,28
87501,Babbage,Charles,49
02134,Wirth,Niklaus,48
```

pcodes.csv

```
pcode,city,state
02134,Boston,MA
94020,Palo Alto,CA
87501,Santa Fe,NM
60645,Chicago,IL
```

```
val peopleDF = spark.read.option("header","true").
  csv("people-no-pcode.csv")

val pcodesDF = spark.read.
  option("header","true").csv("pcodes.csv")
```

Language: *Scala*

Example continued on next slide…

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: A Simple Inner Join (2)

- **This example shows an inner join when join column is in both DataFrames**

```
peopleDF.join(pcodesDF, "pcode").show()
+-----+--------+---------+---+---------+-----+
|pcode|lastName|firstName|age|     city|state|
+-----+--------+---------+---+---------+-----+
|02134|  Hopper|    Grace| 52|   Boston|   MA|
|94020|Lovelace|      Ada| 28|Palo Alto|   CA|
|87501| Babbage|  Charles| 49| Santa Fe|   NM|
|02134|   Wirth|  Niklaus| 48|   Boston|   MA|
+-----+--------+---------+---+---------+-----+
…
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: A Left Outer Join (1)

- Specify type of join as `inner` (default), `outer`, `left_outer`, `right_outer`, or `leftsemi`

```
peopleDF.join(pcodesDF, "pcode", "left_outer").show()
+-----+--------+---------+---+---------+-----+
|pcode|lastName|firstName|age|     city|state|
+-----+--------+---------+---+---------+-----+
|02134|  Hopper|    Grace| 52|   Boston|   MA|
| null|  Turing|     Alan| 32|     null| null|
|94020|Lovelace|      Ada| 28|Palo Alto|   CA|
|87501| Babbage|  Charles| 49| Santa Fe|   NM|
|02134|   Wirth|  Niklaus| 48|   Boston|   MA|
+-----+--------+---------+---+---------+-----+
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: A Left Outer Join (2)

- Specify type of join as `inner` (default), `outer`, `left_outer`, `right_outer`, or `leftsemi`

```
peopleDF.join(pcodesDF,
  peopleDF("pcode") === pcodesDF("pcode"),
  "left_outer").show
+-----+--------+---------+---+---------+-----+
|pcode|lastName|firstName|age|     city|state|
+-----+--------+---------+---+---------+-----+
|02134|  Hopper|    Grace| 52|   Boston|   MA|
| null|  Turing|     Alan| 32|     null| null|
|94020|Lovelace|      Ada| 28|Palo Alto|   CA|
|87501| Babbage|  Charles| 49| Santa Fe|   NM|
|02134|   Wirth|  Niklaus| 48|   Boston|   MA|
+-----+--------+---------+---+---------+-----+
```

Language: *Scala*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Joining on Columns with Different Names (1)

people-no-pcode.csv

```
pcode,lastName,firstName,age
02134,Hopper,Grace,52
,Turing,Alan,32
94020,Lovelace,Ada,28
87501,Babbage,Charles,49
02134,Wirth,Niklaus,48
```

zcodes.csv

```
zip,city,state
02134,Boston,MA
94020,Palo Alto,CA
87501,Santa Fe,NM
60645,Chicago,IL
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Joining on Columns with Different Names (2)

- **Use column expressions when the names of the join columns are different**
  - The result includes both of the join columns

```
peopleDF.join(zcodesDF, $"pcode" === $"zip").show
+-----+--------+---------+---+-----+---------+-----+
|pcode|lastName|firstName|age|  zip|     city|state|
+-----+--------+---------+---+-----+---------+-----+
|02134|  Hopper|    Grace| 52|02134|   Boston|   MA|
|94020|Lovelace|      Ada| 28|94020|Palo Alto|   CA|
|87501| Babbage|  Charles| 49|87501| Santa Fe|   NM|
|02134|   Wirth|  Niklaus| 48|02134|   Boston|   MA|
+-----+--------+---------+---+-----+---------+-----+
```

Language: *Scala*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Joining on Columns with Different Names (3)

```
peopleDF.join(zcodesDF, peopleDF.pcode == zcodesDF.zip).show()
+-----+--------+---------+---+-----+---------+-----+
|pcode|lastName|firstName|age|  zip|     city|state|
+-----+--------+---------+---+-----+---------+-----+
|02134|  Hopper|    Grace| 52|02134|   Boston|   MA|
|94020|Lovelace|      Ada| 28|94020|Palo Alto|   CA|
|87501| Babbage|  Charles| 49|87501| Santa Fe|   NM|
|02134|   Wirth|  Niklaus| 48|02134|   Boston|   MA|
+-----+--------+---------+---+-----+---------+-----+
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Analyzing Data with DataFrame Queries

- Querying DataFrames Using Column Expressions
- Grouping and Aggregation Queries
- Joining DataFrames
- **Essential Points**
- Hands-On Exercise: Analyzing Data with DataFrame Queries

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **Columns in a DataFrame can be specified by name or `Column` objects**
  - You can define column expressions using column operators
- **Calculate aggregate values on groups of rows using `groupBy` and an aggregation function**
- **Use the `join` operation to join two DataFrames**
  - Supports inner, left outer, right outer and semi joins

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Analyzing Data with DataFrame Queries

- Querying DataFrames Using Column Expressions
- Grouping and Aggregation Queries
- Joining DataFrames
- Essential Points
- **Hands-On Exercise: Analyzing Data with DataFrame Queries**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Analyzing Data with DataFrame Queries

- **In this exercise, you will analyze different sets of data using DataFrame queries**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

RDD Overview

Chapter 8

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- **RDD Overview**
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

RDD Overview

In this chapter, you will learn

- **What RDDs are**
- **How RDDs compare with DataFrames and Datasets**
- **How to load and save RDDs with a variety of data source types**
- **How to transform RDD data and return query results**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

RDD Overview

- **RDD Overview**
- RDD Data Sources
- Creating and Saving RDDs
- RDD Operations
- Essential Points
- Hands-On Exercise: Working With RDDs

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Resilient Distributed Datasets (RDDs)

- **RDDs are part of core Spark**
- ***Resilient Distributed Dataset* (RDD)**
  - *Resilient*: If data in memory is lost, it can be recreated
  - *Distributed*: Processed across the cluster
  - *Dataset*: Initial data can come from a source such as a file, or it can be created programmatically
- **Despite the name, RDDs are *not* Spark SQL `Dataset` objects**
  - RDDs predate Spark SQL and the DataFrame/Dataset API

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Comparing RDDs to DataFrames and Datasets (1)

- **RDDs are unstructured**
  - No schema defining columns and rows
  - Not table-like; cannot be queried using SQL-like transformations such as `where` and `select`
  - RDD transformations use lambda functions
- **RDDs can contain any type of object**
  - DataFrames are limited to **Row** objects
  - Datasets are limited to **Row** objects, case class objects (products), and primitive types

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Comparing RDDs to DataFrames and Datasets (2)

- **RDDs are used in all Spark languages (Python, Scala, Java)**
  - Strongly-typed in Scala and Java like Datasets
- **RDDs are not optimized by the Catalyst optimizer**
  - Manually coded RDDs are typically less efficient than DataFrames
- **You can use RDDs to create DataFrames and Datasets**
  - RDDs are often used to convert unstructured or semi-structured data into structured form
  - You can also work directly with the RDDs that underlie DataFrames and Datasets

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

RDD Overview

- RDD Overview
- **RDD Data Sources**
- Creating and Saving RDDs
- RDD Operations
- Essential Points
- Hands-On Exercise: Working With RDDs

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

RDD Data Types

- **RDDs can hold any serializable type of element**
  - Primitive types such as integers, characters, and booleans
  - Collections such as strings, lists, arrays, tuples, and dictionaries (including nested collection types)
  - Scala/Java Objects (if serializable)
  - Mixed types
- **Some RDDs are specialized and have additional functionality**
  - Pair RDDs
    - RDDs consisting of key-value pairs
  - Double RDDs
    - RDDs consisting of numeric data

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

RDD Data Sources

- **There are several types of data sources for RDDs**
  - Files, including text files and other formats
  - Data in memory
  - Other RDDs
  - Datasets or DataFrames

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Creating RDDs from Files

- **Use `SparkContext` object, not `SparkSession`**
  - `SparkContext` is part of the core Spark library
  - `SparkSession` is part of the Spark SQL library
  - One Spark context per application
  - Use `SparkSession.sparkContext` to access the Spark context
    - Called `sc` in the Spark shell
- **Create file-based RDDs using the Spark context**
  - Use `textFile` or `wholeTextFiles` to read text files
  - Use `hadoopFile` or `newAPIHadoopFile` to read other formats
    - The Hadoop "new API" was introduced in Hadoop .20
    - Spark supports both for backward compatibility

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

RDD Overview

- RDD Overview
- RDD Data Sources
- **Creating and Saving RDDs**
- RDD Operations
- Essential Points
- Hands-On Exercise: Working With RDDs

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Creating RDDs from Text Files (1)

- **`SparkContext.textFile` reads newline-terminated text files**
  - Accepts a single file, a directory of files, a wildcard list of files, or a comma-separated list of files
  - Examples
    - `textFile("myfile.txt")`
    - `textFile("mydata/")`
    - `textFile("mydata/*.log")`
    - `textFile("myfile1.txt,myfile2.txt")`

```
myRDD = spark.sparkContext.textFile("mydata/")
```

Language: *Python*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Creating RDDs from Text Files (2)

- **`textFile` maps each line in a file to a separate RDD element**
  - Only supports newline-terminated text

```
myRDD = spark. \
  sparkContext. \
  textFile("purplecow.txt")
```

Language: *Python*

```
I've never seen a purple cow.\n
I never hope to see one;\n
But I can tell you, anyhow,\n
I'd rather see than be one.\n
```

myRDD

I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Multi-line Text Elements

- **`textFile` maps each line in a file to a separate RDD element**
  - What about files with a multi-line input format, such as XML or JSON?
- **Use `wholeTextFiles`**
  - Maps entire contents of each file in a directory to a single RDD element
  - Works only for small files (element must fit in memory)

`user1.json`

```
{
 "firstName":"Fred",
 "lastName":"Flintstone",
 "userid":"123"
}
```

`user2.json`

```
{
 "firstName":"Barney",
 "lastName":"Rubble",
 "userid":"234"
}
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Using `wholeTextFiles`

```
userRDD = spark.sparkContext. \
  wholeTextFiles("userFiles")
```

Language: *Python*

`userRDD`

("user1.json",{"firstName":"Fred", "lastName":"Flintstone","userid":"123"})
("user2.json",{"firstName":"Barney", "lastName":"Rubble","userid":"234"})
("user3.json",…)
("user4.json",…)

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Creating RDDs from Collections

- **You can create RDDs from collections instead of files**
  - `SparkContext.parallelize(collection)`
- **Useful when**
  - Testing
  - Generating data programmatically
  - Integrating with other systems or libraries
  - Learning

```
myData = ["Alice","Carlos","Frank","Barbara"]
myRDD = sc.parallelize(myData)
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Saving RDDs

- **You can save RDDs to the same data source types supported for reading RDDs**
  - Use `RDD.saveAsTextFile` to save as plain text files in the specified directory
  - Use `RDD.saveAsHadoopFile` or `saveAsNewAPIHadoopFile` with a specified Hadoop `OutputFormat` to save using other formats
- **The specified save directory cannot already exist**

```
myRDD.saveAsTextFile("mydata/")
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

RDD Overview

- RDD Overview
- RDD Data Sources
- Creating and Saving RDDs
- **RDD Operations**
- Essential Points
- Hands-On Exercise: Working With RDDs

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

RDD Operations

- **Two general types of RDD operations**
  - *Actions* return a value to the Spark driver or save data to a data source
  - *Transformations* define a new RDD based on the current one(s)
- **RDDs operations are performed lazily**
  - Actions trigger execution of the base RDD transformations

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

RDD Action Operations

- **Some common actions**
  - `count` returns the number of elements
  - `first` returns the first element
  - `take(n)` returns an array (Scala) or list (Python) of the first *n* elements
  - `collect` returns an array (Scala) or list (Python) of all elements
  - `saveAsTextFile(dir)` saves to text files

```
myRDD = sc. \
  textFile("purplecow.txt")

for line in myRDD.take(2):
  print line
I've never seen a purple cow.
I never hope to see one;
```

Language: *Python*

```
val myRDD = sc.
  textFile("purplecow.txt")

for (line <- myRDD.take(2))
  println(line)
I've never seen a purple cow.
I never hope to see one;
```

Language: *Scala*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

RDD Transformation Operations (1)

- **Transformations create a new RDD from an existing one**
- **RDDs are immutable**
  - Data in an RDD is never changed
  - Transform data to create a new RDD
- **A transformation operation executes a *transformation function***
  - The function transforms elements of an RDD into new elements
  - Some transformations implement their own transformation logic
  - For many, you must provide the function to perform the transformation

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

RDD Transformation Operations (2)

- **Transformation operations include**
  - `distinct` creates a new RDD with duplicate elements in the base RDD removed
  - `union(`*`rdd`*`)` creates a new RDD by appending the data in one RDD to another
  - `map(`*`function`*`)` creates a new RDD by performing a function on each record in the base RDD
  - `filter(`*`function`*`)` creates a new RDD by including or excluding each record in the base RDD according to a Boolean function

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: `distinct` and `union` Transformations

cities1.csv

```
Boston,MA
Palo Alto,CA
Santa Fe,NM
Palo Alto,CA
```

cities2.csv

```
Calgary,AB
Chicago,IL
Palo Alto,CA
```

```
distinctRDD = sc.\
  textFile("cities1.csv").distinct()
for city in distinctRDD.collect(): \
  print city
Boston,MA
Palo Alto,CA
Santa Fe,NM

unionRDD = sc.textFile("cities2.csv"). \
  union(distinctRDD)
for city in unionRDD.collect():
  print city
Calgary,AB
Chicago,IL
Palo Alto,CA
Boston,MA
Palo Alto,CA
Santa Fe,NM
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

RDD Overview

- RDD Overview
- RDD Data Sources
- Creating and Saving RDDs
- RDD Operations
- **Essential Points**
- Hands-On Exercise: Working With RDDs

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **RDDs (Resilient Distributed Datasets) are a key concept in Spark**
  - Represent a distributed collection of elements
  - Elements can be of any type
- **RDDs are created from data sources**
  - Text files and other data file formats
  - Data in other RDDs
  - Data in memory
  - DataFrames and Datasets
- **RDDs contain unstructured data**
  - No associated schema like DataFrames and Datasets
- **RDD Operations**
  - Transformations create a new RDD based on an existing one
  - Actions return a value from an RDD

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

RDD Overview

- RDD Overview
- RDD Data Sources
- Creating and Saving RDDs
- RDD Operations
- Essential Points
- **Hands-On Exercise: Working With RDDs**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Working with RDDs

- **In this exercise, you will load, transform, display, and save data using RDDs**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

cloudera®

Transforming Data with RDDs

Chapter 9

cloudera®

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- **Transforming Data with RDDs**
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Transforming Data with RDDs

In this chapter, you will learn

- **How to use functional programming with RDD transformations**
- **How RDD transformations are executed**
- **How to create DataFrames from RDD data**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Transforming Data with RDDs

- **Writing and Passing Transformation Functions**
- Transformation Execution
- Converting Between RDDs and DataFrames
- Essential Points
- Hands-On Exercise: Transforming Data Using RDDs

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Functional Programming in Spark

- **Key concepts of *functional programming***
  - Functions are the fundamental unit of programming
  - Functions have input and output only
    - No state or side effects
  - Functions can be passed as arguments to other functions
    - Called *procedural parameters*
- **Spark's architecture is based on functional programming**
  - Passed functions can be executed by multiple executors in parallel

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

RDD Transformation Procedures

- **RDD transformations execute a *transformation procedure***
  - Transforms elements of an RDD into new elements
  - Runs on executors
- **A few transformation operations implement their own transformation logic**
  - Examples: `distinct` and `union`
- **Most transformation operations require you to pass a function**
  - Function implements your own transformation procedure
  - Examples: `map` and `filter`
- **This is a key difference between RDDs and DataFrame/Datasets**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Passing Functions

- **Passed functions can be named or anonymous**
- **Anonymous functions are defined inline without an identifier**
  - Best for short, one-off functions
  - Supported in many programming languages
    - Python: `lambda x: …`
    - Scala: `x => …`
    - Java 8: `x -> …`

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Passing Named Functions (Python)

```
def toUpper(s):
  return s.upper()

myRDD = sc. \
  textFile("purplecow.txt")

myUpperRDD = myRDD.map(toUpper)

for line in myUpperRDD.take(2):
  print line
I'VE NEVER SEEN A PURPLE COW.
I NEVER HOPE TO SEE ONE;
```

myRDD

I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.

↓

myUpperRDD

I'VE NEVER SEEN A PURPLE COW.
I NEVER HOPE TO SEE ONE;
BUT I CAN TELL YOU, ANYHOW,
I'D RATHER SEE THAN BE ONE.

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Passing Named Functions (Scala)

```
def toUpper(s: String):
  String = { s.toUpperCase }

val myRDD =
 sc.textFile("purplecow.txt")

val myUpperRDD =
  myRDD.map(toUpper)

myUpperRDD.take(2).
  foreach(println)
I'VE NEVER SEEN A PURPLE COW.
I NEVER HOPE TO SEE ONE;
```

myRDD

I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.

↓

myUpperRDD

I'VE NEVER SEEN A PURPLE COW.
I NEVER HOPE TO SEE ONE;
BUT I CAN TELL YOU, ANYHOW,
I'D RATHER SEE THAN BE ONE.

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Passing Anonymous Functions

- **Python: Use the `lambda` keyword to specify the name of the input parameter(s) and the function that returns the output**

```
myUpperRDD = myRDD.map(lambda line: line.upper())
```

Language: *Python*

- **Scala: Use the => operator to specify the name of the input parameter(s) and the function that returns the output**

```
val myUpperRDD = myRDD.map(line => line.toUpperCase)
```

Language: *Scala*

- **Scala shortcut: Use underscore (_) to stand for anonymous input parameters**

```
val myUpperRDD = myRDD.map(_.toUpperCase)
```

Language: *Scala*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: map and filter Transformations

```python
myFilteredRDD = myRDD. \
  map(lambda line: line.upper()). \
  filter(lambda line: \
    line.startswith('I'))
```

Language: *Python*

```scala
val myFilteredRDD = myRDD.
  map(line => line.toUpperCase).
  filter(line =>
    line.startsWith("I"))
```

Language: *Scala*

↓

I'VE NEVER SEEN A PURPLE COW.
I NEVER HOPE TO SEE ONE;
BUT I CAN TELL YOU, ANYHOW,
I'D RATHER SEE THAN BE ONE.

↓

myFilteredRDD

I'VE NEVER SEEN A PURPLE COW.
I NEVER HOPE TO SEE ONE;
I'D RATHER SEE THAN BE ONE.

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Transforming Data with RDDs

- Writing and Passing Transformation Functions
- **Transformation Execution**
- Converting Between RDDs and DataFrames
- Essential Points
- Hands-On Exercise: Transforming Data Using RDDs

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

RDD Execution

- **An RDD query consists of a sequence of one or more transformations completed by an action**
- **RDD queries are executed *lazily***
  - When the action is called
- **RDD queries are executed differently than DataFrame and Dataset queries**
  - DataFrames and Datasets scan their sources to determine the schema *eagerly* (when created)
  - RDDs do not have schemas and do not scan their sources before loading

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

RDD Lineage

- **Transformations create a new RDD based on one or more existing RDDs**
  - Result RDDs are considered *children* of the base (*parent)* RDD
  - Child RDDs depend on their parent RDD
- **An RDD's *lineage* is the sequence of ancestor RDDs that it depends on**
  - When an RDD executes, it executes its lineage starting from the source
- **Spark maintains each RDD's lineage**
  - Use `toDebugString` to view the lineage

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

RDD Lineage and toDebugString (Scala)

```
val myFilteredRDD = sc.
  textFile("purplecow.txt").
  map(line => line.toUpperCase()).
  filter(line =>
    line.startsWith("I"))

myFilteredRDD.toDebugString
(2) MapPartitionsRDD[7] at filter …
 |  MapPartitionsRDD[6] at map …
 |  purplecow.txt
    MapPartitionsRDD[5]
    at textFile …
 |  purplecow.txt HadoopRDD[4]
    at textFile …
```

cloudera®
Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

RDD Lineage and `toDebugString` (Python)

- `toDebugString` output is not displayed as nicely in Python

```
myFilteredRDD.toDebugString()
(2) PythonRDD[7] at RDD at PythonRDD.scala:48 []\n |
 purplecow.txt MapPartitionsRDD[6] … []\n |  purplecow.txt
 HadoopRDD[5] at textFile … []
```

- Use `print` for prettier output

```
print myFilteredRDD.toDebugString()
(2) PythonRDD[7] at RDD at PythonRDD.scala:48 []
 |  purplecow.txt MapPartitionsRDD[6] at textFile …
 |  purplecow.txt HadoopRDD[5] at textFile …
```

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Pipelining (1)

- **When possible, Spark will perform sequences of transformations by element so no data is stored**

```
val myFilteredRDD = sc.
  textFile("purplecow.txt").
  map(line => line.toUpperCase).
  filter(line =>
    line.startsWith("I"))
myFilteredRDD.take(2)
```

Language: *Scala*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Pipelining (2)

- **When possible, Spark will perform sequences of transformations by element so no data is stored**

```
val myFilteredRDD = sc.
  textFile("purplecow.txt").
  map(line => line.toUpperCase).
  filter(line =>
    line.startsWith("I"))
myFilteredRDD.take(2)
```

Language: *Scala*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Pipelining (3)

- **When possible, Spark will perform sequences of transformations by element so no data is stored**

```
val myFilteredRDD = sc.
  textFile("purplecow.txt").
  map(line => line.toUpperCase).
  filter(line =>
    line.startsWith("I"))
myFilteredRDD.take(2)
```

Language: *Scala*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Pipelining (4)

- **When possible, Spark will perform sequences of transformations by element so no data is stored**

```
val myFilteredRDD = sc.
  textFile("purplecow.txt").
  map(line => line.toUpperCase).
  filter(line =>
    line.startsWith("I"))
myFilteredRDD.take(2)
I'VE NEVER SEEN A PURPLE COW.
```

Language: *Scala*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Pipelining (5)

- **When possible, Spark will perform sequences of transformations by element so no data is stored**

```
val myFilteredRDD = sc.
  textFile("purplecow.txt").
  map(line => line.toUpperCase).
  filter(line =>
    line.startsWith("I"))
myFilteredRDD.take(2)
I'VE NEVER SEEN A PURPLE COW.
```

Language: *Scala*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Pipelining (6)

- **When possible, Spark will perform sequences of transformations by element so no data is stored**

```
val myFilteredRDD = sc.
  textFile("purplecow.txt").
  map(line => line.toUpperCase).
  filter(line =>
    line.startsWith("I"))
myFilteredRDD.take(2)
I'VE NEVER SEEN A PURPLE COW.
```

Language: *Scala*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Pipelining (7)

- **When possible, Spark will perform sequences of transformations by element so no data is stored**

```
val myFilteredRDD = sc.
  textFile("purplecow.txt").
  map(line => line.toUpperCase).
  filter(line =>
    line.startsWith("I"))
myFilteredRDD.take(2)
I'VE NEVER SEEN A PURPLE COW.
```

Language: *Scala*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Pipelining (8)

- **When possible, Spark will perform sequences of transformations by element so no data is stored**

```
val myFilteredRDD = sc.
  textFile("purplecow.txt").
  map(line => line.toUpperCase).
  filter(line =>
    line.startsWith("I"))
myFilteredRDD.take(2)
I'VE NEVER SEEN A PURPLE COW.
I NEVER HOPE TO SEE ONE;
```

Language: *Scala*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Transforming Data with RDDs

- Writing and Passing Transformation Functions
- Transformation Execution
- **Converting Between RDDs and DataFrames**
- Essential Points
- Hands-On Exercise: Transforming Data Using RDDs

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Converting RDDs to DataFrames

- **You can create a DataFrame from an RDD**
  - Useful with unstructured or semi-structured data such as text
  - Define a schema
  - Transform the base RDD to an RDD of `Row` objects (Scala) or lists (Python)
  - Use `SparkSession.createDataFrame`
- **You can also return the underlying RDD of a DataFrame**
  - Use the `DataFrame.rdd` attribute to return an RDD of `Row` objects

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Create a DataFrame from an RDD

- **Example data: semi-structured text data source**

```
02134,Hopper,Grace,52
94020,Turing,Alan,32
94020,Lovelace,Ada,28
87501,Babbage,Charles,49
02134,Wirth,Niklaus,48
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Scala Example: Create a DataFrame from an RDD (1)

```
import org.apache.spark.sql.types._
import org.apache.spark.sql.Row

val mySchema = StructType(Array(
  StructField("pcode", StringType),
  StructField("lastName", StringType),
  StructField("firstName", StringType),
  StructField("age", IntegerType)
  ))
```

Language: *Scala*

Continued on next slide...

