

ADDISON
WESLEY
DATA &
ANALYTICS
SERIES



EXPERT HADOOP[®] ADMINISTRATION

Managing, Tuning,
and Securing **Spark**,
YARN, and **HDFS**

SAM R. ALAPATI

FREE SAMPLE CHAPTER
SHARE WITH OTHERS



Expert Hadoop[®] Administration

Expert Hadoop[®] Administration

Managing, Tuning, and Securing
Spark, YARN, and HDFS

Sam R. Alapati

◆ Addison-Wesley

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2016954056

Copyright © 2017 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearsoned.com/permissions/.

ISBN-13: 978-0-13-459719-5

ISBN-10: 0-13-459719-2

*To my cousin, Alapati Srinath, whom I consider my
own brother. Thank you, Srinath, for your kindness, affection, and above all,
graciousness, all of which have meant a lot to me over the years.*

This page intentionally left blank

Contents

Foreword	xxvii
Preface	xxix
Acknowledgments	xxxv
About the Author	xxxvii

I Introduction to Hadoop—Architecture and Hadoop Clusters 1

1 Introduction to Hadoop and Its Environment	3
Hadoop—An Introduction	4
Unique Features of Hadoop	5
Big Data and Hadoop	5
A Typical Scenario for Using Hadoop	7
Traditional Database Systems	7
Data Lake	9
Big Data, Data Science and Hadoop	11
Cluster Computing and Hadoop Clusters	12
Cluster Computing	12
Hadoop Clusters	13
Hadoop Components and the Hadoop Ecosystem	15
What Do Hadoop Administrators Do?	18
Hadoop Administration—A New Paradigm	18
What You Need to Know to Administer Hadoop	20
The Hadoop Administrator’s Toolset	21
Key Differences between Hadoop 1 and Hadoop 2	21
Architectural Differences	22
High-Availability Features	22
Multiple Processing Engines	23
Separation of Processing and Scheduling	23
Resource Allocation in Hadoop 1 and Hadoop 2	24
Distributed Data Processing: MapReduce and Spark, Hive and Pig	24
MapReduce	24
Apache Spark	25
Apache Hive	26
Apache Pig	26

Data Integration: Apache Sqoop, Apache Flume and Apache Kafka **27**

Key Areas of Hadoop Administration **28**

Managing the Cluster Storage **28**

Allocating the Cluster Resources **28**

Scheduling Jobs **29**

Securing Hadoop Data **30**

Summary **31**

2 An Introduction to the Architecture of Hadoop 33

Distributed Computing and Hadoop **33**

Hadoop Architecture **34**

A Hadoop Cluster **35**

Master and Worker Nodes **36**

Hadoop Services **36**

Data Storage—The Hadoop Distributed File System **37**

HDFS Unique Features **37**

HDFS Architecture **38**

The HDFS File System **40**

NameNode Operations **43**

Data Processing with YARN, the Hadoop Operating System **48**

Architecture of YARN **49**

How the ApplicationMaster Works with the ResourceManager to Allocate Resources **53**

Summary **57**

3 Creating and Configuring a Simple Hadoop Cluster 59

Hadoop Distributions and Installation Types **60**

Hadoop Distributions **60**

Hadoop Installation Types **61**

Setting Up a Pseudo-Distributed Hadoop Cluster **62**

Meeting the Operating System Requirements **63**

Modifying Kernel Parameters **64**

Setting Up SSH **68**

Java Requirements **69**

Installing the Hadoop Software **70**

Creating the Necessary Hadoop Users	70
Creating the Necessary Directories	71
Performing the Initial Hadoop Configuration	71
Environment Configuration Files	73
Read-Only Default Configuration Files	74
Site-Specific Configuration Files	74
Other Hadoop-Related Configuration Files	74
Precedence among the Configuration Files	76
Variable Expansion and Configuration Parameters	78
Configuring the Hadoop Daemons Environment	79
Configuring Core Hadoop Properties (with the core-site.xml File)	81
Configuring MapReduce (with the mapred-site.xml File)	82
Configuring YARN (with the yarn-site.xml File)	83
Operating the New Hadoop Cluster	86
Formatting the Distributed File System	86
Setting the Environment Variables	87
Starting the HDFS and YARN Services	87
Verifying the Service Startup	89
Shutting Down the Services	90
Summary	90
4 Planning for and Creating a Fully Distributed Cluster	91
Planning Your Hadoop Cluster	92
General Cluster Planning Considerations	92
Server Form Factors	94
Criteria for Choosing the Nodes	94
Going from a Single Rack to Multiple Racks	95
Sizing a Hadoop Cluster	96
General Principles Governing the Choice of CPU, Memory and Storage	96
Special Treatment for the Master Nodes	99
Recommendations for Sizing the Servers	100
Growing a Cluster	101
Guidelines for Large Clusters	101