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Scala Example: Create a DataFrame from an RDD (2)

```
val rowRDD = sc.textFile("people.txt").
  map(line => line.split(",")).
  map(values =>
      Row(values(0),values(1),values(2),values(3).toInt))

val myDF = spark.createDataFrame(rowRDD,mySchema)

myDF.show(2)
+-----+--------+---------+---+
|pcode|lastName|firstName|age|
+-----+--------+---------+---+
|02134|  Hopper|    Grace| 52|
|94020|  Turing|     Alan| 32|
+-----+--------+---------+---+
```

Language: *Scala*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Python Example: Create a DataFrame from an RDD (1)

```
from pyspark.sql.types import *

mySchema = \
  StructType([
              StructField("pcode",StringType()),
              StructField("lastName",StringType()),
              StructField("firstName",StringType()),
              StructField("age",IntegerType())])
```

Language: *Python*

Continued on next slide...

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Python Example: Create a DataFrame from an RDD (2)

```
myRDD = sc.textFile("people.txt"). \
  map(lambda line: line.split(",")). \
  map(lambda values:
      [values[0],values[1],values[2],int(values[3])])

myDF = spark.createDataFrame(myRDD,mySchema)

myDF.show(2)
+-----+--------+---------+---+
|pcode|lastName|firstName|age|
+-----+--------+---------+---+
|02134|  Hopper|    Grace| 52|
|94020|  Turing|     Alan| 32|
+-----+--------+---------+---+
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Return a DataFrame's Underlying RDD

```
myRDD2 = myDF.rdd

for row in myRDD2.take(2): print row
Row(pcode=u'02134', lastName=u'Hopper', firstName=u'Grace',
 age=52)
Row(pcode=u'94020', lastName=u'Turing', firstName=u'Alan',
 age=32)
```

Language: *Python*

```
val myRDD2 = myDF.rdd

myRDD2.take(2).foreach(println)
[02134,Hopper,Grace,52]
[94020,Turing,Alan,32]
```

Language: *Scala*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Transforming Data with RDDs

- Writing and Passing Transformation Functions
- Transformation Execution
- Converting Between RDDs and DataFrames
- **Essential Points**
- Hands-On Exercise: Transforming Data Using RDDs

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **RDDs (Resilient Distributed Datasets) are a key concept in Spark**
- **RDD Operations**
  - Transformations create a new RDD based on an existing one
  - Actions return a value from an RDD
- **RDD Transformations use functional programming**
  - Take named or anonymous functions as parameters
- **RDD query execution is *lazy*—transformation are not executed until triggered by an action**
- **Operations on the same RDD element are pipelined together if possible**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Transforming Data with RDDs

- Writing and Passing Transformation Functions
- Transformation Execution
- Converting Between RDDs and DataFrames
- Essential Points
- **Hands-On Exercise: Transforming Data Using RDDs**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Transforming Data Using RDDs

- **In this exercise, you will transform, display, and save data using RDDs**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

cloudera®

Aggregating Data with Pair RDDs

Chapter 10

cloudera®

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- **Aggregating Data with Pair RDDs**
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Aggregating Data with Pair RDDs

In this chapter, you will learn

- **How to create pair RDDs of key-value pairs from generic RDDs**
- **What special operations are available on pair RDDs**
- **How map-reduce algorithms are implemented in Spark**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Aggregating Data with Pair RDDs

- **Key-Value Pair RDDs**
- Map-Reduce
- Other Pair RDD Operations
- Essential Points
- Hands-On Exercise: Joining Data Using Pair RDDs

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Pair RDDs

Pair RDD

- **Pair RDDs are a special form of RDD**
  - Each element must be a key/value pair (a two-element *tuple*)
  - Keys and values can be any type
- **Why?**
  - Use with map-reduce algorithms
  - Many additional functions are available for common data processing needs
    - Such as sorting, joining, grouping, and counting

(key1,value1)
(key2,value2)
(key3,value3)
…

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Creating Pair RDDs

- **The first step in most workflows is to get the data into key/value form**
  - What should the RDD should be keyed on?
  - What is the value?
- **Commonly used functions to create pair RDDs**
  - `map`
  - `flatMap/flatMapValues`
  - `keyBy`

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: A Simple Pair RDD (Python)

- **Example: Create a pair RDD from a tab-separated file**

```
usersRDD = sc.textFile("userlist.tsv"). \
  map(lambda line: line.split('\t')). \
  map(lambda fields: (fields[0],fields[1]))
```

```
user001\tFred Flintstone
user090\tBugs Bunny
user111\tHarry Potter
…
```

→

("user001","Fred Flintstone")
("user090","Bugs Bunny")
("user111","Harry Potter")
…

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: A Simple Pair RDD (Scala)

- **Example: Create a pair RDD from a tab-separated file**

```
val usersRDD = sc.textFile("userlist.tsv").
  map(line => line.split('\t').
  map(fields => (fields(0),fields(1)))
```

```
user001\tFred Flintstone
user090\tBugs Bunny
user111\tHarry Potter
…
```

→

("user001","Fred Flintstone")
("user090","Bugs Bunny")
("user111","Harry Potter")
…

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Keying Web Logs by User ID (Python)

```
sc.textFile("weblogs/"). \
  keyBy(lambda line: line.split(' ')[2])
```

```
56.38.234.188 - 99788 "GET /KBDOC-00157.html http/1.0" …
56.38.234.188 - 99788 "GET /theme.css http/1.0" …
203.146.17.59 - 25254 "GET /KBDOC-00230.html http/1.0" …
…
```

↓

(99788,56.38.234.188 - 99788 "GET /KBDOC-00157.html…)
(99788,56.38.234.188 - 99788 "GET /theme.css…)
(25254,203.146.17.59 - 25254 "GET /KBDOC-00230.html…)
…

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Question 1: Pairs with Complex Values

- **How would you do this?**
  - Input: a tab-delimited list of postal codes with latitude and longitude
  - Output: postal code (key) and lat/long pair (value)

```
00210\t43.00589\t-71.01320
01014\t42.17073\t-72.60484
01062\t42.32423\t-72.67915
01263\t42.3929\t-73.22848
…
```

→

```
("00210",(43.00589,-71.01320))
("01014",(42.17073,-72.60484))
("01062",(42.32423,-72.67915))
("01263",(42.3929,-73.22848))
…
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Answer 1: Pairs with Complex Values

```
sc.textFile("latlon.tsv"). \
  map(lambda line: line.split('\t')). \
  map(lambda fields:
    (fields[0],(float(fields[1]),float(fields[2]))))
```

Language: *Python*

```
00210\t43.00589\t-71.01320
01014\t42.17073\t-72.60484
01062\t42.32423\t-72.67915
01263\t42.3929\t-73.22848
…
```

→

```
("00210",(43.00589,-71.01320))
("01014",(42.17073,-72.60484))
("01062",(42.32423,-72.67915))
("01263",(42.3929,-73.22848))
…
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Question 2: Mapping Single Elements to Multiple Pairs

- **How would you do this?**
  - Input: order numbers with a list of SKUs in the order
  - Output: order (key) and sku (value)

```
00001 sku010:sku933:sku022
00002 sku912:sku331
00003 sku888:sku022:sku010:sku594
00004 sku411
```

→

("00001","sku010")
("00001","sku933")
("00001","sku022")
("00002","sku912")
("00002","sku331")
("00003","sku888")
…

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Answer 2: Mapping Single Elements to Multiple Pairs (1)

```
val ordersRDD = sc.textFile("orderskus.txt")
```

Language: *Scala*

"00001 sku010:sku933:sku022"
"00002 sku912:sku331"
"00003 sku888:sku022:sku010:sku594"
"00004 sku411"

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Answer 2: Mapping Single Elements to Multiple Pairs (2)

```
val ordersRDD = sc.textFile("orderskus.txt").
  map(line => line.split(' '))
```

Language: *Scala*

`Array("00001","sku010:sku933:sku022")`
`Array("00002","sku912:sku331")`
`Array("00003","sku888:sku022:sku010:sku594")`
`Array("00004","sku411")`

`split` returns two-element arrays

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Answer 2: Mapping Single Elements to Multiple Pairs (3)

```
val ordersRDD = sc.textFile("orderskus.txt").
  map(line => line.split(' ')).
  map(fields => (fields(0),fields(1)))
```

Language: *Scala*

("00001","sku010:sku933:sku022")
("00002","sku912:sku331")
("00003","sku888:sku022:sku010:sku594")
("00004","sku411")

Map array elements to tuples to produce a pair RDD

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Answer 2: Mapping Single Elements to Multiple Pairs (4)

```
val ordersRDD = sc.textFile("orderskus.txt").
  map(line => line.split(' ')).
  map(fields => (fields(0),fields(1))).
  flatMapValues(skus => skus.split(':'))
```

Language: *Scala*

("00001","sku010")
("00001","sku933")
("00001","sku022")
("00002","sku912")
("00002","sku331")
("00003","sku888")
…

`flatMapValues` splits a single value (a colon-separated string of SKUs) into multiple elements

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Aggregating Data with Pair RDDs

- Key-Value Pair RDDs
- **Map-Reduce**
- Other Pair RDD Operations
- Essential Points
- Hands-On Exercise: Joining Data Using Pair RDDs

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Map-Reduce

- **Map-reduce is a common programming model**
  - Easily applicable to distributed processing of large data sets
- **Hadoop MapReduce was the first major distributed implementation**
  - Somewhat limited
  - Each job has one map phase, one reduce phase
  - Job output is saved to files
- **Spark implements map-reduce with much greater flexibility**
  - Map and reduce functions can be interspersed
  - Results can be stored in memory
    - Operations can easily be chained

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Map-Reduce in Spark

- **Map-reduce in Spark works on pair RDDs**
- **Map phase**
  - Operates on one record at a time
  - "Maps" each record to zero or more new records
  - Examples: `map`, `flatMap`, `filter`, `keyBy`
- **Reduce phase**
  - Works on map output
  - Consolidates multiple records
  - Examples: `reduceByKey`, `sortByKey`, `mean`

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Map-Reduce Example: Word Count

Input Data

```
the cat sat on the mat
the aardvark sat on the sofa
```

?

→

Result

(on, 2)
(sofa, 1)
(mat, 1)
(aardvark, 1)
(the, 4)
(cat, 1)
(sat, 2)

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Word Count (1)

```
countsRDD = sc.textFile("catsat.txt"). \
  flatMap(lambda line: line.split(' '))
```

Language: *Python*

the
cat
sat
on
the
mat
the
aardvark
…

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Word Count (2)

```
countsRDD = sc.textFile("catsat.txt"). \
  flatMap(lambda line: line.split(' ')). \
  map(lambda word: (word,1))
```

Language: *Python*

the
cat
sat
on
the
mat
the
aardvark
…

(the, 1)
(cat, 1)
(sat, 1)
(on, 1)
(the, 1)
(mat, 1)
(the, 1
(aardvark, 1)
…

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Word Count (3)

```
countsRDD = sc.textFile("catsat.txt"). \
  flatMap(lambda line: line.split(' ')). \
  map(lambda word: (word,1)). \
  reduceByKey(lambda v1,v2: v1+v2)
```

Language: *Python*

the
cat
sat
on
the
mat
the
aardvark
…

(the, 1)
(cat, 1)
(sat, 1)
(on, 1)
(the, 1)
(mat, 1)
(the, 1
(aardvark, 1)
…

(on, 2)
(sofa, 1)
(mat, 1)
(aardvark, 1)
(the, 4)
(cat, 1)
(sat, 2)

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

reduceByKey (1)

- **The function passed to `reduceByKey` combines values from two keys**
  - Function must be binary

```
reduceByKey(lambda v1,v2: v1+v2)
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

reduceByKey (2)

- **The function might be called in any order, therefore must be**
  - Commutative: $x + y = y + x$
  - Associative: $(x + y) + z = x + (y + z)$

```
reduceByKey(lambda v1,v2: v1+v2)
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Word Count Recap (The Scala Version)

```
val counts = sc.textFile("catsat.txt").
  flatMap(line => line.split(' ')).
  map(word => (word,1)).
  reduceByKey((v1,v2) => v1+v2)
```

OR

```
val counts = sc.textFile("catsat.txt").
  flatMap(_.split(' ')).
  map((_,1)).
  reduceByKey(_+_)
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Why Do We Care about Counting Words?

- **Word count is challenging with massive amounts of data**
  - Using a single compute node would be too time-consuming
- **Statistics are often simple aggregate functions**
  - Distributive in nature
  - For example: `max`, `min`, `sum`, and `count`
- **Map-reduce breaks complex tasks down into smaller elements which can be executed in parallel**
  - RDD transformations are implemented using the map-reduce paradigm
- **Many common tasks are very similar to word count**
  - Such as log file analysis

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Aggregating Data with Pair RDDs

- Key-Value Pair RDDs
- Map-Reduce
- **Other Pair RDD Operations**
- Essential Points
- Hands-On Exercise: Joining Data Using Pair RDDs

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Pair RDD Operations

- **In addition to `map` and `reduceByKey` operations, Spark has several operations specific to pair RDDs**
- **Examples**
  - `countByKey` returns a map with the count of occurrences of each key
  - `groupByKey` groups all the values for each key in an RDD
  - `sortByKey` sorts in ascending or descending order
  - `join` returns an RDD containing all pairs with matching keys from two RDDs
  - `leftOuterJoin`, `rightOuterJoin` , `fullOuterJoin` join two RDDs, including keys defined in the left, right, or both RDDs respectively
  - `mapValues`, `flatMapValues` execute a function on just the values, keeping the key the same
  - `lookup(`*`key`*`)` returns the value(s) for a key as a list

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: sortByKey Transformation

```
ordersRDD.sortByKey(ascending=false)
```

(00001,sku010)
(00001,sku933)
(00001,sku022)
(00002,sku912)
(00002,sku331)
(00003,sku888)
…

→

(00004,sku411)
(00003,sku888)
(00003,sku022)
(00003,sku010)
(00003,sku594)
(00002,sku912)
…

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: groupByKey Transformation

```
ordersRDD.groupByKey()
```

(00001,sku010)
(00001,sku933)
(00001,sku022)
(00002,sku912)
(00002,sku331)
(00003,sku888)
…

→

(00002,[sku912,sku331])
(00001,[sku022,sku010,sku933])
(00003,[sku888,sku022,sku010,sku594])
(00004,[sku411])

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Joining by Key

movieGrossRDD

(Casablanca,$3.7M)
(Star Wars,$775M)
(Annie Hall,$38M)
(Argo,$232M)
…

movieYearRDD

(Casablanca,1942)
(Star Wars,1977)
(Annie Hall,1977)
(Argo,2012)
…

```
joinedRDD = movieGrossRDD.
  join(movieYearRDD)
```

→

joinedRDD

(Casablanca,($3.7M,1942))
(Star Wars,($775M,1977))
(Annie Hall,($38M,1977))
(Argo,($232M,2012))
…

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Aggregating Data with Pair RDDs

- Key-Value Pair RDDs
- Map-Reduce
- Other Pair RDD Operations
- **Essential Points**
- Hands-On Exercise: Joining Data Using Pair RDDs

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **Pair RDDs are a special form of RDD consisting of key-value pairs (tuples)**
- **Map-reduce is a generic programming model for distributed processing**
  - Spark implements map-reduce with pair RDDs
  - Hadoop MapReduce and other implementations are limited to a single map and single reduce phase per job
  - Spark allows flexible chaining of map and reduce operations
- **Spark provides several operations for working with pair RDDs**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Aggregating Data with Pair RDDs

- Key-Value Pair RDDs
- Map-Reduce
- Other Pair RDD Operations
- Essential Points
- **Hands-On Exercise: Joining Data Using Pair RDDs**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Joining Data Using Pair RDDs

- **In this exercise, you will join account data with web log data**
  - Please refer to the Hands-On Exercise Manual for instructions

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

cloudera®

Querying Tables and Views with Apache Spark SQL

Chapter 11

cloudera®

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- **Querying Tables and Views with Apache Spark SQL**
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Objectives

In this chapter, you will learn

- **How to create DataFrames based on SQL queries**
- **How to query and manage tables and views using the Spark SQL and the Catalog API**
- **How Spark SQL compares to other SQL-on-Hadoop tools**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Querying Tables and Views with Apache Spark SQL

- **Querying Tables in Spark Using SQL**
- Querying Files and Views
- The Catalog API
- Comparing Spark SQL, Apache Impala, and Apache Hive-on-Spark
- Essential Points
- Hands-On Exercise: Querying Tables and Views with SQL

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark SQL Queries

- **You can query data in Spark SQL using SQL commands**
  - Similar to queries in a relational database or Hive/Impala
  - Spark SQL includes a native SQL 2003 parser
- **You can query Hive tables or DataFrame/Dataset views**
- **Spark SQL queries are particularly useful for**
  - Developers or analysts who are comfortable with SQL
  - Doing ad hoc analysis
- **Use the `SparkSession.sql` function to execute a SQL query on a table**
  - Returns a DataFrame

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Spark SQL Query (1)

- **For Spark installations integrated with Hive, Spark can query tables defined in the Hive metastore**

Hive Table: `people`

age	first_name	last_name	pcode
52	Grace	Hopper	02134
null	Alan	Turing	94020
28	Ada	Lovelace	94020
…	…		…

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Spark SQL Query (2)

```
myDF = spark.sql("SELECT * FROM people WHERE pcode = 94020")

myDF.printSchema()
root
 |-- age: long (nullable = true)
 |-- first_name: string (nullable = true)
 |-- last_name: string (nullable = true)
 |-- pcode: string (nullable = true)

myDF.show()
+----+---------+--------+-----+
| age|firstName|lastName|pcode|
+----+---------+--------+-----+
|null|     Alan|  Turing|94020|
|  28|      Ada|Lovelace|94020|
+----+---------+--------+-----+
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: A More Complex Query

```
maAgeDF = spark.
  sql("SELECT MEAN(age) AS mean_age,STDDEV(age)
       AS sdev_age FROM people WHERE pcode IN
         (SELECT pcode FROM pcodes WHERE state='MA')")

maAgeDF.printSchema()
root
 |-- mean_age: double (nullable = true)
 |-- sdev_age: double (nullable = true)

maAgeDF.show()
+--------+------------------+
|mean_age|          sdev_age|
+--------+------------------+
|    50.0|2.828427124746190 3|
+--------+------------------+
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

SQL Queries and DataFrame Queries

- **SQL queries and DataFrame transformations provide equivalent functionality**
- **Both are executed as series of transformations**
  - Optimized by the Catalyst optimizer
- **The following Python examples are equivalent**

```
myDF = spark.sql("SELECT * FROM people WHERE pcode = 94020")
```

```
myDF = spark.read.table("people").where("pcode=94020")
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Querying Tables and Views with Apache Spark SQL

- Querying Tables in Spark Using SQL
- **Querying Files and Views**
- The Catalog API
- Comparing Spark SQL, Apache Impala, and Apache Hive-on-Spark
- Essential Points
- Hands-On Exercise: Querying Tables and Views with SQL

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

SQL Queries on Files

- **You can query directly from Parquet or JSON files that are not Hive tables**

```
spark. \
  sql("SELECT * FROM parquet.`/loudacre/people.parquet`
    WHERE firstName LIKE 'A%' "). \
  show()
+-----+--------+---------+---+
|pcode|lastName|firstName|age|
+-----+--------+---------+---+
|94020|  Turing|     Alan| 32|
|94020|Lovelace|      Ada| 28|
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

SQL Queries on Views

- **You can also query a *view***
  - Views provide the ability to perform SQL queries on a DataFrame or Dataset
- **Views are temporary**
  - Regular views can only be used within a single Spark session
  - Global views can be shared between multiple Spark sessions within a single Spark application
- **Creating a view**
  - `DataFrame.createTempView(`*`view-name`*`)`
  - `DataFrame.createOrReplaceTempView(`*`view-name`*`)`
  - `DataFrame.createGlobalTempView(`*`view-name`*`)`

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Creating and Querying a View

- **After defining a DataFrame view, you can query with SQL just as with a table**

```
spark.read.load("/loudacre/people.parquet"). \
  select("firstName", "lastName"). \
  createTempView("user_names")

spark.sql( \
    "SELECT * FROM user_names WHERE firstName LIKE 'A%'" ). \
  show()
+---------+--------+
|firstName|lastName|
+---------+--------+
|     Alan|  Turing|
|      Ada|Lovelace|
+---------+--------+
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Querying Tables and Views with Apache Spark SQL

- Querying Tables in Spark Using SQL
- Querying Files and Views
- **The Catalog API**
- Comparing Spark SQL, Apache Impala, and Apache Hive-on-Spark
- Essential Points
- Hands-On Exercise: Querying Tables and Views with SQL

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

The Catalog API

- **Use the Catalog API to explore tables and manage views**
- **The entry point for the Catalog API is `spark.catalog`**
- **Functions include**
  - `listDatabases` returns a Dataset (Scala) or list (Python) of existing databases
  - `setCurrentDatabase(`*`dbname`*`)` sets the current default database for the session
    - Equivalent to the `USE` statement in SQL
  - `listTables` returns a Dataset (Scala) or list (Python) of tables and views in the current database
  - `listColumns(`*`tablename`*`)` returns a Dataset (Scala) or list (Python) of the columns in the specified table or view
  - `dropTempView(`*`viewname`*`)` removes a temporary view

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Listing Tables and Views (Scala)

```
spark.catalog.listTables.show
+------------+--------+-----------+---------+-----------+
|        name|database|description|tableType|isTemporary|
+------------+--------+-----------+---------+-----------+
|      people| default|       null| EXTERNAL|      false|
|  user_names| default|       null|TEMPORARY|       true|
+------------+--------+-----------+---------+-----------+
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Listing Tables and Views (Python)

```
for table in spark.catalog.listTables(): print table
Table(name=u'people', database=u'default',
  description=None, tableType=u'EXTERNAL',
  isTemporary=False)
Table(name=u'user_names', database=u'default’,
 description=None,
 tableType=u'TEMPORARY',
 isTemporary=True)
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Querying Tables and Views with Apache Spark SQL

- Querying Tables in Spark Using SQL
- Querying Files and Views
- The Catalog API
- **Comparing Spark SQL, Apache Impala, and Apache Hive-on-Spark**
- Essential Points
- Hands-On Exercise: Querying Tables and Views with SQL

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Querying Tables with SQL

- **Data analysts often need to query Hive metastore tables**
  - Reports and ad hoc queries
  - Many data analysts are familiar with SQL
    - No experience with a procedural language required
- **There are several ways to use SQL with tables in Hive**
  - Apache Impala
  - Apache Hive
    - Running on Hadoop MapReduce or Spark
  - Spark SQL API (`SparkSession.sql`)
  - Spark SQL Server command-line interface

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Apache Impala

- **Impala is a specialized SQL engine**
  - Better performance than Spark SQL
  - More mature
  - Robust security using Apache Sentry
  - Highly optimized
  - Low latency
- **Best for**
  - Interactive and ad hoc queries
  - Data analysis
  - Integration with third-party visual analytics and business intelligence tools
    - Such as Tableau, Zoomdata, or Microstrategy
  - Typical job: seconds or less

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Apache Hive

- **Apache Hive**
  - Runs using either Spark or MapReduce
  - In most cases, Hive on Spark has much better performance
  - Very mature
  - High stability and resilience
- **Best for**
  - Batch ETL processing
  - Typical job: minutes to hours

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark SQL API

- **Spark SQL API**
  - Mixed procedural and SQL applications
  - Supports a rich ecosystem of related APIs for machine learning, streaming, statistical computations
  - Catalyst optimizer for good performance
  - Supports Python, a common language for data scientists
- **Best for**
  - Complex data manipulation and analytics
  - Integration with other data systems and APIs
  - Machine learning
  - Streaming and other long-running applications

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark SQL Server

- **Spark SQL Server**
  - Lightweight tool included with Spark
  - Easy, convenient setup
  - Includes a command line interface: `spark-sql`
- **Best for**
  - Non-production use such as testing and data exploration
  - Running locally without a cluster

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Querying Tables and Views with Apache Spark SQL

- Querying Tables in Spark Using SQL
- Querying Files and Views
- The Catalog API
- Comparing Spark SQL, Apache Impala, and Apache Hive-on-Spark
- **Essential Points**
- Hands-On Exercise: Querying Tables and Views with SQL

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **You can query using SQL in addition to DataFrame and Dataset operations**
  - Incorporates SQL queries into procedural development
- **You can use SQL with Hive tables and temporary views**
  - Temporary views let you use SQL on data in DataFrames and Datasets
- **SQL queries and DataFrame/Dataset queries are equivalent**
  - Both are optimized by Catalyst
- **The Catalog API lets you list and describe tables, views, and columns, choose a database, or delete a view**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Querying Tables and Views with Apache Spark SQL

- Querying Tables in Spark Using SQL
- Querying Files and Views
- The Catalog API
- Comparing Spark SQL, Apache Impala, and Apache Hive-on-Spark
- Essential Points
- **Hands-On Exercise: Querying Tables and Views with SQL**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Querying Tables and Views with SQL

- **In this exercise, you will query tables and views with SQL**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Working with Datasets in Scala

Chapter 12

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- **Working with Datasets in Scala**
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Working with Datasets in Scala

In this chapter, you will learn

- **What Datasets are and how they differ from DataFrames**
- **How to create Datasets in Scala from data sources and in-memory data**
- **How to query Datasets using typed and untyped transformations**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Working with Datasets in Scala

- **Datasets and DataFrames**
- Creating Datasets
- Loading and Saving Datasets
- Dataset Operations
- Essential Points
- Hands-On Exercise: Using Datasets in Scala

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

What Is a Dataset?

- **A distributed collection of *strongly-typed* objects**
  - Primitive types such as `Int` or `String`
  - Complex types such as arrays and lists containing supported types
  - `Product` objects based on Scala case classes (or JavaBean objects in Java)
  - `Row` objects
- **Mapped to a relational schema**
  - The schema is defined by an encoder
  - The schema maps object properties to typed columns
- **Implemented only in Scala and Java**
  - Python is not a statically-typed language—no benefit from Dataset strong typing

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Datasets and DataFrames

- **In Scala, DataFrame is an alias for a Dataset containing `Row` objects**
  - There is no distinct class for `DataFrame`
- **DataFrames and Datasets represent different types of data**
  - DataFrames (Datasets of `Row` objects) represent tabular data
  - Datasets represent typed, object-oriented data
- **DataFrame transformations are referred to as *untyped***
  - `Row`s can hold elements of any type
  - Schemas defining column types are not applied until runtime
- **Dataset transformations are *typed***
  - Object properties are inherently typed at compile time

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Working with Datasets in Scala

- Datasets and DataFrames
- **Creating Datasets**
- Loading and Saving Datasets
- Dataset Operations
- Essential Points
- Hands-On Exercise: Using Datasets in Scala

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Creating Datasets: A Simple Example

- Use `SparkSession.createDataset(Seq)` to create a Dataset from in-memory data (experimental)
- Example: Create a Dataset of strings (`Dataset[String]`)

```
val strings = Seq("a string","another string")
val stringDS = spark.createDataset(strings)
stringDS.show
+--------------+
|         value|
+--------------+
|      a string|
|another string|
+--------------+
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Datasets and Case Classes (1)

- **Scala case classes are a useful way to represent data in a Dataset**
  - They are often used to create simple data-holding objects in Scala
  - Instances of case classes are called *products*

```
case class Name(firstName: String, lastName: String)

val names = Seq(Name("Fred","Flintstone"),
                Name("Barney","Rubble"))
names.foreach(name => println(name.firstName))
Fred
Barney
```

Continues on next slide...

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Datasets and Case Classes (2)

- **Encoders define a Dataset's schema using reflection on the object type**
  - Case class arguments are treated as columns

```
import spark.implicits._ // required if not running in shell

val namesDS = spark.createDataset(names)
namesDS.show
+---------+----------+
|firstName|  lastName|
+---------+----------+
|     Fred|Flintstone|
|   Barney|    Rubble|
+---------+----------+
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Type Safety in Datasets and DataFrames

- Type safety means that type errors are found at compile time rather than runtime
- Example: Assigning a `String` value to an `Int` variable

```
val i:Int = namesDS.first.lastName // Name(Fred,Flintsone)
Compilation: error: type mismatch;
         found: String / required: Int
```

```
val row = namesDF.first // Row(Fred,Flintstone)
val i:Int = row.getInt(row.fieldIndex("lastName"))
Run time: java.lang.ClassCastException: java.lang.String
       cannot be cast to java.lang.Integer
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Working with Datasets in Scala

- Datasets and DataFrames
- Creating Datasets
- **Loading and Saving Datasets**
- Dataset Operations
- Essential Points
- Hands-On Exercise: Using Datasets in Scala

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Loading and Saving Datasets

- **You cannot load a Dataset directly from a structured source**
  - Create a Dataset by loading a DataFrame or RDD and converting to a Dataset
- **Datasets are saved as DataFrames**
  - Save using `Dataset.write` (returns a `DataFrameWriter`)
  - The type of object in the Dataset is not saved

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Creating a Dataset from a DataFrame (1)

- **Use `DataFrame.as[`*`type`*`]` to create a Dataset from a DataFrame**
  - Encoders convert `Row` elements to the Dataset's type
  - The `DataFrame.as` function is experimental
- **Example: a Dataset of type `Name` based a JSON file**

Data File: `names.json`

```
{"firstName":"Grace","lastName":"Hopper"}
{"firstName":"Alan","lastName":"Turing"}
{"firstName":"Ada","lastName":"Lovelace"}
{"firstName":"Charles","lastName":"Babbage"}
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Creating a Dataset from a DataFrame (2)

```
val namesDF = spark.read.json("names.json")
namesDF: org.apache.spark.sql.DataFrame =
  [firstName: string, lastName: string]

namesDF.show
+---------+--------+
|firstName|lastName|
+---------+--------+
|    Grace|  Hopper|
|     Alan|  Turing|
|      Ada|Lovelace|
|  Charles| Babbage|
+---------+--------+
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Creating a Dataset from a DataFrame (3)

```
case class Name(firstName: String, lastName: String)

val namesDS = namesDF.as[Name]
namesDS: org.apache.spark.sql.Dataset[Name] =
  [firstName: string, lastName: string]

namesDS.show
+---------+--------+
|firstName|lastName|
+---------+--------+
|    Grace|  Hopper|
|     Alan|  Turing|
|      Ada|Lovelace|
|  Charles| Babbage|
+---------+--------+
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Creating Datasets from RDDs (1)

- **Datasets can be created based on RDDs**
  - Useful with unstructured or semi-structured data such as text
- **Example:**
  - Tab-separated text file with postal code, latitude, and longitude

```
00210\t43.005895\t71.013202
01014\t42.170731\t-72.604842
01062\t42.324232\t-72.67915
…
```

  - Output: Dataset with `PcodeLatLon(pcode,(lat,lon))` objects

```
case class PcodeLatLon(pcode: String,
                       latlon: Tuple2[Double,Double])
```

Continues on next slide…

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Creating Datasets from RDDs (2)

- **Parse input to structure the data**

```
val pLatLonRDD = sc.textFile("latlon.tsv").
  map(line => line.split('\t')).
  map(fields =>
    (PcodeLatLon(fields(0),
       (fields(1).toFloat,fields(2).toFloat))))
```

Continues on next slide...

PcodeLatLon("00210",(43.005895,-71.013202))
PcodeLatLon("01014",(42.170731,-72.604842))
PcodeLatLon("01062",(42.324232,-72.67915))
…

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Creating Datasets from RDDs (3)

- **Convert RDD to a Dataset of `PcodeLatLon` objects**

```
val pLatLonDS = spark.createDataset(pLatLonRDD)
pLatLonDF: org.apache.spark.sql.Dataset[PcodeLatLon] = [pcode:
 string, latlon: struct<_1: double, _2: double>]

pLatLonDS.printSchema
root
 |-- pcode: string (nullable = true)
 |-- latlon: struct (nullable = true)
 |    |-- _1: double (nullable = true)
 |    |-- _2: double (nullable = true)

println(pLatLonDS.first)
PcodeLatLon(00210,(43.00589370727539,-71.01319885253906))
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Working with Datasets in Scala

- Datasets and DataFrames
- Creating Datasets
- Loading and Saving Datasets
- **Dataset Operations**
- Essential Points
- Hands-On Exercise: Using Datasets in Scala

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Typed and Untyped Transformations (1)

- **Typed transformations create a new Dataset based on an existing Dataset**
  - Typed transformations can be used on Datasets of any type (including `Row`)
- **Untyped transformations return DataFrames (Datasets containing `Row` objects) or untyped `Columns`**
  - Do not preserve type of the data in the parent Dataset

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Typed and Untyped Transformations (2)

- **Untyped operations (those that return Row Datasets) include**
  - `join`
  - `groupBy`
  - `col`
  - `drop`
  - `select` (using column names or `Columns`)
- **Typed operations (operations that return typed Datasets) include**
  - `filter` (and its alias, `where`)
  - `distinct`
  - `limit`
  - `sort` (and its alias, `orderBy`)
  - `groupByKey` (experimental)

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Typed and Untyped Transformations (1)

```
case class Person(pcode:String, lastName:String,
                  firstName:String, age:Int)

val people = Seq(Person("02134","Hopper","Grace",48),…)

val peopleDS = spark.createDataset(people)
peopleDS: org.apache.spark.sql.Dataset[Person] =
  [pcode: string, firstName: string ... 2 more fields]
```

Continues on next slide…

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Typed and Untyped Transformations (2)

- **Typed operations return Datasets based on the starting Dataset**
- **Untyped operations return DataFrames (Datasets of Rows)**

```
val sortedDS = peopleDS.sort("age")
sortedDS: org.apache.spark.sql.Dataset[Person] =
  [pcode: string, lastName: string ... 2 more fields]

val firstLastDF = peopleDS.select("firstName","lastName")
firstLastDF: org.apache.spark.sql.DataFrame =
  [firstName: string, lastName: string]
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Combining Typed and Untyped Operations

```
val combineDF = peopleDS.sort("lastName").
  where("age > 40").select("firstName","lastName")
combineDF: org.apache.spark.sql.DataFrame =
  [firstName: string, lastName: string]

combineDF.show
+---------+--------+
|firstName|lastName|
+---------+--------+
|  Charles| Babbage|
|    Grace|  Hopper|
+---------+--------+
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Working with Datasets in Scala

- Datasets and DataFrames
- Creating Datasets
- Loading and Saving Datasets
- Dataset Operations
- **Essential Points**
- Hands-On Exercise: Using Datasets in Scala

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **Datasets represent data consisting of strongly-typed objects**
  - Primitive types, complex types, and `Product` and `Row` objects
  - Encoders map the Dataset's data type to a table-like schema
- **Datasets are defined in Scala and Java**
  - Python is a dynamically-typed language, no need for strongly-typed data representation
- **In Scala and Java, `DataFrame` is just an alias for `Dataset[Row]`**
- **Datasets can be created from in-memory data, DataFrames, and RDDs**
- **Datasets have typed and untyped operations**
  - Typed operations return Datasets based on the original type
  - Untyped operations return DataFrames (Datasets of rows)

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Working with Datasets in Scala

- Datasets and DataFrames
- Creating Datasets
- Loading and Saving Datasets
- Dataset Operations
- Essential Points
- **Hands-On Exercise: Using Datasets in Scala**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Using Datasets in Scala

- **In this exercise, you will create, query, and save Datasets in Scala**
  - Please refer to the Hands-On Exercise Manual for instructions

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

cloudera®

Writing, Configuring, and Running Apache Spark Applications

Chapter 13

cloudera®

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- **Writing, Configuring, and Running Apache Spark Applications**
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Writing and Running Apache Spark Applications

In this chapter, you will learn

- **The difference between a Spark application and the Spark shell**
- **How to write a Spark application**
- **How to build a Scala or Java Spark application**
- **How to submit and run a Spark application**
- **How to view the Spark application web UI**
- **How to configure Spark application properties**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Writing, Configuring, and Running Apache Spark Applications

- **Writing a Spark Application**
- Building and Running an Application
- Application Deployment Mode
- The Spark Application Web UI
- Configuring Application Properties
- Essential Points
- Hands-On Exercise: Writing, Configuring, and Running a Spark Application

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

The Spark Shell and Spark Applications

- **The Spark shell allows interactive exploration and manipulation of data**
  - REPL using Python or Scala
- **Spark applications run as independent programs**
  - For jobs such as ETL processing, streaming, and so on
  - Python, Scala, or Java

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

The Spark Session and Spark Context

- **Every Spark program needs**
  - One `SparkContext` object
  - One or more `SparkSession` objects
    - If you are using Spark SQL
- **The interactive shell creates these for you**
  - A `SparkSession` object called `spark`
  - A `SparkContext` object called `sc`
- **In a standalone Spark application you must create these yourself**
  - Use a Spark session builder to create a new session
    - The builder automatically creates a new Spark context as well
  - Call `stop` on the session or context when program terminates

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Creating a `SparkSession` Object

- **`SparkSession.builder` points to a `Builder` object**
  - Use the builder to create and configure a `SparkSession` object
- **The `getOrCreate` builder function returns the existing `SparkSession` object if it exists**
  - Creates a new Spark session if none exists
  - Automatically creates a new `SparkContext` object as `sparkContext` on the `SparkSession` object

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Python Example: Name List

```
import sys

from pyspark.sql import SparkSession

if __name__ == "__main__":
  if len(sys.argv) < 3:
    print >> sys.stderr,
      "Usage: NameList.py <input-file> <output-file>"
    sys.exit()

  spark = SparkSession.builder.getOrCreate()
  spark.sparkContext.setLogLevel("WARN")

  peopleDF = spark.read.json(sys.argv[1])
  namesDF = peopleDF.select("firstName","lastName")
  namesDF.write.option("header","true").csv(sys.argv[2])

  spark.stop()
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Scala Example: Name List

```
import org.apache.spark.sql.SparkSession

object NameList {
  def main(args: Array[String]) {
    if (args.length < 2) {
      System.err.println(
        "Usage: NameList <input-file> <output-file>")
      System.exit(1)
    }

    val spark = SparkSession.builder.getOrCreate()
    spark.sparkContext.setLogLevel("WARN")
    val peopleDF = spark.read.json(args(0))
    val namesDF = peopleDF.select("firstName","lastName")
    namesDF.write.option("header","true").csv(args(1))

    spark.stop
  }
}
```

Language: *Scala*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Writing, Configuring, and Running Apache Spark Applications

- Writing a Spark Application
- **Building and Running an Application**
- Application Deployment Mode
- The Spark Application Web UI
- Configuring Application Properties
- Essential Points
- Hands-On Exercise: Writing, Configuring, and Running a Spark Application

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Building an Application: Scala or Java

- **Scala or Java Spark applications must be compiled and assembled into JAR files**
  - JAR file will be passed to worker nodes
- **Apache Maven is a popular build tool**
  - For specific setting recommendations, see the Spark Programming Guide
- **Build details will differ depending on**
  - Version of Hadoop and CDH
  - Deployment platform (YARN, Mesos, Spark Standalone)
- **Consider using an Integrated Development Environment (IDE)**
  - IntelliJ or Eclipse are two popular examples
  - Can run Spark locally in a debugger

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Building Scala Applications in the Hands-On Exercises

- **Basic Apache Maven projects are provided in the exercise directory**
  - `stubs`: starter Scala files—do exercises here
  - `solution`: exercise solutions
  - `bonus`: bonus solutions
- **Build command: `mvn package`**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Running a Spark Application

- **The easiest way to run a Spark application is to use the submit script**
  - Python

```
$ spark2-submit NameList.py people.json namelist/
```

  - Scala or Java

```
$ spark2-submit --class NameList MyJarFile.jar \
  people.json namelist/
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Submit Script Options

- **The Spark submit script provides many options to specify how the application should run**
  - Most are the same as for `pyspark2` and `spark2-shell`
- **General submit flags include**
  - `master`: `local`, `yarn`, or a Mesos or Spark Standalone cluster manager URI
  - `jars`: Additional JAR files (Scala and Java only)
  - `pyfiles`: Additional Python files (Python only)
  - `driver-java-options`: Parameters to pass to the driver JVM
- **YARN-specific flags include**
  - `num-executors`: Number of executors to start application with
  - `driver-cores`: Number cores to allocate for the Spark driver
  - `queue`: YARN queue to run in
- **Show all available options**
  - `help`

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Writing, Configuring, and Running Apache Spark Applications

- Writing a Spark Application
- Building and Running an Application
- **Application Deployment Mode**
- The Spark Application Web UI
- Configuring Application Properties
- Essential Points
- Hands-On Exercise: Writing, Configuring, and Running a Spark Application

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Application Deployment Mode

- **Spark applications can run**
  - Locally with one or more threads
  - On a cluster
    - In `client` mode (default), the driver runs locally on a gateway node
      - Requires direct communication between driver and cluster worker nodes
    - In `cluster` mode, the driver runs in the application master on the cluster
      - Common in production systems
- **Specify the deployment mode when submitting the application**

```
$ spark2-submit --master yarn --deploy-mode cluster \
  NameList.py people.json namelist/
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark Deployment Mode on YARN: Client Mode (1)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark Deployment Mode on YARN: Client Mode (2)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark Deployment Mode on YARN: Client Mode (3)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark Deployment Mode on YARN: Cluster Mode (1)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark Deployment Mode on YARN: Cluster Mode (2)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark Deployment Mode on YARN: Cluster Mode (3)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark Deployment Mode on YARN: Cluster Mode (4)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Writing, Configuring, and Running Apache Spark Applications

- Writing a Spark Application
- Building and Running an Application
- Application Deployment Mode
- **The Spark Application Web UI**
- Configuring Application Properties
- Essential Points
- Hands-On Exercise: Writing, Configuring, and Running a Spark Application

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

The Spark Application Web UI

- **The Spark UI lets you monitor running jobs, and view statistics and configuration**

Spark 2.1.0.cloudera1 | Jobs | Stages | Storage | Environment | Executors | SQL | Spark shell application UI

Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Blacklisted
Total(4)	13	267.6 KB / 1.5 GB	0.0 B	3	1	0	60	61	12 s (0.4 s)	35.6 MB	0.0 B	2.2 MB	0
Dead(2)	4	87.8 KB / 768.2 MB	0.0 B	2	0	0	60	60	12 s (0.4 s)	35.6 MB	0.0 B	2.2 MB	0
Active(2)	9	179.8 KB / 768.2 MB	0.0 B	1	1	0	0	1	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0

Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
driver	10.0.6.229:40885	Active	7	148.5 KB / 384.1 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump
1	worker-2:46762	Dead	2	30.6 KB / 384.1 MB	0.0 B	1	0	0	1	1	2 s (49 ms)	65.5 KB	0.0 B	0.0 B	stdout stderr	Thread Dump
2	worker-2:36897	Dead	2	57.2 KB / 384.1 MB	0.0 B	1	0	0	59	59	10 s (0.3 s)	35.5 MB	0.0 B	2.2 MB	stdout stderr	Thread Dump
3	worker-2:42528	Active	2	31.3 KB / 384.1 MB	0.0 B	1	1	0	0	1	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump

Showing 1 to 4 of 4 entries Previous 1 Next

Spark 2.1.0.cloudera1 | Jobs

Spark Jobs (?)

User: training
Total Uptime: 50 min
Scheduling Mode: FIFO
Completed Jobs: 4

Event Timeline

Completed Jobs (4)

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
3	saveAsTextFile at <console>:33	2017/05/26 06:52:57	12 s	3/3	41/41
2	saveAsTextFile at <console>:33	2017/05/26 06:50:38	2 s	1/1 (2 skipped)	18/18 (23 skipped)
1	saveAsTextFile at <console>:33	2017/05/26 06:50:19	13 s	3/3	41/41
0	first at <console>:27	2017/05/26 06:46:47	15 s	1/1	1/1

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Accessing the Spark UI

- **The web UI is run by the Spark driver**
  - When running locally: `http://localhost:4040`
  - When running in client mode: `http://gateway:4040`
  - When running in cluster mode, access via the YARN UI

ueue	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Progress	Tracking UI
sers.training	Fri May 26 07:12:50 -0700 2017	N/A	RUNNING	UNDEFINED	1	1	1024		ApplicationMaster
sers.training	Fri May 26 07:09:52 -0700 2017	Fri May 26 07:10:41 -0700 2017	FINISHED	SUCCEEDED	N/A	N/A	N/A		History
sers.training	Fri May 26 07:05:45 -0700 2017	Fri May 26 07:09:44 -0700 2017	FINISHED	SUCCEEDED	N/A	N/A	N/A		History

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark Application History UI

- **The Spark UI is only available while the application is running**
- **Use the Spark application history server to view metrics for a completed application**
  - Optional Spark component

Spark 2.1.0.cloudera1 **History Server**

Event log directory: hdfs://master-1:8020/user/spark/spark2ApplicationHistory

Last updated: 5/26/2017, 7:11:04 AM

Search:

App ID	App Name	Started	Completed	Duration	Spark User	Last Updated	Event Log
application_1495803008771_0004	PythonWordCount	2017-05-26 14:09:48	2017-05-26 14:10:41	53 s	training	2017-05-26 14:10:41	Download
application_1495803008771_0003	PythonWordCount	2017-05-26 14:05:40	2017-05-26 14:09:43	4.1 min	training	2017-05-26 14:09:43	Download
application_1495803008771_0001	Spark shell	2017-05-26 13:07:56	2017-05-26 14:02:39	55 min	training	2017-05-26 14:02:39	Download

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Viewing the Application History UI

- **You can access the history server UI by**
  - Using a URL with host and port configured by a system administrator
  - Following the History link in the YARN UI

ueue	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Progress	Tracking UI
sers.training	Fri May 26 07:12:50 -0700 2017	N/A	RUNNING	UNDEFINED	1	1	1024		ApplicationMaster
sers.training	Fri May 26 07:09:52 -0700 2017	Fri May 26 07:10:41 -0700 2017	FINISHED	SUCCEEDED	N/A	N/A	N/A		History
sers.training	Fri May 26 07:05:45 -0700 2017	Fri May 26 07:09:44 -0700 2017	FINISHED	SUCCEEDED	N/A	N/A	N/A		History

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Writing, Configuring, and Running Apache Spark Applications

- Writing a Spark Application
- Building and Running an Application
- Application Deployment Mode
- The Spark Application Web UI
- **Configuring Application Properties**
- Essential Points
- Hands-On Exercise: Writing, Configuring, and Running a Spark Application

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark Application Configuration Properties

- **Spark provides numerous properties to configure your application**
- **Some example properties**
  - `spark.master`: Cluster type or URI to submit application to
  - `spark.app.name`: Application name displayed in the Spark UI
  - `spark.submit.deployMode`: Whether to run application in client or cluster mode (default: `client`)
  - `spark.ui.port`: Port to run the Spark Application UI (default `4040`)
  - `spark.executor.memory`: How much memory to allocate to each Executor (default `1g`)
  - `spark.pyspark.python`: Which Python executable to use for Pyspark applications
  - And many more…
    - See the Spark Configuration page in the Spark documentation for more details

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Setting Configuration Properties

- **Most properties are set by system administrators**
  - Managed manually or using Cloudera Manager
  - Stored in a *properties file*
- **Developers can override system settings when submitting applications by**
  - Using submit script flags
  - Loading settings from a custom properties file instead of the system file
  - Setting properties programmatically in the application
- **Properties that are not set explicitly use Spark default values**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Overriding Properties Using Submit Script

- **Some Spark submit script flags set application properties**
  - For example
    - Use `--master` to set `spark.master`
    - Use `--name` to set `spark.app.name`
    - Use `--deploy-mode` to set `spark.submit.deployMode`
- **Not every property has a corresponding script flag**
  - Use `--conf` to set any property

```
$ spark2-submit \
  --conf spark.pyspark.python=/usr/bin/python2.7
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Setting Properties in a Properties File