Creating a Multinode Cluster	102
How the Test Cluster Is Set Up	102
Modifying the Hadoop Configuration	106
Changing the HDFS Configuration (hdfs-site.xml file)	106
Changing the YARN Configuration	109
Changing the MapReduce Configuration	113
Starting Up the Cluster	114
Starting Up and Shutting Down the Cluster with Scripts	116
Performing a Quick Check of the New Cluster's File System	118
Configuring Hadoop Services, Web Interfaces and Ports	119
Service Configuration and Web Interfaces	119
Setting Port Numbers for Hadoop Services	122
Hadoop Clients	124
Summary	126

II Hadoop Application Frameworks **127**

5 Running Applications in a Cluster—The MapReduce Framework (and Hive and Pig) **129**

The MapReduce Framework	129
The MapReduce Model	130
How MapReduce Works	131
MapReduce Job Processing	133
A Simple MapReduce Program	135
Understanding Hadoop's Job Processing—Running a WordCount Program	136
MapReduce Input and Output Directories	137
How Hadoop Shows You the Job Details	137
Hadoop Streaming	139
Apache Hive	141
Hive Data Organization	142
Working with Hive Tables	142
Loading Data into Hive	142
Querying with Hive	143

Apache Pig	144
Pig Execution Modes	144
A Simple Pig Example	145
Summary	145
6 Running Applications in a Cluster—The Spark Framework	147
What Is Spark?	148
Why Spark?	149
Speed	149
Ease of Use and Accessibility	151
General-Purpose Framework	152
Spark and Hadoop	153
The Spark Stack	153
Installing Spark	155
Spark Examples	157
Key Spark Files and Directories	157
Compiling the Spark Binaries	157
Reducing Spark’s Verbosity	158
Spark Run Modes	158
Local Mode	158
Cluster Mode	158
Understanding the Cluster Managers	159
The Standalone Cluster Manager	159
Spark on Apache Mesos	161
Spark on YARN	162
How YARN and Spark Work Together	163
Setting Up Spark on a Hadoop Cluster	163
Spark and Data Access	164
Loading Data from the Linux File System	164
Loading Data from HDFS	164
Loading Data from a Relational Database	166
Summary	167
7 Running Spark Applications	169
The Spark Programming Model	169
Spark Programming and RDDs	169
Programming Spark	172

Spark Applications	173
Basics of RDDs	174
Creating an RDD	174
RDD Operations	176
RDD Persistence	179
Architecture of a Spark Application	179
Spark Terminology	180
Components of a Spark Application	180
Running Spark Applications Interactively	181
Spark Shell and Spark Applications	181
A Bit about the Spark Shell	182
Using the Spark Shell	182
Overview of Spark Cluster Execution	185
Creating and Submitting Spark Applications	185
Building the Spark Application	186
Running an Application in the Standalone Spark Cluster	186
Using <code>spark-submit</code> to Execute Applications	187
Running Spark Applications on Mesos	189
Running Spark Applications in a YARN-Managed Hadoop Cluster	189
Using the JDBC/ODBC Server	191
Configuring Spark Applications	192
Spark Configuration Properties	192
Specifying Configuration when Running <code>spark-submit</code>	193
Monitoring Spark Applications	194
Handling Streaming Data with Spark Streaming	194
How Spark Streaming Works	195
A Spark Streaming Example—WordCount Again!	197
Using Spark SQL for Handling Structured Data	198
DataFrames	198
HiveContext and SQLContext	198
Working with Spark SQL	199
Creating DataFrames	200
Summary	201