- **System administrators set system properties in properties files**
  - You can use your own custom properties file instead

```
spark.master                local[*]
spark.executor.memory       512k
spark.pyspark.python        /usr/bin/python2.7
```

- **Specify your properties file using the `properties-file` option**

```
$ spark2-submit \
  --properties-file=dir/my-properties.conf
```

- **Note that Spark will load *only* your custom properties file**
  - System properties file is ignored
  - Copy important system settings into your custom properties file
  - Custom file will not reflect future changes to system settings

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Setting Configuration Properties Programmatically

- **Spark configuration settings are part of the Spark session or Spark context**
- **Set using the Spark session builder functions**
  - `appName` sets `spark.app.name`
  - `master` sets `spark.master`
  - `config` can set any property

```
import org.apache.spark.sql.SparkSession
…
val spark = SparkSession.builder.
  appName("my-spark-app").
  config("spark.ui.port","5050").
  getOrCreate()
…
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Priority of Spark Property Settings

- **Properties set with higher priority methods override lower priority methods**
  1. Programmatic settings
  2. Submit script (command line) settings
  3. Properties file settings
     - Either administrator site-wide file or custom properties file
  4. Spark default settings
     - See the Spark Configuration guide

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Viewing Spark Properties

- **You can view the Spark property settings two ways**
  - Using `--verbose` with the submit script
  - In the Spark Application UI Environment tab

Spark 2.1.0.cloudera1 | Jobs | Stages | Storage | Environment | Executors | SQL | PySparkShell application UI

Environment

Runtime Information

Name	Value
Java Home	/usr/java/jdk1.7.0_67-cloudera/jre
Java Version	1.7.0_67 (Oracle Corporation)
Scala Version	version 2.11.8

Spark Properties

Name	Value
spark.app.id	application_1495803008771_0005
spark.app.name	PySparkShell
spark.authenticate	false
spark.driver.appUIAddress	http://10.0.6.229:4040
spark.driver.extraLibraryPath	/opt/cloudera/parcels/CDH-5.11.0-1.cdh5.11.0.p0.34 /lib/hadoop/lib/native
spark.driver.host	10.0.6.229
spark.driver.port	50884

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Writing, Configuring, and Running Apache Spark Applications

- Writing a Spark Application
- Building and Running an Application
- Application Deployment Mode
- The Spark Application Web UI
- Configuring Application Properties
- **Essential Points**
- Hands-On Exercise: Writing, Configuring, and Running a Spark Application

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **Use the Spark shell for interactive data exploration**
- **Write a Spark application to run independently**
- **Spark applications require a `SparkContext` object and usually a `SparkSession` object**
- **Use Maven or a similar build tool to compile and package Scala and Java applications**
  - Not required for Python
- **Deployment mode determines where the application driver runs—on the gateway or on a worker node**
- **Use the `spark2-submit` script to run Spark applications locally or on a cluster**
- **Application properties can be set on the command line, in a properties file, or in the application code**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Writing, Configuring, and Running Apache Spark Applications

- Writing a Spark Application
- Building and Running an Application
- Application Deployment Mode
- The Spark Application Web UI
- Configuring Application Properties
- Essential Points
- **Hands-On Exercise: Writing, Configuring, and Running a Spark Application**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Writing, Configuring, and Running a Spark Application

- **In this exercise, you will write, configure, and run a standalone Spark application**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Distributed Processing

Chapter 14

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- **Distributed Processing**
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Distributed Processing

In this chapter, you will learn

- **How partitions distribute data in RDDs, DataFrames, and Datasets across a cluster**
- **How Spark executes queries in parallel**
- **How to control parallelization through partitioning**
- **How to view execution plans**
- **How to view and monitor tasks and stages**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Distributed Processing

- **Review: Apache Spark on a Cluster**
- RDD Partitions
- Example: Partitioning in Queries
- Stages and Tasks
- Job Execution Planning
- Example: Catalyst Execution Plan
- Example: RDD Execution Plan
- Essential Points
- Hands-On Exercise: Exploring Query Execution

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Review of Spark on YARN (1)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Review of Spark on YARN (2)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Review of Spark on YARN (3)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Distributed Processing

- Review: Apache Spark on a Cluster
- **RDD Partitions**
- Example: Partitioning in Queries
- Stages and Tasks
- Job Execution Planning
- Example: Catalyst Execution Plan
- Example: RDD Execution Plan
- Essential Points
- Hands-On Exercise: Exploring Query Execution

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Data Partitioning (1)

- **Data in Datasets and DataFrames is managed by underlying RDDs**
- **Data in an RDD is *partitioned* across executors**
  - This is what makes RDDs *distributed*
  - Spark assigns tasks to process a partition to the executor managing that partition
- **Data Partitioning is done automatically by Spark**
  - In some cases, you can control how many partitions are created
  - More partitions = more parallelism

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Data Partitioning (2)

- **Spark determines how to partition data in an RDD, Dataset, or DataFrame when**
  - The data source is read
  - An operation is performed on a DataFrame, Dataset, or RDD
  - Spark optimizes a query
  - You call `repartition` or `coalesce`

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Partitioning from Data in Files

- **Partitions are determined when files are read**
  - Core Spark determines RDD partitioning based on location, number, and size of files
    - Usually each file is loaded into a single partition
    - Very large files are split across multiple partitions
  - Catalyst optimizer manages partitioning of RDDs that implement DataFrames and Datasets

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Finding the Number of Partitions in an RDD

- **You can view the number of partitions in an RDD by calling the function `getNumPartitions`**

```
myRDD.getNumPartitions
```

Language: *Scala*

```
myRDD.getNumPartitions()
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Distributed Processing

- Review: Apache Spark on a Cluster
- RDD Partitions
- **Example: Partitioning in Queries**
- Stages and Tasks
- Job Execution Planning
- Example: Catalyst Execution Plan
- Example: RDD Execution Plan
- Essential Points
- Hands-On Exercise: Exploring Query Execution

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Average Word Length by Letter (1)

```
avglens = sc.textFile(mydata)
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Average Word Length by Letter (2)

```
avglens = sc.textFile(mydata) \
  .flatMap(lambda line: line.split(' '))
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Average Word Length by Letter (3)

```
avglens = sc.textFile(mydata) \
  .flatMap(lambda line: line.split(' ')) \
  .map(lambda word: (word[0],len(word)))
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Average Word Length by Letter (4)

```
avglens = sc.textFile(mydata) \
  .flatMap(lambda line: line.split(' ')) \
  .map(lambda word: (word[0],len(word))) \
  .groupByKey()
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Average Word Length by Letter (5)

```
avglens = sc.textFile(mydata) \
  .flatMap(lambda line: line.split(' ')) \
  .map(lambda word: (word[0],len(word))) \
  .groupByKey() \
  .map(lambda (k, values): \
     (k, sum(values)/len(values)))
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Distributed Processing

- Review: Apache Spark on a Cluster
- RDD Partitions
- Example: Partitioning in Queries
- **Stages and Tasks**
- Job Execution Planning
- Example: Catalyst Execution Plan
- Example: RDD Execution Plan
- Essential Points
- Hands-On Exercise: Exploring Query Execution

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Stages and Tasks

- **A *task* is a series of operations that work on the same partition and are pipelined together**
- ***Stages* group together tasks that can run in parallel on different partitions of the same RDD**
- ***Jobs* consist of all the stages that make up a query**
- **Catalyst optimizes partitions and stages when using DataFrames and Datasets**
  - Core Spark provides limited optimizations when you work directly with RDDs
    - You need to code most RDD optimizations manually
  - To improve performance, be aware of how tasks and stages are executed when working with RDDs

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Query Stages and Tasks (1)

```
avglens = sc.textFile(mydata) \
  .flatMap(lambda line: line.split(' ')) \
  .map(lambda word: (word[0],len(word))) \
  .groupByKey() \
  .map(lambda (k, values): \
     (k, sum(values)/len(values)))

avglens.saveAsTextFile("avglen-output")
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Query Stages and Tasks (2)

```
avglens = sc.textFile(mydata) \
  .flatMap(lambda line: line.split(' ')) \
  .map(lambda word: (word[0],len(word))) \
  .groupByKey() \
  .map(lambda (k, values): \
     (k, sum(values)/len(values)))

avglens.saveAsTextFile("avglen-output")
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Query Stages and Tasks (3)

```
avglens = sc.textFile(mydata) \
  .flatMap(lambda line: line.split(' ')) \
  .map(lambda word: (word[0],len(word))) \
  .groupByKey() \
  .map(lambda (k, values): \
     (k, sum(values)/len(values)))

avglens.saveAsTextFile("avglen-output")
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Query Stages and Tasks (4)

```
avglens = sc.textFile(mydata) \
  .flatMap(lambda line: line.split(' ')) \
  .map(lambda word: (word[0],len(word))) \
  .groupByKey() \
  .map(lambda (k, values): \
     (k, sum(values)/len(values)))

avglens.saveAsTextFile("avglen-output")
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Query Stages and Tasks (5)

```
avglens = sc.textFile(mydata) \
  .flatMap(lambda line: line.split(' ')) \
  .map(lambda word: (word[0],len(word))) \
  .groupByKey() \
  .map(lambda (k, values): \
     (k, sum(values)/len(values)))

avglens.saveAsTextFile("avglen-output")
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Summary of Spark Terminology

- **Job—a set of tasks executed as a result of an *action***
- **Stage—a set of tasks in a job that can be executed in parallel**
- **Task—an individual unit of work sent to one executor**
- **Application—the set of jobs managed by a single driver**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Distributed Processing

- Review: Apache Spark on a Cluster
- RDD Partitions
- Example: Partitioning in Queries
- Stages and Tasks
- **Job Execution Planning**
- Example: Catalyst Execution Plan
- Example: RDD Execution Plan
- Essential Points
- Hands-On Exercise: Exploring Query Execution

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Execution Plans

- **Spark creates an execution plan for each job in an application**
- **Catalyst creates SQL, Dataset, and DataFrame execution plans**
  - Highly optimized
- **Core Spark creates execution plans for RDDs**
  - Based on RDD lineage
  - Limited optimization

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

How Execution Plans are Created

- **Spark constructs a DAG (Directed Acyclic Graph) based on RDD dependencies**
- ***Narrow* dependencies**
  - Each partition in the child RDD depends on just one partition of the parent RDD
  - No shuffle required between executors
  - Can be pipelined into a single stage
  - Examples: `map`, `filter`, and `union`

- ***Wide* (or *shuffle*) dependencies**
  - Child partitions depend on multiple partitions in the parent RDD
  - Defines a new stage
  - Examples: `reduceByKey`, `join`, and `groupByKey`

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Controlling the Number of Partitions in RDDs (1)

- **Partitioning determines how queries execute on a cluster**
  - More partitions = more parallel tasks
  - Cluster will be under-utilized if there are too few partitions
    - But too many partitions will increase overhead without an offsetting increase in performance
- **Catalyst controls partitioning for SQL, DataFrame, and Dataset queries**
- **You can control how many partitions are created for RDD queries**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Controlling the Number of Partitions in RDDs (2)

- **Specify the number of partitions when data is read**
  - Default partitioning is based on size and number of the files (minimum is two)
  - Specify a different minimum number when reading a file

```
myRDD = sc.textFile(myfile,5)
```

- **Manually repartition**
  - Create a new RDD with a specified number of partitions using `repartition` or `coalesce`
    - `coalesce` reduces the number of partitions without requiring a shuffle
    - `repartition` shuffles the data into more or fewer partitions

```
newRDD = myRDD.repartition(15)
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Controlling the Number of Partitions in RDDs (3)

- **Specify the number of partitions created by transformations**
  - Wide (shuffle) operations such as `reduceByKey` and `join` repartition data
  - By default, the number of partitions created is based on the number of partitions of the parent RDD(s)
  - Choose a different default by configuring the `spark.default.parallelism` property

```
spark.default.parallelism  15
```

  - Override the default with the optional `numPartitions` operation parameter

```
countRDD = wordsRDD. \
  reduceByKey(lambda v1, v2: v1 + v2, 15)
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Catalyst Optimizer

- **Catalyst can improve SQL, DataFrame, and Dataset query performance by optimizing the DAG to**
  - Minimize data transfer between executors
    - Such as *broadcast* joins—small data sets are pushed to the executors where the larger data sets reside
  - Minimize wide (shuffle) operations
    - Such as unioning two RDDs—grouping, sorting, and joining do not require shuffling
  - Pipeline as many operations into a single stage as possible
  - Generate code for a whole stage at run time
  - Break a query job into multiple jobs, executed in a series

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Catalyst Execution Plans

- **Execution plans for DataFrame, Dataset, and SQL queries include the following phases**
  - **Parsed logical plan**—calculated directly from the sequence of operations specified in the query
  - **Analyzed logical plan**—resolves relationships between data sources and columns
  - **Optimized logical plan**—applies rule-based optimizations
  - **Physical plan**—describes the actual sequence of operations
  - **Code generation**—generates bytecode to run on each node, based on a cost model

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Distributed Processing

- Review: Apache Spark on a Cluster
- RDD Partitions
- Example: Partitioning in Queries
- Stages and Tasks
- Job Execution Planning
- **Example: Catalyst Execution Plan**
- Example: RDD Execution Plan
- Essential Points
- Hands-On Exercise: Exploring Query Execution

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Viewing Catalyst Execution Plans

- **You can view SQL, DataFrame, and Dataset (Catalyst) execution plans**
  - Use DataFrame/Dataset `explain`
    - Shows only the physical execution plan by default
    - Pass `true` to see the full execution plan
  - Use **SQL** tab in the Spark UI or history server
    - Shows details of execution after job runs

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Catalyst Execution Plan (1)

```
peopleDF = spark.read. \
  option("header","true").csv("people.csv")
pcodesDF = spark.read. \
  option("header","true").csv("pcodes.csv")
joinedDF = peopleDF.join(pcodesDF, "pcode")
joinedDF.explain(True)

== Parsed Logical Plan ==
'Join UsingJoin(Inner,ArrayBuffer('pcode))
:- Relation[pcode#0,lastName#1,firstName#2,age#3] csv
+- Relation[pcode#9,city#10,state#11] csv
```

Language: *Python*

continued on next slide...

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Catalyst Execution Plan (2)

```
== Analyzed Logical Plan ==
pcode: string, lastName: string, firstName: string, age:
 string, city: string, state: string
Project [pcode#0, lastName#1, firstName#2, age#3, city#10,
 state#11]
+- Join Inner, (pcode#0 = pcode#9)
   :- Relation[pcode#0,lastName#1,firstName#2,age#3] csv
   +- Relation[pcode#9,city#10,state#11] csv

== Optimized Logical Plan ==
Project [pcode#0, lastName#1, firstName#2, age#3, city#10,
 state#11]
+- Join Inner, (pcode#0 = pcode#9)
   :- Filter isnotnull(pcode#0)
   :  +- Relation[pcode#0,lastName#1,firstName#2,age#3] csv
   +- Filter isnotnull(pcode#9)
      +- Relation[pcode#9,city#10,state#11] csv
```

Language: *Python*

continued on next slide…

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Catalyst Execution Plan (3)

```
== Physical Plan ==
*Project [pcode#0, lastName#1, firstName#2, age#3, city#10,
 state#11]
+- *BroadcastHashJoin [pcode#0], [pcode#9], Inner, BuildRight
   :- *Project [pcode#0, lastName#1, firstName#2, age#3]
   :  +- *Filter isnotnull(pcode#0)
   :     +- *Scan csv [pcode#0,lastName#1,firstName#2,age#3]
 Format: CSV, InputPaths: file:/home/training/people.csv,
 PushedFilters: [IsNotNull(pcode)], ReadSchema:
 struct<pcode:string,lastName:string,firstName:string,age:string>
   +- BroadcastExchange
 HashedRelationBroadcastMode(List(input[0, string, true]))
      +- *Project [pcode#9, city#10, state#11]
         +- *Filter isnotnull(pcode#9)
            +- *Scan csv [pcode#9,city#10,state#11]
 Format: CSV, InputPaths: file:/home/training/pcodes.csv,
 PushedFilters: [IsNotNull(pcode)], ReadSchema:
 struct<pcode:string,city:string,state:string></
pcode:string,city:string,state:string>
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Catalyst Execution Plan (4)

Jobs | Stages | Storage | Environment | Executors | SQL

SQL

Completed Queries

ID	Description	Submitted	Duration	Jobs
0	collect at <ipython-input-65-31b1b37d0504>:1 +details	2017/05/24 08:45:13	1 s	31 32

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Catalyst Execution Plan (5)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Distributed Processing

- Review: Apache Spark on a Cluster
- RDD Partitions
- Example: Partitioning in Queries
- Stages and Tasks
- Job Execution Planning
- Example: Catalyst Execution Plan
- **Example: RDD Execution Plan**
- Essential Points
- Hands-On Exercise: Exploring Query Execution

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Viewing RDD Execution Plans

- **You can view RDD (lineage-based) execution plans**
  - Use the RDD `toDebugString` function
  - Use **Jobs** and **Stages** tabs in the Spark UI or history server
    - Shows details of execution after job runs
- **Note that plans may be different depending on programming language**
  - Plan optimization rules vary

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: RDD Execution Plan (1)

```
val peopleRDD = sc.textFile("people2.csv").keyBy(s => s.split(',')(0))
val pcodesRDD = sc.textFile("pcodes2.csv").keyBy(s => s.split(',')(0))
val joinedRDD = peopleRDD.join(pcodesRDD)
joinedRDD.toDebugString
(2) MapPartitionsRDD[8] at join at … ①
 |  MapPartitionsRDD[7] at join at …
 |  CoGroupedRDD[6] at join at …
 +-(2) MapPartitionsRDD[2] at keyBy at … ②
 |  |  people2.csv MapPartitionsRDD[1] at textFile at …
 |  |  people2.csv HadoopRDD[0] at …
 +-(2) MapPartitionsRDD[5] at keyBy at … ③
    |  pcodes2.csv MapPartitionsRDD[4] at textFile at …
 ④  |  pcodes2.csv HadoopRDD[3] at textFile at …
```

Language: *Scala*

① **Stage 2**

② **Stage 1**

③ **Stage 0**

④ **Indents indicate stages (shuffle boundaries)**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: RDD Execution Plan (2)

Jobs | Stages | Storage | Environment | Executors | SQL

Spark Jobs (?)

User: training
Total Uptime: 55 s
Scheduling Mode: FIFO
Completed Jobs: 1

▸ Event Timeline

Completed Jobs (1)

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	collect at <console>:31	2017/05/24 11:09:51	1 s	3/3	6/6

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: RDD Execution Plan (3)

Details for Job 0

Status: SUCCEEDED
Completed Stages: 3

▸ Event Timeline
▾ DAG Visualization

Completed Stages (3)

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
2	collect at <console>:31 +details	2017/05/24 11:09:52	0.2 s	2/2			586.0 B	
1	keyBy at <console>:24 +details	2017/05/24 11:09:51	99 ms	2/2	170.0 B			339.0 B
0	keyBy at <console>:24 +details	2017/05/24 11:09:51	0.7 s	2/2	105.0 B			247.0 B

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Distributed Processing

- Review: Apache Spark on a Cluster
- RDD Partitions
- Example: Partitioning in Queries
- Stages and Tasks
- Job Execution Planning
- Example: Catalyst Execution Plan
- Example: RDD Execution Plan
- **Essential Points**
- Hands-On Exercise: Exploring Query Execution

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **Spark partitions split data across different executors in an application**
- **Executors execute query tasks that process the data in their partitions**
- **Narrow operations like `map` and `filter` are pipelined within a single stage**
  - Wide operations like `groupByKey` and `join` shuffle and repartition data between stages
- **Jobs consist of a sequence of stages triggered by a single action**
- **Jobs execute according to execution plans**
  - Core Spark creates RDD execution plans based on RDD lineages
  - Catalyst builds optimized query execution plans
- **You can explore how Spark executes queries in the Spark Application UI**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Distributed Processing

- Review: Apache Spark on a Cluster
- RDD Partitions
- Example: Partitioning in Queries
- Stages and Tasks
- Job Execution Planning
- Example: Catalyst Execution Plan
- Example: RDD Execution Plan
- Essential Points
- **Hands-On Exercise: Exploring Query Execution**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Exploring Query Executions

- **In this exercise, you will explore how Spark plans and executes RDD and DataFrame/Dataset queries**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Distributed Data Persistence

Chapter 15

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- **Distributed Data Persistence**
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Distributed Persistence

In this chapter, you will learn

- **How to improve performance and fault-tolerance using persistence**
- **When to use different storage levels**
- **How to use the Spark UI to view details about persisted data**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Distributed Data Persistence

- **DataFrame and Dataset Persistence**
- Persistence Storage Levels
- Viewing Persisted RDDs
- Essential Points
- Hands-On Exercise: Persisting Data

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Persistence

- **You can *persist* a DataFrame, Dataset, or RDD**
  - Also called *caching*
  - Data is temporarily saved to memory and/or disk
- **Persistence can improve performance and fault-tolerance**
- **Use persistence when**
  - Query results will be used repeatedly
  - Executing the query again in case of failure would be very expensive
- **Persisted data cannot be shared between applications**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: DataFrame Persistence (1)

```
over20DF = spark.read. \
  json("people.json"). \
  where("age > 20")
pcodesDF = spark.read. \
  json("pcodes.json")
joinedDF = over20DF. \
  join(pcodesDF, "pcode")
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: DataFrame Persistence (2)

```
over20DF = spark.read. \
  json("people.json"). \
  where("age > 20")
pcodesDF = spark.read. \
  json("pcodes.json")
joinedDF = over20DF. \
  join(pcodesDF, "pcode")
joinedDF. \
  where("pcode = 94020"). \
  show()
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: DataFrame Persistence (3)

```
over20DF = spark.read. \
  json("people.json"). \
  where("age > 20")
pcodesDF = spark.read. \
  json("pcodes.json")
joinedDF = over20DF. \
  join(pcodesDF, "pcode"). \
  persist()
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: DataFrame Persistence (4)

```
over20DF = spark.read. \
  json("people.json"). \
  where("age > 20")
pcodesDF = spark.read. \
  json("pcodes.json")
joinedDF = over20DF. \
  join(pcodesDF, "pcode").
  persist()
joinedDF. \
  where("pcode = 94020"). \
  show()
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: DataFrame Persistence (5)

```
over20DF = spark.read. \
  json("people.json"). \
  where("age > 20")
pcodesDF = spark.read. \
  json("pcodes.json")
joinedDF = over20DF. \
  join(pcodesDF, "pcode"). \
  persist()
joinedDF. \
  where("pcode = 94020"). \
  show()
joinedDF. \
  where("pcode = 87501"). \
  show()
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Table and View Persistence

- **Tables and views can be persisted in memory using `CACHE TABLE`**

```
spark.sql("CACHE TABLE people")
```

- **`CACHE TABLE` can create a view based on a SQL query and cache it at the same time**

```
spark.sql("CACHE TABLE over_20 AS SELECT *
  FROM people WHERE age > 20")
```

- **Queries on cached tables work the same as on persisted DataFrames, Datasets, and RDDs**
  - The first query caches the data
  - Subsequent queries use the cached data

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Distributed Data Persistence

- DataFrame and Dataset Persistence
- **Persistence Storage Levels**
- Viewing Persisted RDDs
- Essential Points
- Hands-On Exercise: Persisting Data

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Storage Levels

- ***Storage levels* provide several options to manage how data is persisted**
  - Storage location (memory and/or disk)
  - Serialization of data in memory
  - Replication
- **Specify storage level when persisting a DataFrame, Dataset, or RDD**
  - Tables and views do not use storage levels
    - Always persisted in memory
- **Data is persisted based on partitions of the underlying RDDs**
  - Executors persist partitions in JVM memory or temporary local files
  - The application driver keeps track of the location of each persisted partition's data

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Storage Levels: Location

- **Storage location—where is the data stored?**
  - `MEMORY_ONLY`: Store data in memory if it fits
  - `DISK_ONLY`: Store all partitions on disk
  - `MEMORY_AND_DISK`: Store any partition that does not fit in memory on disk
    - Called *spilling*

```
from pyspark import StorageLevel
myDF.persist(StorageLevel.DISK_ONLY)
```

Language: Python

```
import org.apache.spark.storage.StorageLevel
myDF.persist(StorageLevel.DISK_ONLY)
```

Language: Scala

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Storage Levels: Memory Serialization