III Managing and Protecting Hadoop Data and High Availability 203

8 The Role of the NameNode and How HDFS

Works 205

HDFS—The Interaction between the NameNode and the DataNodes **205**

Interaction between the Clients and HDFS **206**

NameNode and DataNode Communications **207**

Rack Awareness and Topology **209**

How to Configure Rack Awareness in Your Cluster **210**

Finding Your Cluster's Rack Information **210**

HDFS Data Replication **212**

HDFS Data Organization and Data Blocks **213**

Data Replication **213**

Block and Replica States **216**

How Clients Read and Write HDFS Data **218**

How Clients Read HDFS Data **219**

How Clients Write Data to HDFS **220**

Understanding HDFS Recovery Processes **224**

Generation Stamp **224**

Lease Recovery **224**

Block Recovery **226**

Pipeline Recovery **226**

Centralized Cache Management in HDFS **227**

Hadoop and OS Page Caching **228**

The Key Principles Behind Centralized Cache Management **228**

How Centralized Cache Management Works **229**

Configuring Caching **229**

Cache Directives **230**

Cache Pools **230**

Using the Cache **231**

Hadoop Archival Storage, SSD and Memory (Heterogeneous Storage) **232**

Performance Characteristics of Storage Types **233**

The Need for Heterogeneous HDFS Storage **233**

- Changes in the Storage Architecture **234**
- Storage Preferences for Files **235**
- Setting Up Archival Storage **235**
- Managing Storage Policies **239**
- Moving Data Around **239**
- Implementing Archival Storage **240**
- Summary **241**

9 HDFS Commands, HDFS Permissions and HDFS Storage 243

- Managing HDFS through the HDFS Shell Commands **243**
 - Using the `hdfs dfs` Utility to Manage HDFS **245**
 - Listing HDFS Files and Directories **247**
 - Creating an HDFS Directory **249**
 - Removing HDFS Files and Directories **249**
 - Changing File and Directory Ownership and Groups **250**
- Using the `dfsadmin` Utility to Perform HDFS Operations **251**
 - The `dfsadmin -report` Command **252**
- Managing HDFS Permissions and Users **255**
 - HDFS File Permissions **255**
 - HDFS Users and Super Users **257**
- Managing HDFS Storage **260**
 - Checking HDFS Disk Usage **260**
 - Allocating HDFS Space Quotas **263**
- Rebalancing HDFS Data **267**
 - Reasons for HDFS Data Imbalance **268**
 - Running the Balancer Tool to Balance HDFS Data **268**
 - Using `hdfs dfsadmin` to Make Things Easier **271**
 - When to Run the Balancer **273**
- Reclaiming HDFS Space **274**
 - Removing Files and Directories **274**
 - Decreasing the Replication Factor **274**
- Summary **276**

10	Data Protection, File Formats and Accessing HDFS	277
	Safeguarding Data	278
	Using HDFS Trash to Prevent Accidental Data Deletion	278
	Using HDFS Snapshots to Protect Important Data	280
	Ensuring Data Integrity with File System Checks	284
	Data Compression	289
	Common Compression Formats	290
	Evaluating the Various Compression Schemes	291
	Compression at Various Stages for MapReduce	291
	Compression for Spark	295
	Data Serialization	295
	Hadoop File Formats	295
	Criteria for Determining the Right File Format	296
	File Formats Supported by Hadoop	298
	The Ideal File Format	302
	The Hadoop Small Files Problem and Merging Files	303
	Using a Federated NameNode to Overcome the Small Files Problem	304
	Using Hadoop Archives to Manage Many Small Files	304
	Handling the Performance Impact of Small Files	307
	Using Hadoop WebHDFS and HttpFS	308
	WebHDFS—The Hadoop REST API	308
	Using the WebHDFS API	309
	Understanding the WebHDFS Commands	310
	Using HttpFS Gateway to Access HDFS from Behind a Firewall	313
	Summary	315
11	NameNode Operations, High Availability and Federation	317
	Understanding NameNode Operations	318
	HDFS Metadata	319

- The NameNode Startup Process **321**
 - How the NameNode and the DataNodes Work Together **322**
- The Checkpointing Process **323**
 - Secondary, Checkpoint, Backup and Standby Nodes **324**
 - Configuring the Checkpointing Frequency **325**
 - Managing Checkpoint Performance **327**
 - The Mechanics of Checkpointing **327**
- NameNode Safe Mode Operations **329**
 - Automatic Safe Mode Operations **329**
 - Placing the NameNode in Safe Mode **330**
 - How the NameNode Transitions Through Safe Mode **331**
 - Backing Up and Recovering the NameNode Metadata **332**
- Configuring HDFS High Availability **334**
 - NameNode HA Architecture (QJM) **335**
 - Setting Up an HDFS HA Quorum Cluster **337**
 - Deploying the High-Availability NameNodes **342**
 - Managing an HA NameNode Setup **345**
 - HA Manual and Automatic Failover **346**
- HDFS Federation **349**
 - Architecture of a Federated NameNode **350**
- Summary **351**

- IV Moving Data, Allocating Resources, Scheduling Jobs and Security 353**
 - 12 Moving Data Into and Out of Hadoop 355**
 - Introduction to Hadoop Data Transfer Tools **355**
 - Loading Data into HDFS from the Command Line **356**
 - Using the `-cat` Command to Dump a File's Contents **356**
 - Testing HDFS Files **357**
 - Copying and Moving Files from and to HDFS **358**
 - Using the `-get` Command to Move Files **359**

Moving Files from and to HDFS	360
Using the <code>-tail</code> and <code>head</code> Commands	360
Copying HDFS Data between Clusters with DistCp	361
How to Use the DistCp Command to Move Data	361
DistCp Options	363
Ingesting Data from Relational Databases with Sqoop	365
Sqoop Architecture	366
Deploying Sqoop	367
Using Sqoop to Move Data	368
Importing Data with Sqoop	368
Importing Data into Hive	379
Exporting Data with Sqoop	381
Ingesting Data from External Sources with Flume	388
Flume Architecture in a Nutshell	389
Configuring the Flume Agent	391
A Simple Flume Example	392
Using Flume to Move Data to HDFS	394
A More Complex Flume Example	395
Ingesting Data with Kafka	398
Benefits Offered by Kafka	398
How Kafka Works	399
Setting Up an Apache Kafka Cluster	401
Integrating Kafka with Hadoop and Storm	404
Summary	406

13 Resource Allocation in

a Hadoop Cluster	407
Resource Allocation in Hadoop	407
Managing Cluster Workloads	408
Hadoop's Resource Schedulers	409
The FIFO Scheduler	410
The Capacity Scheduler	411
Queues and Subqueues	412
How the Cluster Allocates Resources	418
Preempting Applications	421

Enabling the Capacity Scheduler	422
A Typical Capacity Scheduler	422
The Fair Scheduler	426
Queues	427
Configuring the Fair Scheduler	428
How Jobs Are Placed into Queues	430
Application Preemption in the Fair Scheduler	431
Security and Resource Pools	432
A Sample fair-scheduler.xml File	432
Submitting Jobs to the Scheduler	434
Moving Applications between Queues	434
Monitoring the Fair Scheduler	434
Comparing the Capacity Scheduler and the Fair Scheduler	435
Similarities between the Two Schedulers	435
Differences between the Two Schedulers	435
Summary	436
14 Working with Oozie to Manage Job Workflows	437
Using Apache Oozie to Schedule Jobs	437
Oozie Architecture	439
The Oozie Server	439
The Oozie Client	440
The Oozie Database	440
Deploying Oozie in Your Cluster	441
Installing and Configuring Oozie	442
Configuring Hadoop for Oozie	444
Understanding Oozie Workflows	446
Workflows, Control Flow, and Nodes	446
Defining the Workflows with the workflow.xml File	447
How Oozie Runs an Action	449
Configuring the Action Nodes	449
Creating an Oozie Workflow	454
Configuring the Control Nodes	456
Configuring the Job	460
Running an Oozie Workflow Job	461
Specifying the Job Properties	461

Deploying Oozie Jobs	463
Creating Dynamic Workflows	463
Oozie Coordinators	464
Time-Based Coordinators	465
Data-Based Coordinators	467
Time-and-Data-Based Coordinators	469
Submitting the Oozie Coordinator from the Command Line	469
Managing and Administering Oozie	470
Common Oozie Commands and How to Run Them	471
Troubleshooting Oozie	473
Oozie <code>cron</code> Scheduling and Oozie Service Level Agreements	474
Summary	475
15 Securing Hadoop	477
Hadoop Security—An Overview	478
Authentication, Authorization and Accounting	480
Hadoop Authentication with Kerberos	481
Kerberos and How It Works	482
The Kerberos Authentication Process	483
Kerberos Trusts	484
A Special Principal	485
Adding Kerberos Authorization to your Cluster	486
Setting Up Kerberos for Hadoop	490
Securing a Hadoop Cluster with Kerberos	495
How Kerberos Authenticates Users and Services	501
Managing a Kerberized Hadoop Cluster	501
Hadoop Authorization	505
HDFS Permissions	505
Service Level Authorization	510
Role-Based Authorization with Apache Sentry	512
Auditing Hadoop	518
Auditing HDFS Operations	519
Auditing YARN Operations	519