- **In Python, data in memory is always serialized**
- **In Scala, you can choose to serialize data in memory**
  - By default, in Scala and Java, data in memory is stored objects
  - Use `MEMORY_ONLY_SER` and `MEMORY_AND_DISK_SER` to serialize the objects into a sequence of bytes instead
  - Much more space efficient but less time efficient
- **Datasets are serialized by Spark SQL encoders, which are very efficient**
  - Plain RDDs use native Java/Scala serialization by default
  - Use Kryo instead for better performance
- **Serialization options do not apply to disk persistence**
  - Files are always in serialized form by definition

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Storage Levels: Partition Replication

- **Replication—store partitions on two nodes**
  - `DISK_ONLY_2`
  - `MEMORY_AND_DISK_2`
  - `MEMORY_ONLY_2`
  - `MEMORY_AND_DISK_SER_2` (Scala and Java only)
  - `MEMORY_ONLY_SER_2` (Scala and Java only)
  - You can also define custom storage levels for additional replication

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Default Storage Levels

- **The `storageLevel` parameter for the DataFrame, Dataset, or RDD `persist` operation is optional**
  - The default for DataFrames and Datasets is `MEMORY_AND_DISK`
  - The default for RDDs is `MEMORY_ONLY`
- **`persist` with no storage level specified is a synonym for `cache`**

```
myDF.persist()
```

is equivalent to

```
myDF.cache()
```

- **Table and view storage level is always `MEMORY_ONLY`**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

When and Where to Persist

- **When should you persist a DataFrame, Dataset, or RDD?**
  - When the data is likely to be reused
    - Such as in iterative algorithms and machine learning
  - When it would be very expensive to recreate the data if a job or node fails
- **How to choose a storage level**
  - **Memory**—use when possible for best performance
    - Save space by serializing the data if necessary
  - **Disk**—use when re-executing the query is more expensive than disk read
    - Such as expensive functions or filtering large datasets
  - **Replication**—use when re-execution is more expensive than bandwidth

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Changing Storage Levels

- **You can remove persisted data from memory and disk**
  - Use `unpersist` for Datasets, DataFrames, and RDDs
  - Use `Catalog.uncacheTable(`*`table-name`*`)` for tables and views
- **Unpersist before changing to a different storage level**
  - Re-persisting already-persisted data results in an exception

```
myDF.unpersist()
myDF.persist(new-level)
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Distributed Data Persistence

- DataFrame and Dataset Persistence
- Persistence Storage Levels
- **Viewing Persisted RDDs**
- Essential Points
- Hands-On Exercise: Persisting Data

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Viewing Persisted RDDs (1)

- **The Storage tab in the Spark UI shows persisted RDDs**

Jobs | Stages | Storage | Environment | Executors | SQL

Storage

RDDs

RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
*Project [acct_num#0] +- *Filter isnotnull(acct_close_dt#2) +- HiveTableScan [acct_num#0, acct_close_dt#2], MetastoreRelation default, accounts	Memory Deserialized 1x Replicated	5	100%	43.4 KB	0.0 B
ShuffledRDD	Disk Serialized 1x Replicated	5	100%	0.0 B	23.4 MB

DataFrame

RDD

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Viewing Persisted RDDs (2)

RDD Storage Info for ShuffledRDD

Storage Level: Memory Deserialized 1x Replicated
Cached Partitions: 5
Total Partitions: 5
Memory Size: 42.0 MB
Disk Size: 0.0 B

Data Distribution on 3 Executors

Host	Memory Usage	Disk Usage
worker-1:57827	8.4 MB (357.8 MB Remaining)	0.0 B
10.0.8.135:36865	0.0 B (366.2 MB Remaining)	0.0 B
worker-2:58504	33.6 MB (332.6 MB Remaining)	0.0 B

5 Partitions

Block Name ▲	Storage Level	Size in Memory	Size on Disk	Executors
rdd_8_0	Memory Deserialized 1x Replicated	8.4 MB	0.0 B	worker-1:57827
rdd_8_1	Memory Deserialized 1x Replicated	8.3 MB	0.0 B	worker-2:58504
rdd_8_2	Memory Deserialized 1x Replicated	8.4 MB	0.0 B	worker-2:58504
rdd_8_3	Memory Deserialized 1x Replicated	8.4 MB	0.0 B	worker-2:58504
rdd_8_4	Memory Deserialized 1x Replicated	8.5 MB	0.0 B	worker-2:58504

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Distributed Data Persistence

- DataFrame and Dataset Persistence
- Persistence Storage Levels
- Viewing Persisted RDDs
- **Essential Points**
- Hands-On Exercise: Persisting Data

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- Persisting data means temporarily storing data in Datasets, DataFrames, RDDs, tables, and views to improve performance and resilience
- Persisted data is stored in executor memory and/or disk files on worker nodes
- Replication can improve performance when recreating partitions after executor failure
- Replication is most useful in iterative applications or when executing a complicated query is very expensive

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Distributed Data Persistence

- DataFrame and Dataset Persistence
- Persistence Storage Levels
- Viewing Persisted RDDs
- Essential Points
- **Hands-On Exercise: Persisting Data**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercises: Persisting Data

- **In this exercise, you will explore DataFrame persistence**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

cloudera®

Common Patterns in Apache Spark Data Processing

Chapter 16

cloudera®

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- **Common Patterns in Apache Spark Data Processing**
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Common Patterns in Apache Spark Data Processing

In this chapter, you will learn

- **At what kinds of processing and analysis Apache Spark is best**
- **How to implement an iterative algorithm in Spark**
- **What major features and benefits are provided by Spark's machine learning libraries**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Common Patterns in Apache Spark Data Processing

- **Common Apache Spark Use Cases**
- Iterative Algorithms in Apache Spark
- Machine Learning
- Example: k-means
- Essential Points
- Hands-On Exercise: Implement an Iterative Algorithm with Apache Spark

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Common Spark Use Cases (1)

- **Spark is especially useful when working with any combination of:**
  - Large amounts of data
    - Distributed storage
  - Intensive computations
    - Distributed computing
  - Iterative algorithms
    - In-memory processing and pipelining

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Common Spark Use Cases (2)

- **Risk analysis**
  - "How likely is this borrower to pay back a loan?"
- **Recommendations**
  - "Which products will this customer enjoy?"
- **Predictions**
  - "How can we prevent service outages instead of simply reacting to them?"
- **Classification**
  - "How can we tell which mail is spam and which is legitimate?"

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark Examples

- **Spark includes many example programs that demonstrate some common Spark programming patterns and algorithms**
  - k-means
  - Logistic regression
  - Calculating pi
  - Alternating least squares (ALS)
  - Querying Apache web logs
  - Processing Twitter feeds
- **Examples**
  - *SPARK_HOME*`/lib`
    - `spark-examples.jar`: Java and Scala examples
    - `python.tar.gz`: Pyspark examples

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Common Patterns in Apache Spark Data Processing

- Common Apache Spark Use Cases
- **Iterative Algorithms in Apache Spark**
- Machine Learning
- Example: k-means
- Essential Points
- Hands-On Exercise: Implement an Iterative Algorithm with Apache Spark

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: PageRank

- **PageRank gives web pages a ranking score based on links from other pages**
  - Higher scores given for more links, and links from other high ranking pages
- **PageRank is a classic example of big data analysis (like word count)**
  - Lots of data: Needs an algorithm that is distributable and scalable
  - Iterative: The more iterations, the better than answer

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

PageRank Algorithm (1)

1. **Start each page with a rank of 1.0**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

PageRank Algorithm (2)

1. Start each page with a rank of 1.0
2. On each iteration:
   a. Each page contributes to its neighbors its own rank divided by the number of its neighbors: $contrib_p = rank_p / neighbors_p$

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

PageRank Algorithm (3)

1. **Start each page with a rank of 1.0**
2. **On each iteration:**
   a. **Each page contributes to its neighbors its own rank divided by the number of its neighbors:** $contrib_p = rank_p / neighbors_p$
   b. **Set each page's new rank based on the sum of its neighbors contribution:** $new_rank = \Sigma contrib * .85 + .15$

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

PageRank Algorithm (4)

1. Start each page with a rank of 1.0
2. On each iteration:
   a. Each page contributes to its neighbors its own rank divided by the number of its neighbors: $contrib_p = rank_p / neighbors_p$
   b. Set each page's new rank based on the sum of its neighbors contribution: $new_rank = \Sigma contrib * .85 + .15$
3. Each iteration incrementally improves the page ranking

cloudera
Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

PageRank Algorithm (5)

1. Start each page with a rank of 1.0
2. On each iteration:
   a. Each page contributes to its neighbors its own rank divided by the number of its neighbors: $contrib_p = rank_p / neighbors_p$
   b. Set each page's new rank based on the sum of its neighbors contribution: $new_rank = \Sigma contrib * .85 + .15$
3. Each iteration incrementally improves the page ranking

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

PageRank in Spark: Neighbor Contribution Function

```
def computeContribs(neighbors, rank):
  for neighbor in neighbors:
    yield(neighbor, rank/len(neighbors))
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

PageRank in Spark: Example Data

Data Format:

```
source-page destination-page
…
```

```
page1 page3
page2 page1
page4 page1
page3 page1
page4 page2
page3 page4
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

PageRank in Spark: Pairs of Page Links

```
def computeContribs(neighbors, rank):…

links = sc.textFile(file). \
  map(lambda line: line.split()). \
  map(lambda pages: \
    (pages[0],pages[1])). \
  distinct()
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

PageRank in Spark: Page Links Grouped by Source Page

```
def computeContribs(neighbors, rank):…

links = sc.textFile(file). \
  map(lambda line: line.split()). \
  map(lambda pages: \
    (pages[0],pages[1])). \
  distinct(). \
  groupByKey()
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

PageRank in Spark: Persisting the Link Pair RDD

```
def computeContribs(neighbors, rank):…

links = sc.textFile(file). \
  map(lambda line: line.split()). \
  map(lambda pages: \
    (pages[0],pages[1])). \
  distinct(). \
  groupByKey(). \
  persist()
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

PageRank in Spark: Set Initial Ranks

```
def computeContribs(neighbors, rank):…

links = sc.textFile(file). \
  map(lambda line: line.split()). \
  map(lambda pages: \
    (pages[0],pages[1])). \
  distinct(). \
  groupByKey(). \
  persist()

ranks=links.map(lambda (page,neighbors):
  (page,1.0))
```

Language: *Python*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

PageRank in Spark: First Iteration (1)

```
def computeContribs(neighbors, rank):…

links = …

ranks = …

for x in xrange(10):
  contribs=links. \
    join(ranks)
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

PageRank in Spark: First Iteration (2)

```
def computeContribs(neighbors, rank):…

links = …

ranks = …

for x in xrange(10):
  contribs=links. \
    join(ranks). \
    flatMap(lambda \
      (page,(neighbors,rank)): \
        computeContribs(neighbors,rank))
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

PageRank in Spark: First Iteration (3)

```
def computeContribs(neighbors, rank):…

links = …

ranks = …

for x in xrange(10):
  contribs=links. \
    join(ranks). \
    flatMap(lambda \
      (page,(neighbors,rank)): \
        computeContribs(neighbors,rank))
  ranks=contribs. \
    reduceByKey(lambda v1,v2: v1+v2)
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

PageRank in Spark: First Iteration (4)

```
def computeContribs(neighbors, rank):…

links = …

ranks = …

for x in xrange(10):
  contribs=links. \
    join(ranks). \
    flatMap(lambda \
      (page,(neighbors,rank)): \
        computeContribs(neighbors,rank))
  ranks=contribs. \
    reduceByKey(lambda v1,v2: v1+v2) .
    map(lambda (page,contrib): \
         (page,contrib * 0.85 + 0.15))
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

PageRank in Spark: Second Iteration

```
def computeContribs(neighbors, rank):…

links = …

ranks = …

for x in xrange(10):
  contribs=links. \
    join(ranks). \
    flatMap(lambda \
      (page,(neighbors,rank)): \
        computeContribs(neighbors,rank))
  ranks=contribs. \
    reduceByKey(lambda v1,v2: v1+v2) .
    map(lambda (page,contrib): \
         (page,contrib * 0.85 + 0.15))

for rank in ranks.collect(): print rank
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Checkpointing (1)

- **In some cases, RDD lineages can get very long**
  - For example: iterative algorithms, streaming
- **Long lineages can cause problems**
  - Recovery might be very expensive
  - Potential stack overflow

```
myrdd = …initial-value…
for i in xrange(100):
  myrdd = myrdd.transform(…)
myrdd.saveAsTextFile(dir)
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Checkpointing (2)

- **Checkpointing saves the data to HDFS**
  - Provides fault-tolerant storage across nodes
- **Lineage is not saved**
- **Must be checkpointed before any actions on the RDD**

```
sc.setCheckpointDir(dir)
myrdd = …initial-value…
for x in xrange(100):
  myrdd = myrdd.transform(…)
  if x % 3 == 0:
    myrdd.checkpoint()
    myrdd.count()
myrdd.saveAsTextFile(dir)
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Common Patterns in Apache Spark Data Processing

- Common Apache Spark Use Cases
- Iterative Algorithms in Apache Spark
- **Machine Learning**
- Example: k-means
- Essential Points
- Hands-On Exercise: Implement an Iterative Algorithm with Apache Spark

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Fundamentals of Computer Programming

- **First consider how a typical program works**
  - Hardcoded conditional logic
  - Predefined reactions when those conditions are met

```
#!/usr/bin/env python

import sys
for line in sys.stdin:
  if "Make MONEY Fa$t At Home!!!" in line:
      print "This message is likely spam"
  if "Happy Birthday from Aunt Betty" in line:
      print "This message is probably OK"
```

- **The programmer must consider all possibilities at design time**
- **An alternative technique is to have computers *learn* what to do**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

What is Machine Learning?

- **Machine learning is a field within artificial intelligence (AI)**
  - AI: "The science and engineering of making intelligent machines"
- **Machine learning focuses on automated knowledge acquisition**
  - Primarily through the design and implementation of algorithms
  - These algorithms require empirical data as input
- **Machine learning algorithms "learn" from data and often produce a predictive model as their output**
  - Model can then be used to make predictions as new data arrives
- **For example, consider a predictive model based on credit card customers**
  - Build model with data about customers who did/did not default on debt
  - Model can then be used to predict whether new customers will default

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Types of Machine Learning

- **Three established categories of machine learning techniques:**
  - Collaborative filtering (recommendations)
  - Clustering
  - Classification

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

What is Collaborative Filtering?

- **Collaborative filtering is a technique for making recommendations**
- **Helps users find items of relevance**
  - Among a potentially vast number of choices
  - Based on comparison of preferences between users
  - Preferences can be either *explicit* (stated) or *implicit* (observed)

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Applications Involving Collaborative Filtering

- **Collaborative filtering is domain agnostic**
- **Can use the same algorithm to recommend practically anything**
  - Movies (Netflix, Amazon Instant Video)
  - Television (TiVO Suggestions)
  - Music (several popular music download and streaming services)
- **Amazon uses CF to recommend a variety of products**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

What is Clustering?

- **Clustering algorithms discover structure in collections of data**
  - Where no formal structure previously existed
- **They discover which clusters ("groupings") naturally occur in data**
  - By examining various properties of the input data
- **Clustering is often used for exploratory analysis**
  - Divide huge amount of data into smaller groups
  - Can then tune analysis for each group

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Unsupervised Learning (1)

- **Clustering is an example of *unsupervised learning***
  - Begin with a data set that has no apparent label
  - Use an algorithm to discover structure in the data

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Unsupervised Learning (2)

- **Once the model has been created, you can use it to assign groups**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Applications Involving Clustering

- **Market segmentation**
  - Group similar customers in order to target them effectively
- **Finding related news articles**
  - Google News
- **Epidemiological studies**
  - Identifying a “cancer cluster” and finding a root cause
- **Computer vision (groups of pixels that cohere into objects)**
  - Related pixels clustered to recognize faces or license plates

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

What is Classification?

- **Classification is a form of *supervised* learning**
  - This requires training with data that has known labels
  - A classifier can then label new data based on what it learned in training
- **This example depicts how a classifier might identify animals**
  - In this case, it learned to distinguish between these two classes of animals based on height and weight

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Supervised Learning (1)

- **Classification is an example of supervised learning**
  - Begin with a data set that includes the value to be predicted (the label)
  - Use an algorithm to train a predictive model using the data-label pairs

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Supervised Learning (2)

- **Once the model has been trained, you can make predictions**
  - This will take new (previously unseen) data as input
  - The new data will not have labels

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Applications Involving Classification

- **Spam filtering**
  - Train using a set of spam and non-spam messages
  - System will eventually learn to detect unwanted email
- **Oncology**
  - Train using images of benign and malignant tumors
  - System will eventually learn to identify cancer
- **Risk Analysis**
  - Train using financial records of customers who do/don't default
  - System will eventually learn to identify risk customers

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Relationship of Algorithms and Data Volume (1)

- **There are many algorithms for each type of machine learning**
  - There's no overall "best" algorithm
  - Each algorithm has advantages and limitations
- **Algorithm choice is often related to data volume**
  - Some scale better than others
- **Most algorithms offer better results as volume increases**
  - Best approach = simple algorithm + lots of data
- **Spark is an excellent platform for machine learning over large data sets**
  - Resilient, iterative, parallel computations over distributed data sets

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Relationship of Algorithms and Data Volume (2)

It's not who has the best algorithms that wins. It's who has the most data.

—Banko and Brill, 2001

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Machine Learning Challenges

- **Highly computation-intensive and iterative**
- **Many traditional numerical processing systems do not scale directly to very large datasets**
  - Some make use of Spark to bridge the gap, such as MATLAB®

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark MLlib and Spark ML

- **Spark MLlib is a Spark machine learning library**
  - Makes practical machine learning scalable and easy
  - Includes many common machine learning algorithms
  - Includes base data types for efficient calculations at scale
  - Supports scalable statistics and data transformations
- **Spark ML is a new higher-level API for machine learning pipelines**
  - Built on top of Spark's DataFrames API
  - Simple and clean interface for running a series of complex tasks
  - Supports most functionality included in Spark MLlib
- **Spark MLlib and ML support a variety of machine learning algorithms**
  - Such as ALS (alternating least squares), k-means, linear regression, logistic regression, gradient descent

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Common Patterns in Apache Spark Data Processing

- Common Apache Spark Use Cases
- Iterative Algorithms in Apache Spark
- Machine Learning
- **Example: k-means**
- Essential Points
- Hands-On Exercise: Implement an Iterative Algorithm with Apache Spark

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

k-means Clustering

- **k-means clustering**
  - A common iterative algorithm used in graph analysis and machine learning
  - You will implement a simplified version in the hands-on exercises

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Clustering (1)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Clustering (2)

Goal: Find “clusters” of data points

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: k-means Clustering (1)

1. Choose *k* random points as starting centers

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: k-means Clustering (2)

1. Choose *k* random points as starting centers
2. Find all points closest to each center

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: k-means Clustering (3)

1. Choose *k* random points as starting centers
2. Find all points closest to each center
3. Find the center (mean) of each cluster

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: k-means Clustering (4)

1. Choose *k* random points as starting centers
2. Find all points closest to each center
3. Find the center (mean) of each cluster
4. If the centers changed, iterate again

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: k-means Clustering (5)

1. Choose *k* random points as starting centers
2. Find all points closest to each center
3. Find the center (mean) of each cluster
4. If the centers changed, iterate again

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: k-means Clustering (6)

1. Choose *k* random points as starting centers
2. Find all points closest to each center
3. Find the center (mean) of each cluster
4. If the centers changed, iterate again

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: k-means Clustering (7)

1. Choose *k* random points as starting centers
2. Find all points closest to each center
3. Find the center (mean) of each cluster
4. If the centers changed, iterate again

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: k-means Clustering (8)

1. Choose *k* random points as starting centers
2. Find all points closest to each center
3. Find the center (mean) of each cluster
4. If the centers changed, iterate again

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: k-means Clustering (9)

1. Choose *k* random points as starting centers
2. Find all points closest to each center
3. Find the center (mean) of each cluster
4. If the centers changed, iterate again

   …
5. Done!

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Approximate k-means Clustering

1. Choose *k* random points as starting centers
2. Find all points closest to each center
3. Find the center (mean) of each cluster
4. If the centers changed by more than *c*, iterate again …
5. Close enough!

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Common Patterns in Apache Spark Data Processing

- Common Apache Spark Use Cases
- Iterative Algorithms in Apache Spark
- Machine Learning
- Example: k-means
- **Essential Points**
- Hands-On Exercise: Implement an Iterative Algorithm with Apache Spark

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **Spark is especially suited to big data problems that require iteration**
  - In-memory persistence makes this very efficient
- **Common in many types of analysis**
  - For example, common algorithms such as PageRank and k-means
- **Spark is especially well-suited for implementing machine learning**
- **Spark includes MLlib and ML**
  - Specialized libraries to implement many common machine learning functions
  - Efficient, scalable functions for machine learning (for example: logistic regression, k-means)

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Common Patterns in Apache Spark Data Processing

- Common Apache Spark Use Cases
- Iterative Algorithms in Apache Spark
- Machine Learning
- Example: k-means
- Essential Points
- **Hands-On Exercise: Implement an Iterative Algorithm with Apache Spark**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Implementing an Iterative Algorithm with Apache Spark

- **In this exercise, you will implement the k-means algorithm in Spark**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

cloudera®

Apache Spark Streaming: Introduction to DStreams

Chapter 17

cloudera®

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- **Apache Spark Streaming: Introduction to DStreams**
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Apache Spark Streaming: Introduction to DStreams

In this chapter, you will learn

- **The features and typical use cases for Apache Spark Streaming**
- **How to write Spark Streaming applications**
- **How to create and operate on file and socket-based DStreams**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Streaming: Introduction to DStreams

- **Apache Spark Streaming Overview**
- Example: Streaming Request Count
- DStreams
- Developing Streaming Applications
- Essential Points
- Hands-On Exercise: Writing a Streaming Application

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

What Is Spark Streaming?

- An extension of core Spark
- Provides real-time data processing
- Segments an incoming stream of data into micro-batches

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Why Spark Streaming?

- **Many big data applications need to process large data streams in real time, such as**
  - Continuous ETL
  - Website monitoring
  - Fraud detection
  - Advertisement monetization
  - Social media analysis
  - Financial market trends

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark Streaming Features

- Latencies of a few seconds or less
- Scalability and efficient fault tolerance
- “Once and only once” processing
- Integrates batch and real-time processing
- Uses the core Spark RDD API

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark Streaming Overview

DStream—RDDs (batches of *n* seconds)

- **A *DStream* (Discretized Stream) divides a data stream into batches of *n* seconds**
- **Processes each batch in Spark as an RDD**
- **Returns results of RDD operations in batches**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Streaming: Introduction to DStreams

- Apache Spark Streaming Overview
- **Example: Streaming Request Count**
- DStreams
- Developing Streaming Applications
- Essential Points
- Hands-On Exercise: Writing a Streaming Application

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Streaming Request Count Overview (Scala)

```
object StreamingRequestCount {

 def main(args: Array[String]) {
  val ssc = new StreamingContext(new SparkContext(),Seconds(2))
  val mystream = ssc.socketTextStream(hostname, port)
  val userreqs = mystream.map(line => (line.split(' ')(2),1))
     .reduceByKey((x,y) => x+y)

  userreqs.print()

  ssc.start()
  ssc.awaitTermination()
 }
}
```

Language: *Scala*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Configuring the Streaming Context

- **The `StreamingContext` class is the entry point for Spark Streaming applications**
- **Create a Streaming context using the Spark context and a batch duration**
  - Duration can be in `Milliseconds`, `Seconds`, or `Minutes`

```
object StreamingRequestCount {

 def main(args: Array[String]) {
  val ssc = new StreamingContext(new SparkContext(),Seconds(2))
  val mystream = ssc.socketTextStream(hostname, port)
  val userreqs = mystream.map(line => (line.split(' ')(2),1))
   .reduceByKey((x,y) => x+y)

  userreqs.print

  ssc.start
  ssc.awaitTermination
 }
}
```

Language: *Scala*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Streaming Example: Creating a DStream

- **Create a DStream from a streaming data source**
  - For example, text from a socket

```
object StreamingRequestCount {

 def main(args: Array[String]) {
  val ssc = new StreamingContext(new SparkContext(),Seconds(2))
  val mystream = ssc.socketTextStream(hostname, port)
  val userreqs = mystream.map(line => (line.split(' ')(2),1))
   .reduceByKey((x,y) => x+y)

  userreqs.print

  ssc.start
  ssc.awaitTermination
 }
}
```

Language: *Scala*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Streaming Example: DStream Transformations

- **DStream operations are applied to each batch RDD in the stream.**
  - Similar to RDD operations—`filter`, `map`, `reduceByKey`, `joinByKey`, and so on

```
object StreamingRequestCount {

 def main(args: Array[String]) {
  val ssc = new StreamingContext(new SparkContext(),Seconds(2))
  val mystream = ssc.socketTextStream(hostname, port)
  val userreqs = mystream.map(line => (line.split(' ')(2),1))
   .reduceByKey((x,y) => x+y)

  userreqs.print

  ssc.start
  ssc.awaitTermination
 }
}
```

Language: *Scala*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Streaming Example: DStream Result Output

- **The DStream `print` function displays transformation results**
  - Prints first 10 items by default

```
object StreamingRequestCount {

 def main(args: Array[String]) {
  val ssc = new StreamingContext(new SparkContext(),Seconds(2))
  val mystream = ssc.socketTextStream(hostname, port)
  val userreqs = mystream.map(line => (line.split(' ')(2),1))
   .reduceByKey((x,y) => x+y)

  userreqs.print

  ssc.start
  ssc.awaitTermination
 }
}
```

Language: *Scala*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Streaming Example: Starting the Streams

- `start` starts the execution of all DStreams in a Streaming context
- `awaitTermination` waits for all background threads to complete before ending the main thread

```
object StreamingRequestCount {

 def main(args: Array[String]) {
  val ssc = new StreamingContext(new SparkContext(),Seconds(2))
  val mystream = ssc.socketTextStream(hostname, port)
  val userreqs = mystream.map(line => (line.split(' ')(2),1))
   .reduceByKey((x,y) => x+y)

  userreqs.print

  ssc.start
  ssc.awaitTermination
 }
}
```

Language: *Scala*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Streaming Example: Python versus Scala

- Durations are expressed in seconds
- Use `pprint` instead of `print`

```
if __name__ == "__main__":

  ssc = StreamingContext(SparkContext(),2)

  mystream = ssc.socketTextStream(hostname, port)
  userreqs = mystream. \
    map(lambda line: (line.split(' ')[2],1)). \
    reduceByKey(lambda v1,v2: v1+v2)

  userreqs.pprint()

  ssc.start()
  ssc.awaitTermination()
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Streaming Example: Streaming Request Count (Recap)

```
object StreamingRequestCount {

 def main(args: Array[String]) {
  val sc = new SparkContext()
  val ssc = new StreamingContext(sc,Seconds(2))
  val mystream = ssc.socketTextStream(hostname, port)
  val userreqs = mystream
   .map(line => (line.split(' ')(2),1))
   .reduceByKey((x,y) => x+y)

  userreqs.print()

  ssc.start()
  ssc.awaitTermination()
 }
}
```

Language: *Scala*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Streaming Example Output (1)

```
-------------------------------------------
Time: 1401219545000 ms ①
-------------------------------------------
(23713,2)
(53,2)
(24433,2)
(127,2)
...
```

① **Two seconds after `ssc.start` (time interval t1)**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Streaming Example Output (2)

```
-------------------------------------------
Time: 1401219545000 ms
-------------------------------------------
(23713,2)
(53,2)
(24433,2)
(127,2)
...
-------------------------------------------
Time: 1401219547000 ms ①
-------------------------------------------
(42400,2)
(24996,2)
(97464,2)
(161,2)
...
```

① **t2 (two seconds later…)**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Streaming Example Output (3)

```
-------------------------------------------
Time: 1401219545000 ms
-------------------------------------------
(127,2)
(93,2)
...
-------------------------------------------
Time: 1401219547000 ms
-------------------------------------------
(161,2)
(6011,2)
...
-------------------------------------------
Time: 1401219549000 ms ①
-------------------------------------------
(465,2)
(120,2)
...
```

① **t3 (two seconds later…)—Continues until termination…**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Streaming: Introduction to DStreams

- Apache Spark Streaming Overview
- Example: Streaming Request Count
- **DStreams**
- Developing Streaming Applications
- Essential Points
- Hands-On Exercise: Writing a Streaming Application

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DStreams

- **A DStream is a sequence of RDDs representing a data stream**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DStream Data Sources

- **DStreams are defined for a given input stream (such as a Unix socket)**
  - Created by the Streaming context

```
ssc.socketTextStream(hostname, port)
```

  - Similar to how RDDs are created by the Spark context
- **Out-of-the-box data sources**
  - Network
    - Sockets
    - Services such as Apache Flume, Apache Kafka, ZeroMQ, or Amazon Kinesis
  - Files
    - Monitors an HDFS directory for new content

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DStream Operations

- **DStream operations are applied to every RDD in the stream**
  - Executed once per *duration*
- **Two types of DStream operations**
  - Transformations
    - Create a new DStream from an existing one
  - Output operations
    - Write data (for example, to a file system, database, or console)
    - Similar to RDD *actions*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DStream Transformations (1)

- **Many RDD transformations are also available on DStreams**
  - General transformations such as `map`, `flatMap`, `filter`
  - Pair transformations such as `reduceByKey`, `groupByKey`, `join`
- **What if you want to do something else?**
  - `transform(`*`function`*`)`
  - Creates a new DStream by executing *function* on RDDs in the current DStream

```
val distinctDS = myDS.
  transform(rdd => rdd.distinct())
```

Language: *Scala*

```
distinctDS = myDS.\
  transform(lambda rdd: rdd.distinct())
```

Language: *Python*

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DStream Transformations (2)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

DStream Output Operations

- **Console output**
  - `print` **(Scala)** /`pprint` **(Python)** prints out the first 10 elements of each RDD
    - Optionally pass an integer to print another number of elements
- **File output**
  - `saveAsTextFiles` saves data as text
  - `saveAsObjectFiles` saves as serialized object files (SequenceFiles)
- **Executing other functions**
  - `foreachRDD(`*`function`*`)` performs a function on each RDD in the DStream
  - Function input parameters
    - The RDD on which to perform the function
    - The timestamp of the RDD (optional)

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Saving DStream Results as Files

```
val userreqs = logs.
  map(line => (line.split(' ')(2),1)).
  reduceByKey((v1,v2) => v1+v2)
userreqs.saveAsTextFiles("…/outdir/reqcounts")
```

Language: *Scala*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Scala Example: Find Top Users (1)

```
…
val userreqs = logs.
  map(line => (line.split(' ')(2),1)).
  reduceByKey((v1,v2) => v1+v2)
userreqs.saveAsTextFiles(path)

val sortedreqs = userreqs.
  map(pair => pair.swap).
  transform(rdd => rdd.sortByKey(false))①

sortedreqs.foreachRDD((rdd,time) => {
 println("Top users @ " + time)
 rdd.take(5).foreach(
  pair => printf("User: %s (%s)\n",pair._2, pair._1))})

ssc.start
ssc.awaitTermination
…
```

① **Transform each RDD: swap user ID with count, sort by count**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Scala Example: Find Top Users (2)

```
…
val userreqs = logs.
  map(line => (line.split(' ')(2),1)).
  reduceByKey((v1,v2) => v1+v2)
userreqs.saveAsTextFiles(path)

val sortedreqs = userreqs.
  map(pair => pair.swap).
  transform(rdd => rdd.sortByKey(false))

sortedreqs.foreachRDD((rdd,time) => { ①
 println("Top users @ " + time)
 rdd.take(5).foreach(
  pair => printf("User: %s (%s)\n",pair._2, pair._1))})

ssc.start
ssc.awaitTermination
…
```

① **Print out the top 5 users as "`User: userID (count)`"**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Python Example: Find Top Users (1)

```
def printTop5(rdd,time):
 print "Top users @",time
 for count,user in rdd.take(5):
   print "User:",user,"("+str(count)+")"
…
userreqs = mystream. \
  map(lambda line: (line.split(' ')[2],1)). \
  reduceByKey(lambda v1,v2: v1+v2)
userreqs.saveAsTextFiles(path)

sortedreqs=userreqs.map(lambda (k,v): (v,k)). \   ①
  transform(lambda rdd: rdd.sortByKey(False))

sortedreqs.foreachRDD(lambda time,rdd: printTop5(rdd,time))

ssc.start()
ssc.awaitTermination()
…
```

① **Transform each RDD: swap user ID with count, sort by count**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Python Example: Find Top Users (2)

```
…
userreqs = mystream. \
  map(lambda line: (line.split(' ')[2],1)). \
  reduceByKey(lambda v1,v2: v1+v2)
userreqs.saveAsTextFiles(path)

sortedreqs=userreqs.map(lambda (k,v): (v,k)). \
  transform(lambda rdd: rdd.sortByKey(False))

sortedreqs.foreachRDD(lambda time,rdd: printTop5(rdd,time)) ①

ssc.start()
ssc.awaitTermination()
…

def printTop5(r,t) ①
 print "Top users @",t
 for count,user in r.take(5):
   print "User:",user,"("+str(count)+")"
```

① **Print out the top 5 users as "User: *userID* (*count*)"**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Find Top Users—Output (1)

```
Top users @ 1401219545000 ms ①
User: 16261 (8)
User: 22232 (7)
User: 66652 (4)
User: 21205 (2)
User: 24358 (2)
```

① **t1 (2 seconds after `ssc.start`)**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Find Top Users—Output (2)

```
Top users @ 1401219545000 ms
User: 16261 (8)
User: 22232 (7)
User: 66652 (4)
User: 21205 (2)
User: 24358 (2)
Top users @ 1401219547000 ms ①
User: 53667 (4)
User: 35600 (4)
User: 62 (2)
User: 165 (2)
User: 40 (2)
```

① **t2 (2 seconds later)**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Find Top Users—Output (3)

```
Top users @ 1401219545000 ms
User: 16261 (8)
User: 22232 (7)
User: 66652 (4)
User: 21205 (2)
User: 24358 (2)
Top users @ 1401219547000 ms
User: 53667 (4)
User: 35600 (4)
User: 62 (2)
User: 165 (2)
User: 40 (2)
Top users @ 1401219549000 ms (1)
User: 31 (12)
User: 6734 (10)
User: 14986 (10)
…
```

(1) **t3 (2 seconds later)—Continues until termination...**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Streaming: Introduction to DStreams

- Apache Spark Streaming Overview
- Example: Streaming Request Count
- DStreams
- **Developing Streaming Applications**
- Essential Points
- Hands-On Exercise: Writing a Streaming Application

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Building and Running Spark Streaming Applications

- **Building Spark Streaming applications**
  - Include the main Spark Streaming library (included with Spark)
  - Include additional Spark Streaming libraries if necessary, for example, Kafka, Flume, Twitter
- **Running Spark Streaming applications**
  - Cluster must have at least two cores available
    - Use at least two threads if running locally
  - Adding operations after the Streaming context has been started is unsupported
  - Stopping and restarting the Streaming context is unsupported

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Using Spark Streaming with Spark Shell

- **Spark Streaming is designed for batch applications, not interactive use**
- **The Spark shell can be used for limited testing— *not intended for production use!***
  - Be sure to run the shell on a cluster with at least two cores available
    - Or locally with at least two threads

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

The Spark Streaming Application UI

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Streaming: Introduction to DStreams

- Apache Spark Streaming Overview
- Example: Streaming Request Count
- DStreams
- Developing Streaming Applications
- **Essential Points**
- Hands-On Exercise: Writing a Streaming Application

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **Spark Streaming is an extension of core Spark to process real-time streaming data**
- **DStreams are *discretized streams* of streaming data, batched into RDDs by time intervals**
  - Operations applied to DStreams are applied to each RDD
  - Transformations produce new DStreams by applying a function to each RDD in the base DStream

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Streaming: Introduction to DStreams

- Apache Spark Streaming Overview
- Example: Streaming Request Count
- DStreams
- Developing Streaming Applications
- Essential Points
- **Hands-On Exercise: Writing a Streaming Application**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Writing a Streaming Application

- **In this exercise, you will write a Spark Streaming application to process web log data**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

cloudera®

Apache Spark Streaming: Processing Multiple Batches

Chapter 18

cloudera®

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- **Apache Spark Streaming: Processing Multiple Batches**
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Apache Spark Streaming: Processing Multiple Batches

In this chapter, you will learn

- **How to return data from a specific time period in a DStream**
- **How to perform analysis using sliding window operations on a DStream**
- **How to maintain state values across all time periods in a DStream**
- **Some basic concepts of the Structured Streaming API**
  - The next generation streaming framework for Spark

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Streaming: Processing Multiple Batches

- **Multi-Batch Operations**
- Time Slicing
- State Operations
- Sliding Window Operations
- Preview: Structured Streaming
- Essential Points
- Hands-On Exercise: Processing Multiple Batches of Streaming Data

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Multi-Batch DStream Operations

- **DStreams consist of a series of "batches" of data**
  - Each batch is an RDD
- **Basic DStream operations analyze each batch individually**
- **Advanced operations allow you to analyze data collected across batches**
  - Slice: allows you to operate on a collection of batches
  - State: allows you to perform cumulative operations
  - Windows: allows you to aggregate data across a sliding time period

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Streaming: Processing Multiple Batches

- Multi-Batch Operations
- **Time Slicing**
- State Operations
- Sliding Window Operations
- Preview: Structured Streaming
- Essential Points
- Hands-On Exercise: Processing Multiple Batches of Streaming Data

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Time Slicing

- `DStream.slice(`*`fromTime,toTime`*`)`
  - Returns a collection of batch RDDs based on data from the stream
- `StreamingContext.remember(`*`duration`*`)`
  - By default, input data is automatically cleared when no RDD's lineage depends on it
  - `slice` will return no data for time periods for which data has already been cleared
  - Use `remember` to keep data around longer

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Streaming: Processing Multiple Batches

- Multi-Batch Operations
- Time Slicing
- **State Operations**
- Sliding Window Operations
- Preview: Structured Streaming
- Essential Points
- Hands-On Exercise: Processing Multiple Batches of Streaming Data

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

State DStreams (1)

- Use the `updateStateByKey` function to create a state DStream
- Example: Total request count by User ID

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

State DStreams (2)

- Use the `updateStateByKey` function to create a state DStream
- Example: Total request count by User ID

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

State DStreams (3)

- Use the `updateStateByKey` function to create a state DStream
- Example: Total request count by User ID

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Python Example: Total User Request Count (1)

```
…
userreqs = logs \
 .map(lambda line: (line.split(' ')[2],1)) \
 .reduceByKey(lambda v1,v2: v1+v2)
…
ssc.checkpoint("checkpoints") ①

totalUserreqs = userreqs. \
  updateStateByKey(lambda newCounts, state: \
    updateCount(newCounts, state))
totalUserreqs.pprint()

ssc.start()
ssc.awaitTermination()…
```

Language: *Python*

① **Set checkpoint directory to enable checkpointing. Required to prevent infinite lineages.**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Python Example: Total User Request Count (2)

```
…
userreqs = logs. \
  map(lambda line: (line.split(' ')[2],1)). \
  reduceByKey(lambda v1,v2: v1+v2)
…
ssc.checkpoint("checkpoints")

totalUserreqs = userreqs. \ ①
  updateStateByKey(lambda newCounts, state: \
    updateCount(newCounts, state)) ②
totalUserreqs.pprint()

ssc.start()
ssc.awaitTermination() …
```

Language: *Python*

① **Compute a state DStream based on the previous states, updated with the values from the current batch of request counts.**

② **Next slide...**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Python Example: Total User Request Count—Update Function

```
def updateCount(newCounts①, state②):
  if state == None:
    return sum(newCounts)
  else:
    return state + sum(newCounts) ③
```

① **New Values**

② **Current State (or None)**

③ **New State**

Example at t2

```
user001:
updateCount([4],5) → 9
user012:
updateCount([2],None)) → 2
user921:
updateCount([5],None)) → 5
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Scala Example: Total User Request Count (1)

```
…
val userreqs = logs
 .map(line => (line.split(' ')(2),1))
 .reduceByKey((x,y) => x+y)
…
ssc.checkpoint("checkpoints") ①

val totalUserreqs = userreqs.updateStateByKey(updateCount)
totalUserreqs.print

ssc.start
ssc.awaitTermination
…
```

Language: *Scala*

① **Set checkpoint directory to enable checkpointing. Required to prevent infinite lineages.**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Scala Example: Total User Request Count (2)

```
…
val userreqs = logs
 .map(line => (line.split(' ')(2),1))
 .reduceByKey((x,y) => x+y)
…
ssc.checkpoint("checkpoints")

val totalUserreqs = userreqs.updateStateByKey(updateCount) ①
totalUserreqs.print ②

ssc.start
ssc.awaitTermination
…
```

Language: *Scala*

① **Next slide...**

② **Compute a state DStream based on the previous states, updated with the values from the current batch of request counts.**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Scala Example: Total User Request Count—Update Function

```
def updateCount = (newCounts: Seq[Int]①,
                   state: Option[Int]②) => {
  val newCount = newCounts.foldLeft(0)(_+_)
  val previousCount = state.getOrElse(0)
  Some(newCount + previousCount)③
}
```

① New Values
② Current State (or None)
③ New State

Example at t2

```
user001:
updateCount([4],5) → 9
user012:
updateCount([2],None)) → 2
user921:
updateCount([5],None)) → 5
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Maintaining State—Output

```
-------------------------------------------
Time: 2017-08-28 05:56:45
-------------------------------------------
(user001,5)
(user102,1)
(user009,2)
-------------------------------------------
Time: 2017-08-28 05:56:47
-------------------------------------------
(user001,9)
(user102,1)
(user009,2)
(user012,2)
(user921,5)
-------------------------------------------
Time: 2017-08-28 05:56:49
-------------------------------------------
(user001,9)
(user102,8)
(user009,2)
(user012,5)
(user921,5)
(user660,4)
-------------------------------------------
Time: 2017-08-28 05:56:51
-------------------------------------------
…
```

t1

(user001,5)
(user102,1)
(user009,2)

t2

(user001,9)
(user102,1)
(user009,2)
(user012,2)
(user921,5)

t3

(user001,9)
(user102,8)
(user009,2)
(user012,5)
(user921,5)
(user660,4)

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Streaming: Processing Multiple Batches

- Multi-Batch Operations
- Time Slicing
- State Operations
- **Sliding Window Operations**
- Preview: Structured Streaming
- Essential Points
- Hands-On Exercise: Processing Multiple Batches of Streaming Data

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Sliding Window Operations (1)

- **Regular DStream operations execute for each RDD based on SSC duration**
- **"Window" operations span RDDs over a given duration**
  - For example `reduceByKeyAndWindow`, `countByWindow`

```
reduceByKeyAndWindow(fn, Seconds(12))
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Sliding Window Operations (2)

- **By default, window operations will execute with an "interval" the same as the SSC duration**
  - For two-second batch duration, window will "slide" every two seconds

```
reduceByKeyAndWindow(fn, Seconds(12))
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Sliding Window Operations (3)

- **You can specify a different slide duration (must be a multiple of the SSC duration)**

```
reduceByKeyAndWindow(fn, Seconds(12), Seconds(4))
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Scala Example: Count and Sort User Requests by Window (1)

```
val ssc = new StreamingContext(new SparkConf(), Seconds(2))
val logs = ssc.socketTextStream(hostname, port)
…
val reqcountsByWindow = logs.
  map(line => (line.split(' ')(2),1)).
  reduceByKeyAndWindow((v1: Int, v2: Int) => v1+v2,
    Minutes(5),Seconds(30)) ①

val topreqsByWindow=reqcountsByWindow.
  map(pair => pair.swap).
  transform(rdd => rdd.sortByKey(false))
topreqsByWindow.map(pair => pair.swap).print

ssc.start
ssc.awaitTermination
```

Language: *Scala*

① **Every 30 seconds, count requests by user over the last five minutes.**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Scala Example: Count and Sort User Requests by Window (2)

```
val ssc = new StreamingContext(new SparkConf(), Seconds(2))
val logs = ssc.socketTextStream(hostname, port)
…
val reqcountsByWindow = logs.
  map(line => (line.split(' ')(2),1)).
  reduceByKeyAndWindow((v1: Int, v2: Int) =>
    v1+v2,  Minutes(5),Seconds(30))

val topreqsByWindow=reqcountsByWindow. ①
  map(pair => pair.swap).
  transform(rdd => rdd.sortByKey(false))
topreqsByWindow.map(pair => pair.swap).print

ssc.start
ssc.awaitTermination
```

Language: *Scala*

① **Sort and print the top users for every RDD (every 30 seconds).**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Python Example: Count and Sort User Requests by Window (1)

```
ssc = new StreamingContext(new SparkConf(), 2)
logs = ssc.socketTextStream(hostname, port)
…
reqcountsByWindow = logs. \
  map(lambda line: (line.split(' ')[2],1)). \
  reduceByKeyAndWindow(lambda v1,v2: v1+v2, 5*60, 30) ①

topreqsByWindow=reqcountsByWindow. \
  map(lambda (k,v): (v,k)). \
  transform(lambda rdd: rdd.sortByKey(False))
topreqsByWindow.map(lambda (k,v): (v,k)).pprint()

ssc.start()
ssc.awaitTermination()
```

Language: *Python*

① **Every 30 seconds, count requests by user over the last five minutes.**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Python Example: Count and Sort User Requests by Window (2)

```
ssc = new StreamingContext(new SparkConf(), 2)
logs = ssc.socketTextStream(hostname, port)
…
reqcountsByWindow = logs. \
 map(lambda line: (line.split(' ')[2],1)). \
 reduceByKeyAndWindow(lambda v1,v2: v1+v2,5*60,30)

topreqsByWindow=reqcountsByWindow.① \
  map(lambda (k,v): (v,k)). \ #swap
  transform(lambda rdd: rdd.sortByKey(False))
topreqsByWindow.map(lambda (k,v): (v,k)).pprint()

ssc.start()
ssc.awaitTermination()
```

Language: *Python*

① **Sort and print the top users for every RDD (every 30 seconds).**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Streaming: Processing Multiple Batches

- Multi-Batch Operations
- Time Slicing
- State Operations
- Sliding Window Operations
- **Preview: Structured Streaming**
- Essential Points
- Hands-On Exercise: Processing Multiple Batches of Streaming Data

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Preview: Structured Streaming

- **Spark Structured Streaming is a new high-level API for continuous streaming applications**
  - Uses the DataFrames and Datasets API with streaming data
  - Addresses shortcomings in Spark Streaming with building end-to-end applications
  - Integrates storage, serving systems, and batch jobs
  - Provides consistency across systems and handles out-of-order events
- **Alpha version (experimental) introduced in Spark 2.0**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Structured Streaming (1)

```
peopleSchema = StructType([
  StructField("lastName", StringType()),
  StructField("firstName", StringType()),
  StructField("pcode", StringType())])

peopleDF = spark.readStream. \
  schema(peopleSchema).csv("people")
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Structured Streaming (2)

```
peopleSchema = StructType([
  StructField("lastName", StringType()),
  StructField("firstName", StringType()),
  StructField("pcode", StringType())])

peopleDF = spark.readStream. \
  schema(peopleSchema).csv("people")

pcodeCountsDF = peopleDF.groupBy("pcode").count()

countsQuery = pcodeCountsDF.writeStream. \
  outputMode("complete").format("console").start()

countsQuery.awaitTermination()
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Structured Streaming Output

```
-------------
Batch: 0
-------------
+-----+-----+
|pcode|count|
+-----+-----+
|91910|    3|
|97128|    4|
|91788|    3|
...
-------------
Batch: 1
-------------
+-----+-----+
|pcode|count|
+-----+-----+
|91910|   11|
|97128|   12|
|91788|   13|
...
```

```
-------------
Batch: 2
-------------
+-----+-----+
|pcode|count|
+-----+-----+
|91910|   11|
|97128|   15|
|91788|   22|
...
-------------
Batch: 3
-------------
+-----+-----+
|pcode|count|
+-----+-----+
|91910|   12|
|97128|   20|
|91788|   22|
...
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Streaming: Processing Multiple Batches

- Multi-Batch Operations
- Time Slicing
- State Operations
- Sliding Window Operations
- Preview: Structured Streaming
- **Essential Points**
- Hands-On Exercise: Processing Multiple Batches of Streaming Data