- Securing Hadoop Data **520**
 - HDFS Transparent Encryption **520**
 - Encrypting Data in Transition **523**
- Other Hadoop-Related Security Initiatives **524**
 - Securing a Hadoop Infrastructure with Apache Knox Gateway **524**
 - Apache Ranger for Security Administration **525**
- Summary **525**

V Monitoring, Optimization and Troubleshooting 527

16 Managing Jobs, Using Hue and Performing Routine Tasks 529

- Using the YARN Commands to Manage Hadoop Jobs **530**
 - Viewing YARN Applications **531**
 - Checking the Status of an Application **532**
 - Killing a Running Application **532**
 - Checking the Status of the Nodes **533**
 - Checking YARN Queues **533**
 - Getting the Application Logs **533**
 - Yarn Administrative Commands **534**
- Decommissioning and Recommissioning Nodes **535**
 - Including and Excluding Hosts **536**
 - Decommissioning DataNodes and NodeManagers **537**
 - Recommissioning Nodes **539**
 - Things to Remember about Decommissioning and Recommissioning **539**
 - Adding a New DataNode and/or a NodeManager **540**
- ResourceManager High Availability **541**
 - ResourceManager High-Availability Architecture **541**
 - Setting Up ResourceManager High Availability **542**
 - ResourceManager Failover **543**
 - Using the ResourceManager High-Availability Commands **545**

Performing Common Management Tasks	545
Moving the NameNode to a Different Host	545
Managing High-Availability NameNodes	546
Using a Shutdown/Startup Script to Manage your Cluster	546
Balancing HDFS	546
Balancing the Storage on the DataNodes	547
Managing the MySQL Database	548
Configuring a MySQL Database	548
Configuring MySQL High Availability	549
Backing Up Important Cluster Data	551
Backing Up HDFS Metadata	552
Backing Up the Metastore Databases	553
Using Hue to Administer Your Cluster	553
Allowing Your Users to Use Hue	554
Installing Hue	556
Configuring Your Cluster to Work with Hue	557
Managing Hue	561
Working with Hue	561
Implementing Specialized HDFS Features	562
Deploying HDFS and YARN in a Multihomed Network	562
Short-Circuit Local Reads	563
Mountable HDFS	564
Using an NFS Gateway for Mounting HDFS to a Local File System	566
Summary	567
17 Monitoring, Metrics and Hadoop Logging	569
Monitoring Linux Servers	570
Basics of Linux System Monitoring	570
Monitoring Tools for Linux Systems	572
Hadoop Metrics	576
Hadoop Metric Types	577
Using the Hadoop Metrics	578
Capturing Metrics to a File System	578
Using Ganglia for Monitoring	579
Ganglia Architecture	580

Setting Up the Ganglia and Hadoop Integration	580
Setting Up the Hadoop Metrics	582
Understanding Hadoop Logging	582
Hadoop Log Messages	583
Daemon and Application Logs and How to View Them	584
How Application Logging Works	585
How Hadoop Uses HDFS Staging Directories and Local Directories During a Job Run	587
How the NodeManager Uses the Local Directories	588
Storing Job Logs in HDFS through Log Aggregation	592
Working with the Hadoop Daemon Logs	597
Using Hadoop's Web UIs for Monitoring	599
Monitoring Jobs with the ResourceManager Web UI	599
The JobHistoryServer Web UI	606
Monitoring with the NameNode Web UI	608
Monitoring Other Hadoop Components	609
Monitoring Hive	609
Monitoring Spark	610
Summary	610
18 Tuning the Cluster Resources, Optimizing MapReduce Jobs and Benchmarking	611
How to Allocate YARN Memory and CPU	612
Allocating Memory	612
Configuring the Number of CPU Cores	620
Relationship between Memory and CPU Vcores	621
Configuring Efficient Performance	621
Speculative Execution	621
Reducing the I/O Load on the System	624
Tuning Map and Reduce Tasks—What the Administrator Can Do	625
Tuning the Map Tasks	626
Input and Output	627
Tuning the Reduce Tasks	630
Tuning the MapReduce Shuffle Process	632