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **You can get a "slice" of data from a stream based on absolute start and end times**
  - For example, all data received between midnight October 1, 2016 and midnight October 2, 2016
- **You can update state based on prior state**
  - For example, total requests by user
- **You can perform operations on "windows" of data**
  - For example, number of logins in the last hour

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Streaming: Processing Multiple Batches

- Multi-Batch Operations
- Time Slicing
- State Operations
- Sliding Window Operations
- Preview: Structured Streaming
- Essential Points
- **Hands-On Exercise: Processing Multiple Batches of Streaming Data**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Processing Multiple Batches of Streaming Data

- **In this exercise, you will extend an Apache Spark Streaming application to perform multi-batch analysis on web log data**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Apache Spark Streaming: Data Sources

Chapter 19

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- **Apache Spark Streaming: Data Sources**
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Apache Spark Streaming: Data Sources

In this chapter, you will learn

- **How data sources are integrated with Spark Streaming**
- **How receiver-based integration differs from direct integration**
- **How Apache Flume and Apache Kafka are integrated with Spark Streaming**
- **How to use direct Kafka integration to create a DStream**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Streaming: Data Sources

- **Streaming Data Source Overview**
- Apache Flume and Apache Kafka Data Sources
- Example: Using a Kafka Direct Data Source
- Essential Points
- Hands-On Exercise: Processing Streaming Apache Kafka Messages

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark Streaming Data Sources

- **Basic data sources**
  - Network socket
  - Text file
- **Advanced data sources**
  - Apache Kafka
  - Apache Flume
  - Twitter
  - ZeroMQ
  - Amazon Kinesis
  - and more coming in the future…
- **To use advanced data sources, download (if necessary) and link to the required library**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Receiver-Based Data Sources

- **Most data sources are based on *receivers***
  - Network data is received on a worker node
  - Receiver distributes data (RDDs) to the cluster as partitions

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Receiver-Based Replication

- **Spark Streaming RDD replication is enabled by default**
  - Data is copied to another node as it received

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Receiver-Based Fault Tolerance

- **If the receiver fails, Spark will restart it on a different executor**
  - Potential for brief loss of incoming data

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Managing Incoming Data

- **Receivers queue jobs to process data as it arrives**
- **Data must be processed fast enough that the job queue does not grow**
  - Manage by setting `spark.streaming.backpressure.enabled = true`
- **Monitor scheduling delay and processing time in Spark UI**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Streaming: Data Sources

- Streaming Data Source Overview
- **Apache Flume and Apache Kafka Data Sources**
- Example: Using a Kafka Direct Data Source
- Essential Points
- Hands-On Exercise: Processing Streaming Apache Kafka Messages

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Apache Flume Basics

- **Apache Flume is a scalable, extensible, real-time data collection system**
  - Originally designed to ingest log files from multiple servers into HDFS
- **A Flume *event* includes a header (metadata) and a payload containing event data**
  - Example event: a line of web server log output
- **Flume agents are configured with a *source* and a *sink***
  - A source receives events from an external system
  - A sink sends events to their destination
- **Types of sources include server output, Kafka messages, HTTP requests, and files**
- **Types of sinks include Spark Streaming applications, HDFS directories, Apache Avro, and Kafka**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark Streaming with Flume

- **A Spark Streaming DStream can receive data from a Flume sink**
- **Two approaches to creating a Flume receiver**
  - Push-based
  - Pull-based
- **Push-based**
  - One Spark executor must run a network receiver on a specified host
  - Configure Flume with an Avro sink to send to that receiver on that host
- **Pull-based**
  - Uses a custom Flume sink in `spark.streaming.flume` package
  - Strong reliability and fault tolerance guarantees

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Kafka Basics (1)

- **Apache Kafka is a fast, scalable, distributed publish-subscribe messaging system that provides**
  - Durability by persisting data to disk
  - Fault tolerance through replication
- **A *message* is a single data record passed by Kafka**
- **One or more *brokers* in a cluster receive, store, and distribute messages**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Kafka Basics (2)

- **A *topic* is a named feed or queue of messages**
  - A Kafka cluster can include any number of topics
- ***Producers* are programs that publish (send) messages to a topic**
- ***Consumers* are programs that subscribe to (receive messages from) a topic**
- **A Spark Streaming application can be a Kafka producer, consumer, or both**
- **Kafka allows a topic to be *partitioned***
  - Topic partitions are handled by different brokers for scalability
  - Note that topic partitions are not related to RDD partitions

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Kafka Basics (3)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Spark Streaming with Kafka

- **You can use Kafka messages as a Spark Streaming data source**
  - A Kafka DStream is a message consumer
- **Two approaches to creating a Kafka DStream**
  - Receiver-based
  - Direct (receiverless)

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Receiver-Based Kafka Streams (1)

- **Receiver-based DStreams are configured with a Kafka topic and a partition in that topic**
- **Reliability is not guaranteed**
  - To protect from data loss, enable write-ahead logs
  - Offers reliability at the expense of performance

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Receiver-Based Kafka Streams (2)

- **Each receiver-based DStream is receives messages from specific topic partitions**
- **For scalability, create multiple DStreams for different partitions**
  - Union DStreams to process all the messages in a topic

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Direct Kafka Streams

- Consumes messages in parallel
- Automatically assigns each topic partition to an RDD partition
- Support for efficient zero-loss and exactly-once receipt
- Direct is also sometimes called *receiverless*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Streaming: Data Sources

- Streaming Data Source Overview
- Apache Flume and Apache Kafka Data Sources
- **Example: Using a Kafka Direct Data Source**
- Essential Points
- Hands-On Exercise: Processing Streaming Apache Kafka Messages

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Scala Example: Direct Kafka Stream (1)

```
import org.apache.spark.SparkContext
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.Seconds
import org.apache.spark.streaming.kafka._
import kafka.serializer.StringDecoder

object StreamingRequestCount {

  def main(args: Array[String]) {
    val ssc = new StreamingContext(new SparkContext(),
      Seconds(2))
```

Language: *Scala*

Example continues on next slide…

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Scala Example: Direct Kafka Stream (2)

```
val kafkaStream = KafkaUtils.createDirectStream
        [String,String,StringDecoder,StringDecoder] (ssc,
    Map("metadata.broker.list"->"broker1:port,broker2:port"),
    Set("mytopic"))

  val logs = kafkaStream.map(pair => pair._2)

  val userreqs = logs.
    map(line => (line.split(' ')(2),1)).
    reduceByKey((x,y) => x+y)

  userreqs.print

  ssc.start
  ssc.awaitTermination
 }
}
```

Language: *Scala*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Python Example: Direct Kafka Stream (1)

```
import sys
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils

if __name__ == "__main__":
  ssc = StreamingContext(SparkContext(),2)
```

Language: *Python*

Example continues on next slide…

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Python Example: Direct Kafka Stream (2)

```
kafkaStream = KafkaUtils. \
  createDirectStream(ssc, ["mytopic"], \
  {"metadata.broker.list": "broker1:port,broker2:port"})

  logs = kafkaStream.map(lambda (key,value): value)

  userreqs = logs. \
    map(lambda line: (line.split(' ')[2],1)). \
    reduceByKey(lambda v1,v2: v1+v2)

userreqs.pprint()

ssc.start()
ssc.awaitTermination()
```

Language: *Python*

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Streaming: Data Sources

- Streaming Data Source Overview
- Apache Flume and Apache Kafka Data Sources
- Example: Using a Kafka Direct Data Source
- **Essential Points**
- Hands-On Exercise: Processing Streaming Apache Kafka Messages

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **Spark Streaming integrates with a number of data sources**
- **Most use a receiver-based integration**
  - Flume, for example
- **Kafka can be integrated using a receiver-based or a direct (receiverless) approach**
  - The direct approach provides efficient strong reliability

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Apache Spark Streaming: Data Sources

- Streaming Data Source Overview
- Apache Flume and Apache Kafka Data Sources
- Example: Using a Kafka Direct Data Source
- Essential Points
- **Hands-On Exercise: Processing Streaming Apache Kafka Messages**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Processing Streaming Apache Kafka Messages

- **In this exercise, you will write a Spark Streaming application to process web logs using a direct Kafka data source**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Conclusion

Chapter 20

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- **Conclusion**
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Course Objectives

During this course, you have learned

- **How the Apache Hadoop ecosystem fits in with the data processing lifecycle**
- **How data is distributed, stored, and processed in a Hadoop cluster**
- **How to write, configure, and deploy Apache Spark applications on a Hadoop cluster**
- **How to use the Spark shell and Spark applications to explore, process, and analyze distributed data**
- **How to query data using Spark SQL, DataFrames, and Datasets**
- **How to use Spark Streaming to process a live data stream**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Which Course to Take Next

- **For developers**
  - *Cloudera Search Training*
  - *Cloudera Training for Apache HBase*
- **For system administrators**
  - *Cloudera Administrator Training for Apache Hadoop*
- **For data analysts and data scientists**
  - *Cloudera Data Analyst Training*
  - *Cloudera Data Scientist Training*

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Message Processing with Apache Kafka

Appendix A

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- **Appendix: Message Processing with Apache Kafka**
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Message Processing with Apache Kafka

In this appendix, you will learn

- **What Apache Kafka is and what advantages it offers**
- **About the high-level architecture of Kafka**
- **How to create topics, publish messages, and read messages from the command line**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Message Processing with Apache Kafka

- **What Is Apache Kafka?**
- Apache Kafka Overview
- Scaling Apache Kafka
- Apache Kafka Cluster Architecture
- Apache Kafka Command Line Tools
- Essential Points
- Hands-On Exercise: Producing and Consuming Apache Kafka Messages

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

What Is Apache Kafka?

- **Apache Kafka is a distributed commit log service**
  - Widely used for data ingest
  - Conceptually similar to a publish-subscribe messaging system
  - Offers scalability, performance, reliability, and flexibility
- **Originally created at LinkedIn, now an open source Apache project**
  - Donated to the Apache Software Foundation in 2011
  - Graduated from the Apache Incubator in 2012
  - Supported by Cloudera for production use with CDH in 2015

APACHE kafka™
A distributed streaming platform

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Characteristics of Kafka

- **Scalable**
  - Kafka is a distributed system that supports multiple nodes
- **Fault-tolerant**
  - Data is persisted to disk and can be replicated throughout the cluster
- **High throughput**
  - Each broker can process hundreds of thousands of messages per second*
- **Low latency**
  - Data is delivered in a fraction of a second
- **Flexible**
  - Decouples the production of data from its consumption

*Using modest hardware, with messages of a typical size

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Kafka Use Cases

- **Kafka is used for a variety of use cases, such as**
  - Log aggregation
  - Messaging
  - Web site activity tracking
  - Stream processing
  - Event sourcing

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Message Processing with Apache Kafka

- What Is Apache Kafka?
- **Apache Kafka Overview**
- Scaling Apache Kafka
- Apache Kafka Cluster Architecture
- Apache Kafka Command Line Tools
- Essential Points
- Hands-On Exercise: Producing and Consuming Apache Kafka Messages

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Key Terminology

- **Message**
  - A single data record passed by Kafka
- **Topic**
  - A named log or feed of messages within Kafka
- **Producer**
  - A program that writes messages to Kafka
- **Consumer**
  - A program that reads messages from Kafka

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: High-Level Architecture

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Messages (1)

- **Messages in Kafka are variable-size byte arrays**
  - Represent arbitrary user-defined content
  - Use any format your application requires
  - Common formats include free-form text, JSON, and Avro
- **There is no explicit limit on message size**
  - Optimal performance at a few KB per message
  - Practical limit of 1MB per message

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Messages (2)

- **Kafka retains all messages for a defined time period and/or total size**
  - Administrators can specify retention on global or per-topic basis
  - Kafka will retain messages regardless of whether they were read
  - Kafka discards messages automatically after the retention period or total size is exceeded (whichever limit is reached first)
  - Default retention is one week
  - Retention can reasonably be one year or longer

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Topics

- **There is no explicit limit on the number of topics**
  - However, Kafka works better with a few large topics than many small ones
- **A topic can be created explicitly or simply by publishing to the topic**
  - This behavior is configurable
  - Cloudera recommends that administrators disable auto-creation of topics to avoid accidental creation of large numbers of topics

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Producers

- **Producers publish messages to Kafka topics**
  - They communicate with Kafka, not a consumer
  - Kafka persists messages to disk on receipt

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Consumers

- **A consumer reads messages that were published to Kafka topics**
  - They communicate with Kafka, not any producer
- **Consumer actions do not affect other consumers**
  - For example, having one consumer display the messages in a topic as they are published does not change what is consumed by other consumers
- **They can come and go without impact on the cluster or other consumers**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Producers and Consumers

- **Tools available as part of Kafka**
  - Command-line producer and consumer tools
  - Client (producer and consumer) Java APIs
- **A growing number of other APIs are available from third parties**
  - Client libraries in many languages including Python, PHP, C/C++, Go, .NET, and Ruby
- **Integrations with other tools and projects include**
  - Apache Flume
  - Apache Spark
  - Amazon AWS
  - syslog
- **Kafka also has a large and growing ecosystem**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Message Processing with Apache Kafka

- What Is Apache Kafka?
- Apache Kafka Overview
- **Scaling Apache Kafka**
- Apache Kafka Cluster Architecture
- Apache Kafka Command Line Tools
- Essential Points
- Hands-On Exercise: Producing and Consuming Apache Kafka Messages

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Scaling Kafka

- **Scalability is one of the key benefits of Kafka**
- **Two features let you scale Kafka for performance**
  - Topic partitions
  - Consumer groups

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Topic Partitioning

- **Kafka divides each topic into some number of partitions†**
  - Topic partitioning improves scalability and throughput
- **A topic partition is an ordered and immutable sequence of messages**
  - New messages are appended to the partition as they are received
  - Each message is assigned a unique sequential ID known as an *offset*

†Note that this is unrelated to partitioning in HDFS or Spark

cloudera
Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Consumer Groups

- One or more consumers can form their own consumer group that work together to consume the messages in a topic
- Each partition is consumed by only one member of a consumer group
- Message ordering is preserved per partition, but not across the topic

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Increasing Consumer Throughput

- **Additional consumers can be added to scale consumer group processing**
- **Consumer instances that belong to the same consumer group can be in separate processes or on separate machines**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Multiple Consumer Groups

- **Each message published to a topic is delivered to one consumer instance within each subscribing consumer group**
- **Kafka scales to large numbers of consumer groups and consumers**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Publish and Subscribe to Topic

- **Kafka functions like a traditional queue when all consumer instances belong to the same consumer group**
  - In this case, a given message is received by one consumer
- **Kafka functions like traditional publish-subscribe when each consumer instance belongs to a different consumer group**
  - In this case, all messages are broadcast to all consumer groups

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Message Processing with Apache Kafka

- What Is Apache Kafka?
- Apache Kafka Overview
- Scaling Apache Kafka
- **Apache Kafka Cluster Architecture**
- Apache Kafka Command Line Tools
- Essential Points
- Hands-On Exercise: Producing and Consuming Apache Kafka Messages

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Kafka Clusters

- A Kafka cluster consists of one or more *brokers*—servers running the Kafka broker daemon
- Kafka depends on the Apache ZooKeeper service for coordination

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Apache ZooKeeper

- **Apache ZooKeeper is a coordination service for distributed applications**
- **Kafka depends on the ZooKeeper service for coordination**
  - Typically running three or five ZooKeeper instances
- **Kafka uses ZooKeeper to keep track of brokers running in the cluster**
- **Kafka uses ZooKeeper to detect the addition or removal of consumers**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Kafka Brokers

- **Brokers are the fundamental daemons that make up a Kafka cluster**
- **A broker fully stores a topic partition on disk, with caching in memory**
- **A single broker can reasonably host 1000 topic partitions**
- **One broker is elected controller of the cluster (for assignment of topic partitions to brokers, and so on)**
- **Each broker daemon runs in its own JVM**
  - A single machine can run multiple broker daemons

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Topic Replication

- **At topic creation, a topic can be set with a replication count**
  - Doing so is recommended, as it provides fault tolerance
- **Each broker can act as a leader for some topic partitions and a follower for others**
  - Followers passively replicate the leader
  - If the leader fails, a follower will automatically become the new leader

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Messages Are Replicated

- **Configure the producer with a list of one or more brokers**
  - The producer asks the first available broker for the leader of the desired topic partition
- **The producer then sends the message to the leader**
  - The leader writes the message to its local log
  - Each follower then writes the message to its own log
  - After acknowledgements from followers, the message is committed

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Message Processing with Apache Kafka

- What Is Apache Kafka?
- Apache Kafka Overview
- Scaling Apache Kafka
- Apache Kafka Cluster Architecture
- **Apache Kafka Command Line Tools**
- Essential Points
- Hands-On Exercise: Producing and Consuming Apache Kafka Messages

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Creating Topics from the Command Line

- **Kafka includes a convenient set of command line tools**
  - These are helpful for exploring and experimentation
- **The `kafka-topics` command offers a simple way to create Kafka topics**
  - Provide the topic name of your choice, such as `device_status`
  - You must also specify the ZooKeeper connection string for your cluster

```
$ kafka-topics --create \
  --zookeeper zkhost1:2181,zkhost2:2181,zkhost3:2181 \
  --replication-factor 3 \
  --partitions 5 \
  --topic device_status
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Displaying Topics from the Command Line

- Use the `--list` option to list all topics

```
$ kafka-topics --list \
   --zookeeper zkhost1:2181,zkhost2:2181,zkhost3:2181
```

- Use the `--help` option to list all `kafka-topics` options

```
$ kafka-topics --help
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Running a Producer from the Command Line (1)

- **You can run a producer using the `kafka-console-producer` tool**
- **Specify one or more brokers in the `--broker-list` option**
  - Each broker consists of a hostname, a colon, and a port number
  - If specifying multiple brokers, separate them with commas
- **You must also provide the name of the topic**

```
$ kafka-console-producer \
   --broker-list brokerhost1:9092,brokerhost2:9092  \
   --topic device_status
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Running a Producer from the Command Line (2)

- **You may see a few log messages in the terminal after the producer starts**
- **The producer will then accept input in the terminal window**
  - Each line you type will be a message sent to the topic
- **Until you have configured a consumer for this topic, you will see no other output from Kafka**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Writing File Contents to Topics Using the Command Line

- **Using UNIX pipes or redirection, you can read input from files**
  - The data can then be sent to a topic using the command line producer
- **This example shows how to read input from a file named `alerts.txt`**
  - Each line in this file becomes a separate message in the topic

```
$ cat alerts.txt | kafka-console-producer \
   --broker-list brokerhost1:9092,brokerhost2:9092 \
   --topic device_status
```

- **This technique can be an easy way to integrate with existing programs**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Running a Consumer from the Command Line

- **You can run a consumer with the `kafka-console-consumer` tool**
- **This requires the ZooKeeper connection string for your cluster**
  - Unlike starting a producer, which instead requires a list of brokers
- **The command also requires a topic name**
- **Use `--from-beginning` to read *all* available messages**
  - Otherwise, it reads only *new* messages

```
$ kafka-console-consumer \
   --zookeeper zkhost1:2181,zkhost2:2181,zkhost3:2181 \
   --topic device_status \
   --from-beginning
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Message Processing with Apache Kafka

- What Is Apache Kafka?
- Apache Kafka Overview
- Scaling Apache Kafka
- Apache Kafka Cluster Architecture
- Apache Kafka Command Line Tools
- **Essential Points**
- Hands-On Exercise: Producing and Consuming Apache Kafka Messages

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **Producers publish messages to categories called topics**
- **Messages in a topic are read by consumers**
- **Topics are divided into partitions for performance and scalability**
  - These partitions are replicated for fault tolerance
- **Consumer groups work together to consume the messages in a topic**
- **Nodes running the Kafka service are called brokers**
- **Kafka includes command-line tools for managing topics, and for starting producers and consumers**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Bibliography

The following offer more information on topics discussed in this chapter

- **The Apache Kafka web site**
  - http://kafka.apache.org/
- ***Real-Time Fraud Detection Architecture***
  - http://tiny.cloudera.com/kmc01a
- **Kafka Reference Architecture**
  - http://tiny.cloudera.com/kmc01b
- ***The Log: What Every Software Engineer Should Know…***
  - http://tiny.cloudera.com/kmc01c

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Message Processing with Apache Kafka

- What Is Apache Kafka?
- Apache Kafka Overview
- Scaling Apache Kafka
- Apache Kafka Cluster Architecture
- Apache Kafka Command Line Tools
- Essential Points
- **Hands-On Exercise: Producing and Consuming Apache Kafka Messages**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Producing and Consuming Apache Kafka Messages

- **In this exercise, you will use the Kafka command-line tools to pass messages between a producer and consumer**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Capturing Data with Apache Flume

Appendix B

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- **Appendix: Capturing Data with Apache Flume**
- Appendix: Integrating Apache Flume and Apache Kafka
- Appendix: Importing Relational Data with Apache Sqoop

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Capturing Data with Apache Flume

In this appendix, you will learn

- **What are the main architectural components of Apache Flume**
- **How these components are configured**
- **How to launch a Flume agent**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Capturing Data with Apache Flume

- **What Is Apache Flume?**
- Basic Architecture
- Sources
- Sinks
- Channels
- Configuration
- Essential Points
- Hands-On Exercise: Collecting Web Server Logs with Apache Flume

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

What Is Apache Flume?

- **Apache Flume is a high-performance system for data collection**
  - Name derives from original use case of near-real time log data ingestion
  - Now widely used for collection of any streaming event data
  - Supports aggregating data from many sources into HDFS
- **Originally developed by Cloudera**
  - Donated to Apache Software Foundation in 2011
  - Became a top-level Apache project in 2012
  - Flume OG (Old Generation) gave way to Flume NG (Next Generation)
- **Benefits of Flume**
  - Horizontally-scalable
  - Extensible
  - Reliable

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Flume's Design Goals: Reliability

- **Channels provide Flume's reliability**
- **Examples**
  - Memory channel: Fault intolerant, data will be lost if power is lost
  - Disk-based channel: Fault tolerant
  - Kafka channel: Fault tolerant
- **Data transfer between agents and channels is transactional**
  - A failed data transfer to a downstream agent rolls back and retries
- **You can configure multiple agents with the same task**
  - For example, two agents doing the job of one "collector"—if one agent fails then upstream agents would fail over

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Flume's Design Goals: Scalability

- **Scalability**
  - The ability to increase system performance linearly—or better—by adding more resources to the system
  - Flume scales horizontally
    - As load increases, add more agents to the machine, and add more machines to the system

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Flume's Design Goals: Extensibility

- **Extensibility**
  - The ability to add new functionality to a system
- **Flume can be extended by adding sources and sinks to existing storage layers or data platforms**
  - Flume includes sources that can read data from files, syslog, and standard output from any Linux process
  - Flume includes sinks that can write to files on the local filesystem, HDFS, Kudu, HBase, and so on
  - Developers can write their own sources or sinks

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Common Flume Data Sources

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Large-Scale Deployment Example

- **Flume collects data using configurable *agents***
  - Agents can receive data from many sources, including other agents
  - Large-scale deployments use multiple tiers for scalability and reliability
  - Flume supports inspection and modification of in-flight data

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Capturing Data with Apache Flume

- What Is Apache Flume?
- **Basic Architecture**
- Sources
- Sinks
- Channels
- Configuration
- Essential Points
- Hands-On Exercise: Collecting Web Server Logs with Apache Flume

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Flume Events

- **An *event* is the fundamental unit of data in Flume**
  - Consists of a body (payload) and a collection of headers (metadata)
- **Headers consist of name-value pairs**
  - Headers are mainly used for directing output

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Components in Flume's Architecture

- **Source**
  - Receives events from the external actor that generates them
- **Sink**
  - Sends an event to its destination
- **Channel**
  - Buffers events from the source until they are drained by the sink
- **Agent**
  - Configures and hosts the source, channel, and sink
  - A Java process that runs in a JVM

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Flume Data Flow

- **This diagram illustrates how syslog data might be captured to HDFS**
  1. Server running a syslog daemon logs a message
  2. Flume agent configured with syslog source retrieves event
  3. Source pushes event to the channel, where it is buffered in memory
  4. Sink pulls data from the channel and writes it to HDFS

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Capturing Data with Apache Flume

- What Is Apache Flume?
- Basic Architecture
- **Sources**
- Sinks
- Channels
- Configuration
- Essential Points
- Hands-On Exercise: Collecting Web Server Logs with Apache Flume

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Notable Built-In Flume Sources

- **Syslog**
  - Captures messages from UNIX syslog daemon over the network
- **Netcat**
  - Captures any data written to a socket on an arbitrary TCP port
- **Exec**
  - Executes a UNIX program and reads events from standard output‡
- **Spooldir**
  - Extracts events from files appearing in a specified (local) directory
- **HTTP Source**
  - Retrieves events from HTTP requests
- **Kafka**
  - Retrieves events by consuming messages from a Kafka topic

‡Asynchronous sources do not guarantee that events will be delivered

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Capturing Data with Apache Flume

- What Is Apache Flume?
- Basic Architecture
- Sources
- **Sinks**
- Channels
- Configuration
- Essential Points
- Hands-On Exercise: Collecting Web Server Logs with Apache Flume

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Some Interesting Built-In Flume Sinks

- **Null**
  - Discards all events (Flume equivalent of `/dev/null`)
- **Logger**
  - Logs event to INFO level using SLF4J§
- **IRC**
  - Sends event to a specified Internet Relay Chat channel
- **HDFS**
  - Writes event to a file in the specified directory in HDFS
- **Kafka**
  - Sends event as a message to a Kafka topic
- **HBaseSink**
  - Stores event in HBase

§SLF4J: Simple Logging Façade for Java

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Capturing Data with Apache Flume

- What Is Apache Flume?
- Basic Architecture
- Sources
- Sinks
- **Channels**
- Configuration
- Essential Points
- Hands-On Exercise: Collecting Web Server Logs with Apache Flume

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Built-In Flume Channels

- **Memory**
  - Stores events in the machine's RAM
  - Extremely fast, but not reliable (memory is volatile)
- **File**
  - Stores events on the machine's local disk
  - Slower than RAM, but more reliable (data is written to disk)
- **Kafka**
  - Uses Kafka as a scalable, reliable, and highly available channel between any source and sink type

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Capturing Data with Apache Flume

- What Is Apache Flume?
- Basic Architecture
- Sources
- Sinks
- Channels
- **Configuration**
- Essential Points
- Hands-On Exercise: Collecting Web Server Logs with Apache Flume

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Flume Agent Configuration File

- **Configure Flume agents through a Java properties file**
  - You can configure multiple agents in a single file
- **The configuration file uses hierarchical references**
  - Assign each component a user-defined ID
  - Use that ID in the names of additional properties

```
# Define sources, sinks, and channel for agent named 'agent1'
agent1.sources = mysource
agent1.sinks = mysink
agent1.channels = mychannel

# Sets a property "foo" for the source associated with agent1
agent1.sources.mysource.foo = bar

# Sets a property "baz" for the sink associated with agent1
agent1.sinks.mysink.baz = bat
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Configuring Flume Components (1)

- **Example: Configure a Flume agent to collect data from remote spool directories and save to HDFS**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Configuring Flume Components (2)

```
agent1.sources = src1
agent1.sinks = sink1
agent1.channels = ch1

agent1.channels.ch1.type = memory

agent1.sources.src1.type = spooldir
agent1.sources.src1.spoolDir = /var/flume/incoming
agent1.sources.src1.channels = ch1

agent1.sinks.sink1.type = hdfs
agent1.sinks.sink1.hdfs.path = /loudacre/logdata
agent1.sinks.sink1.channel = ch1
```

- **Properties vary by component type (source, channel, and sink)**
  - Properties also vary by subtype (such as netcat source, syslog source)
  - See the Flume user guide for full details on configuration

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Aside: HDFS Sink Configuration

- **Path may contain patterns based on event headers, such as timestamp**
- **The HDFS sink writes uncompressed SequenceFiles by default**
  - Specifying a codec will enable compression

```
agent1.sinks.sink1.type = hdfs
agent1.sinks.sink1.hdfs.path = /loudacre/logdata/%y-%m-%d
agent1.sinks.sink1.hdfs.codeC = snappy
agent1.sinks.sink1.channel = ch1
```

- **Setting `fileType` parameter to `DataStream` writes *raw* data**
  - Can also specify a file extension, if desired

```
agent1.sinks.sink1.type = hdfs
agent1.sinks.sink1.hdfs.path = /loudacre/logdata/%y-%m-%d
agent1.sinks.sink1.hdfs.fileType = DataStream
agent1.sinks.sink1.hdfs.fileSuffix = .txt
agent1.sinks.sink1.channel = ch1
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Starting a Flume Agent

- **Typical command line invocation**
  - The `--name` argument must match the agent's name in the configuration file
  - Setting root logger as shown will display log messages in the terminal

```
$ flume-ng agent \
  --conf /etc/flume-ng/conf \
  --conf-file /path/to/flume.conf \
  --name agent1 \
  -Dflume.root.logger=INFO,console
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Capturing Data with Apache Flume

- What Is Apache Flume?
- Basic Architecture
- Sources
- Sinks
- Channels
- Configuration
- **Essential Points**
- Hands-On Exercise: Collecting Web Server Logs with Apache Flume

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **Apache Flume is a high-performance system for data collection**
  - Scalable, extensible, and reliable
- **A Flume agent manages the sources, channels, and sinks**
  - Sources retrieve event data from its origin
  - Channels buffer events between the source and sink
  - Sinks send the event to its destination
- **The Flume agent is configured using a properties file**
  - Give each component a user-defined ID
  - Use this ID to define properties of that component

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Bibliography

The following offer more information on topics discussed in this chapter

- **Flume User Guide**
  - http://tiny.cloudera.com/adcc06a

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Capturing Data with Apache Flume

- What Is Apache Flume?
- Basic Architecture
- Sources
- Sinks
- Channels
- Configuration
- Essential Points
- **Hands-On Exercise: Collecting Web Server Logs with Apache Flume**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Collecting Web Server Logs with Apache Flume

- **In this exercise, you will use a Flume agent to ingest log files**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Integrating Apache Flume and Apache Kafka

Appendix C

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- **Appendix: Integrating Apache Flume and Apache Kafka**
- Appendix: Importing Relational Data with Apache Sqoop

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Integrating Apache Flume and Apache Kafka

In this appendix, you will learn

- **What to consider when choosing between Apache Flume and Apache Kafka for a use case**
- **How Flume and Kafka can work together**
- **How to configure a Kafka channel, sink, or source in Flume**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Integrating Apache Flume and Apache Kafka

- **Overview**
- Use Cases
- Configuration
- Essential Points
- Hands-On Exercise: Sending Messages from Flume to Kafka

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Should I Use Kafka or Flume?

- **Both Flume and Kafka are widely used for data ingest**
  - Although these tools differ, their functionality has some overlap
  - Some use cases could be implemented with *either* Flume or Kafka
- **How do you determine which is a better choice for *your* use case?**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Characteristics of Flume

- **Flume is efficient at moving data from a single source into Hadoop**
  - Offers sinks that write to HDFS, an HBase or Kudu table, or a Solr index
  - Easily configured to support common scenarios, without writing code
  - Can also process and transform data during the ingest process

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Characteristics of Kafka

- **Kafka is a publish-subscribe messaging system**
  - Offers more flexibility than Flume for connecting multiple systems
  - Provides better durability and fault tolerance than Flume
  - Often requires writing code for producers and/or consumers
  - Has no direct support for processing messages or loading into Hadoop

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Flafka = Flume + Kafka

- **Both systems have strengths and limitations**
- **You do not necessarily have to choose between them**
  - You can use both when implementing your use case
- **Flafka is the informal name for Flume-Kafka integration**
  - It uses a Flume agent to receive messages from or send messages to Kafka
- **It is implemented as a Kafka source, channel, and sink for Flume**

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Integrating Apache Flume and Apache Kafka

- Overview
- **Use Cases**
- Configuration
- Essential Points
- Hands-On Exercise: Sending Messages from Flume to Kafka

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Using a Flume Kafka Sink as a Producer

- **By using a Kafka sink, Flume can publish messages to a topic**
- **In this example, an application uses Flume to publish application events**
  - The application sends data to the Flume source when events occur
  - The event data is buffered in the channel until it is taken by the sink
  - The Kafka sink publishes messages to a specified topic
  - Any Kafka consumer can then read messages from the topic for application events

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Using a Flume Kafka Source as a Consumer

- **By using a Kafka source, Flume can read messages from a topic**
  - It can then write them to your destination of choice using a Flume sink
- **In this example, the producer sends messages to Kafka**
  - The Flume agent uses a Kafka source, which acts as a consumer
  - The Kafka source reads messages in a specified topic
  - The message data is buffered in the channel until it is taken by the sink
  - The sink then writes the data into HDFS

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Using a Flume Kafka Channel

- **A Kafka channel can be used with any Flume source or sink**
  - Provides a scalable, reliable, high-availability channel
- **In this example, a Kafka channel buffers events**
  - The application sends event data to the Flume source
  - The channel publishes event data as messages on a Kafka topic
  - The sink receives event data and stores it to HDFS

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Using a Kafka Channel as a Consumer (Sourceless Channel)

- **Kafka channels can also be used without a source**
  - It can then write events to your destination of choice using a Flume sink
- **In this example, the Producer sends messages to Kafka brokers**
  - The Flume agent uses a Kafka channel, which acts as a consumer
  - The Kafka channel reads messages in a specified topic
  - Channel passes messages to the sink, which writes the data into HDFS

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Integrating Apache Flume and Apache Kafka

- Overview
- Use Cases
- **Configuration**
- Essential Points
- Hands-On Exercise: Sending Messages from Flume to Kafka

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Configuring Flume with a Kafka Source

- **The table below describes some key properties of the Kafka source**

Name	Description
`type`	`org.apache.flume.source.kafka.KafkaSource`
`zookeeperConnect`	ZooKeeper connection string (example: `zkhost:2181`)
`topic`	Name of Kafka topic from which messages will be read
`groupId`	Unique ID to use for the consumer group (default: `flume`)

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Configuring Flume with a Kafka Source (1)

- **This is the Flume configuration for the example on the previous slide**
  - It defines a source for reading messages from a Kafka topic

```
# Define names for the source, channel, and sink
agent1.sources = source1
agent1.channels = channel1
agent1.sinks = sink1

# Define a Kafka source that reads from the calls_placed topic
# The "type" property line wraps around due to its long value
agent1.sources.source1.type =
org.apache.flume.source.kafka.KafkaSource
agent1.sources.source1.zookeeperConnect = localhost:2181
agent1.sources.source1.topic = calls_placed
agent1.sources.source1.channels = channel1
```

Continued on next slide...

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Configuring Flume with a Kafka Source (2)

- **The remaining portion of the file configures the channel and sink**

```
# Define the properties of our channel
agent1.channels.channel1.type = memory
agent1.channels.channel1.capacity = 10000
agent1.channels.channel1.transactionCapacity = 1000

# Define the sink that writes call data to HDFS
agent1.sinks.sink1.type = hdfs
agent1.sinks.sink1.hdfs.path = /user/training/calls_placed
agent1.sinks.sink1.hdfs.fileType = DataStream
agent1.sinks.sink1.hdfs.fileSuffix = .csv
agent1.sinks.sink1.channel = channel1
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Configuring Flume with a Kafka Sink

- **The table below describes some key properties of the Kafka sink**

Name	Description
`type`	Must be set to `org.apache.flume.sink.kafka.KafkaSink`
`brokerList`	Comma-separated list of brokers (format `host:port`) to contact
`topic`	The topic in Kafka to which the messages will be published
`batchSize`	How many messages to process in one batch

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Configuring Flume with a Kafka Sink (1)

- **This is the Flume configuration for the example on the previous slide**

```
# Define names for the source, channel, and sink
agent1.sources = source1
agent1.channels = channel1
agent1.sinks = sink1

# Define the properties of the source, which receives event
 data
agent1.sources.source1.type = netcat
agent1.sources.source1.bind = localhost
agent1.sources.source1.port = 12345
agent1.sources.source1.channels = channel1

# Define the properties of the channel
agent1.channels.channel1.type = memory
agent1.channels.channel1.capacity = 10000
agent1.channels.channel1.transactionCapacity = 1000
```

Continued on next slide...

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Configuring Flume with a Kafka Sink (2)

- **The remaining portion of the configuration file sets up the Kafka sink**

```
# Define the Kafka sink, which publishes to the app_event topic
agent1.sinks.sink1.type = org.apache.flume.sink.kafka.KafkaSink
agent1.sinks.sink1.topic = app_events
agent1.sinks.sink1.brokerList = localhost:9092
agent1.sinks.sink1.batchSize = 20
agent1.sinks.sink1.channel = channel1
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Configuring Flume with a Kafka Channel

- **The table below describes some key properties of the Kafka channel**

Name	Description
`type`	`org.apache.flume.channel.kafka.KafkaChannel`
`zookeeperConnect`	ZooKeeper connection string (example: `zkhost:2181`)
`brokerList`	Comma-separated list of brokers (format `host:port`) to contact
`topic`	Name of Kafka topic from which messages will be read (optional, default=`flume-channel`)
`parseAsFlumeEvent`	Set to `false` for sourceless configuration (optional, default=`true`)
`readSmallestOffset`	Set to `true` to read from the beginning of the Kafka topic (optional, default=`false`)

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Configuring a Sourceless Kafka Channel (1)

- **This is the Flume configuration for the example shown below**

```
# Define names for the source, channel, and sink
agent1.channels = channel1
agent1.sinks = sink1
```

Continues on next slide...

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Example: Configuring a Sourceless Kafka Channel (2)

- **This is the Flume configuration for the example on the previous slide**

```
# Define the properties of the Kafka channel
# which reads from the calls_placed topic
agent1.channels.channel1.type =
org.apache.flume.channel.kafka.KafkaChannel
agent1.channels.channel1.topic = calls_placed
agent1.channels.channel1.brokerList = localhost:9092
agent1.channels.channel1.zookeeperConnect = localhost:2181
agent1.channels.channel1.parseAsFlumeEvent = false

# Define the sink that writes data to HDFS
agent1.sinks.sink1.type=hdfs
agent1.sinks.sink1.hdfs.path = /user/training/calls_placed
agent1.sinks.sink1.hdfs.fileType = DataStream
agent1.sinks.sink1.hdfs.fileSuffix = .csv
agent1.sinks.sink1.channel = channel1
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Integrating Apache Flume and Apache Kafka

- Overview
- Use Cases
- Configuration
- **Essential Points**
- Hands-On Exercise: Sending Messages from Flume to Kafka

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **Flume and Kafka are distinct systems with different designs**
  - You must weigh the advantages and disadvantages of each when selecting the best tool for your use case
- **Flume and Kafka can be combined**
  - Flafka is the informal name for Flume components integrated with Kafka
  - You can read messages from a topic using a Kafka channel or Kafka source
  - You can publish messages to a topic using a Kafka sink
  - A Kafka channel provides a reliable, high-availability alternative to a memory or file channel

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Bibliography

The following offer more information on topics discussed in this chapter

- **Cloudera documentation on using Flume with Kafka**
  - http://tiny.cloudera.com/flafkadoc
- **Flafka: Apache Flume Meets Apache Kafka for Event Processing**
  - http://tiny.cloudera.com/kmc02a
- **Designing Fraud-Detection Architecture That Works Like Your Brain Does**
  - http://tiny.cloudera.com/kmc02b

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Integrating Apache Flume and Apache Kafka

- Overview
- Use Cases
- Configuration
- Essential Points
- **Hands-On Exercise: Sending Messages from Flume to Kafka**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Sending Messages from Flume to Kafka

- **In this exercise, you will run a Flume agent that sends log data in local files to a Kafka topic**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

cloudera®

Importing Relational Data with Apache Sqoop

Appendix D

cloudera®

Course Chapters

- Introduction
- Introduction to Apache Hadoop and the Hadoop Ecosystem
- Apache Hadoop File Storage
- Distributed Processing on an Apache Hadoop Cluster
- Apache Spark Basics
- Working with DataFrames and Schemas
- Analyzing Data with DataFrame Queries
- RDD Overview
- Transforming Data with RDDs
- Aggregating Data with Pair RDDs
- Querying Tables and Views with Apache Spark SQL
- Working with Datasets in Scala
- Writing, Configuring, and Running Apache Spark Applications
- Distributed Processing
- Distributed Data Persistence
- Common Patterns in Apache Spark Data Processing
- Apache Spark Streaming: Introduction to DStreams
- Apache Spark Streaming: Processing Multiple Batches
- Apache Spark Streaming: Data Sources
- Conclusion
- Appendix: Message Processing with Apache Kafka
- Appendix: Capturing Data with Apache Flume
- Appendix: Integrating Apache Flume and Apache Kafka
- **Appendix: Importing Relational Data with Apache Sqoop**

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Importing Relational Data with Apache Sqoop

In this appendix, you will learn

- **How to import tables from an RDBMS into your Hadoop cluster**
- **How to change the delimiter and file format of imported tables**
- **How to control which tables, columns, and rows are imported**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Importing Relational Data with Apache Sqoop

- **Apache Sqoop Overview**
- Importing Data
- Import File Options
- Exporting Data
- Essential Points
- Hands-On Exercise: Import Data from MySQL Using Apache Sqoop

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

What Is Apache Sqoop?

- **Open source Apache project originally developed by Cloudera**
  - The name is a contraction of “SQL-to-Hadoop”
- **Sqoop exchanges data between a database and HDFS**
  - Can import all tables, a single table, or a partial table into HDFS
  - Data can be imported in a variety of formats
  - Sqoop can also export data from HDFS to a database

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

How Does Sqoop Work?

- **Sqoop is a client-side application that imports data using Hadoop MapReduce**
- **A basic import involves three steps orchestrated by Sqoop**
  - Examine table details
  - Create and submit job to cluster
  - Fetch records from table and write this data to HDFS

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Basic Syntax

- **Sqoop is a command-line utility with several subcommands, called *tools***
  - There are tools for import, export, listing database contents, and more
  - Run `sqoop help` to see a list of all tools
  - Run `sqoop help tool-name` for help on using a specific tool
- **Basic syntax of a Sqoop invocation**

```
$ sqoop tool-name [tool-options]
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Exploring a Database with Sqoop

- This command will list all tables in the `loudacre` database in MySQL

```
$ sqoop list-tables \
  --connect jdbc:mysql://dbhost/loudacre \
  --username dbuser \
  --password pw
```

- You can perform database queries using the `eval` tool

```
$ sqoop eval \
  --query "SELECT * FROM my_table LIMIT 5" \
  --connect jdbc:mysql://dbhost/loudacre \
  --username dbuser \
  --password pw
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Importing Relational Data with Apache Sqoop

- Apache Sqoop Overview
- **Importing Data**
- Import File Options
- Exporting Data
- Essential Points
- Hands-On Exercise: Import Data from MySQL Using Apache Sqoop

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Overview of the Import Process

- **Imports are performed using Hadoop MapReduce jobs**
- **Sqoop begins by examining the table to be imported**
  - Determines the primary key, if possible
  - Runs a *boundary query* to see how many records will be imported
  - Divides result of boundary query by the number of tasks (mappers)
    - Uses this to configure tasks so that they will have equal loads
- **Sqoop also generates a Java source file for each table being imported**
  - It compiles and uses this during the import process
  - The file remains after import, but can be safely deleted

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Importing an Entire Database with Sqoop

- **The `import-all-tables` tool imports an entire database**
  - Stored as comma-delimited files
  - Default base location is your HDFS home directory
  - Data will be in subdirectories corresponding to name of each table

```
$ sqoop import-all-tables \
  --connect jdbc:mysql://dbhost/loudacre \
  --username dbuser --password pw
```

- **Use the `--warehouse-dir` option to specify a different base directory**

```
$ sqoop import-all-tables \
  --connect jdbc:mysql://dbhost/loudacre \
  --username dbuser --password pw \
  --warehouse-dir /loudacre
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Importing a Single Table with Sqoop

- **The `import` tool imports a single table**
- **This example imports the `accounts` table**
  - It stores the data in HDFS as comma-delimited fields

```
$ sqoop import --table accounts \
  --connect jdbc:mysql://dbhost/loudacre \
  --username dbuser --password pw
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Importing Partial Tables with Sqoop

- **Import only specified columns from `accounts` table**

```
$ sqoop import --table accounts \
  --connect jdbc:mysql://dbhost/loudacre \
  --username dbuser --password pw \
  --columns "id,first_name,last_name,state"
```

- **Import only matching rows from `accounts` table**

```
$ sqoop import --table accounts \
  --connect jdbc:mysql://dbhost/loudacre \
  --username dbuser --password pw \
  --where "state='CA'"
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Importing Relational Data with Apache Sqoop

- Apache Sqoop Overview
- Importing Data
- **Import File Options**
- Exporting Data
- Essential Points
- Hands-On Exercise: Import Data from MySQL Using Apache Sqoop

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Specifying a File Location

- **By default, Sqoop stores the data in the user's HDFS home directory**
  - In a subdirectory corresponding to the table name
  - For example `/user/training/accounts`
- **This example specifies an alternate location**

```
$ sqoop import --table accounts \
  --connect jdbc:mysql://dbhost/loudacre \
  --username dbuser --password pw \
  --target-dir /loudacre/customer_accounts
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Specifying an Alternate Delimiter

- **By default, Sqoop generates text files with comma-delimited fields**
- **This example writes tab-delimited fields instead**

```
$ sqoop import --table accounts \
  --connect jdbc:mysql://dbhost/loudacre \
  --username dbuser --password pw \
  --fields-terminated-by "\t"
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Using Compression with Sqoop

- **Sqoop supports storing data in a compressed file**
  - Use the `--compression-codec` flag

```
$ sqoop import --table accounts \
  --connect jdbc:mysql://dbhost/loudacre \
  --username dbuser --password pw \
  --compression-codec \
   org.apache.hadoop.io.compress.SnappyCodec
```

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Storing Data in Other Data Formats

- **By default, Sqoop stores data in text format files**
- **Sqoop supports importing data as Parquet or Avro files**

```
$ sqoop import --table accounts \
  --connect jdbc:mysql://dbhost/loudacre \
  --username dbuser --password pw \
  --as-parquetfile
```

```
$ sqoop import --table accounts \
  --connect jdbc:mysql://dbhost/loudacre \
  --username dbuser --password pw \
  --as-avrodatafile
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Importing Relational Data with Apache Sqoop

- Apache Sqoop Overview
- Importing Data
- Import File Options
- **Exporting Data**
- Essential Points
- Hands-On Exercise: Import Data from MySQL Using Apache Sqoop

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Exporting Data from Hadoop to RDBMS with Sqoop

- **Sqoop's `import` tool pulls records from an RDBMS into HDFS**
- **It is sometimes necessary to *push* data in HDFS back to an RDBMS**
  - Good solution when you must do batch processing on large data sets
  - Export results to a relational database for access by other systems
- **Sqoop supports this via the `export` tool**
  - The RDBMS table must already exist prior to export

```
$ sqoop export \
  --connect jdbc:mysql://dbhost/loudacre \
  --username dbuser --password pw \
  --export-dir /loudacre/recommender_output \
  --update-mode allowinsert \
  --table product_recommendations
```

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Importing Relational Data with Apache Sqoop

- Apache Sqoop Overview
- Importing Data
- Import File Options
- Exporting Data
- **Essential Points**
- Hands-On Exercise: Import Data from MySQL Using Apache Sqoop

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Essential Points

- **Sqoop exchanges data between a database and a Hadoop cluster**
  - Provides subcommands (*tools*) for importing, exporting, and more
- **Tables are imported using MapReduce jobs**
  - These are written as comma-delimited text by default
  - You can specify alternate delimiters or file formats
- **Sqoop provides many options to control imports**
  - You can select only certain columns or limit rows

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Bibliography

The following offer more information on topics discussed in this chapter

- **Sqoop User Guide**
  - http://tiny.cloudera.com/sqoopuser
- ***Apache Sqoop Cookbook*** **(published by O'Reilly)**
  - http://tiny.cloudera.com/sqoopcookbook

cloudera

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Chapter Topics

Importing Relational Data with Apache Sqoop

- Apache Sqoop Overview
- Importing Data
- Import File Options
- Exporting Data
- Essential Points
- **Hands-On Exercise: Import Data from MySQL Using Apache Sqoop**

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.

Hands-On Exercise: Importing Data from MySQL Using Apache Sqoop

- **In this exercise, you will use Sqoop to import data from an RDBMS to HDFS**
  - Please refer to the Hands-On Exercise Manual for instructions

cloudera®

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.