Optimizing Pig and Hive Jobs	635
Optimizing Hive Jobs	635
Optimizing Pig Jobs	637
Benchmarking Your Cluster	638
Using TestDFSIO for Testing I/O Performance	638
Benchmarking with TeraSort	640
Using Hadoop's Rumen and GridMix for Benchmarking	643
Hadoop Counters	647
File System Counters	649
Job Counters	649
MapReduce Framework Counters	650
Custom Java Counters	651
Limiting the Number of Counters	651
Optimizing MapReduce	652
Map-Only versus Map and Reduce Jobs	652
How Combiners Improve MapReduce Performance	652
Using a Partitioner to Improve Performance	654
Compressing Data During the MapReduce Process	654
Too Many Mappers or Reducers?	655
Summary	658
19 Configuring and Tuning Apache Spark on YARN	659
Configuring Resource Allocation for Spark on YARN	659
Allocating CPU	660
Allocating Memory	660
How Resources are Allocated to Spark	660
Limits on the Resource Allocation to Spark Applications	661
Allocating Resources to the Driver	663
Configuring Resources for the Executors	666
How Spark Uses its Memory	670
Things to Remember	672
Cluster or Client Mode?	674
Configuring Spark-Related Network Parameters	676

- Dynamic Resource Allocation when Running Spark on YARN **676**
 - Dynamic and Static Resource Allocation **676**
 - How Spark Manages Dynamic Resource Allocation **677**
 - Enabling Dynamic Resource Allocation **677**
- Storage Formats and Compressing Data **678**
 - Storage Formats **679**
 - File Sizes **680**
 - Compression **680**
- Monitoring Spark Applications **681**
 - Using the Spark Web UI to Understand Performance **682**
 - Spark System and the Metrics REST API **684**
 - The Spark History Server on YARN **684**
 - Tracking Jobs from the Command Line **686**
- Tuning Garbage Collection **686**
 - The Mechanics of Garbage Collection **687**
 - How to Collect GC Statistics **687**
- Tuning Spark Streaming Applications **688**
 - Reducing Batch Processing Time **688**
 - Setting the Right Batch Interval **689**
 - Tuning Memory and Garbage Collection **689**
- Summary **689**
- 20 Optimizing Spark Applications 691**
 - Revisiting the Spark Execution Model **692**
 - The Spark Execution Model **692**
 - Shuffle Operations and How to Minimize Them **694**
 - A WordCount Example to Our Rescue Again **695**
 - Impact of a Shuffle Operation **696**
 - Configuring the Shuffle Parameters **697**
 - Partitioning and Parallelism (Number of Tasks) **703**
 - Level of Parallelism **704**
 - Problems with Too Few Tasks **706**
 - Setting the Default Number of Partitions **706**
 - How to Increase the Number of Partitions **707**
 - Using the Repartition and Coalesce Operators to Change the Number of Partitions in an RDD **708**

Two Types of Partitioners	709
Data Partitioning and How It Can Avoid a Shuffle	709
Optimizing Data Serialization and Compression	710
Data Serialization	710
Configuring Compression	711
Understanding Spark's SQL Query Optimizer	712
Understanding the Optimizer Steps	712
Spark's Speculative Execution Feature	714
The Importance of Data Locality	715
Caching Data	717
Fault-Tolerance Due to Caching	718
How to Specify Caching	718
Summary	723
21 Troubleshooting Hadoop—A Sampler	725
Space-Related Issues	725
Dealing with a 100 Percent Full Linux File System	726
HDFS Space Issues	727
Local and Log Directories Out of Free Space	727
Disk Volume Failure Toleration	729
Handling YARN Jobs That Are Stuck	731
JVM Memory-Allocation and Garbage-Collection Strategies	732
Understanding JVM Garbage Collection	732
Optimizing Garbage Collection	733
Analyzing Memory Usage	734
Out of Memory Errors	734
ApplicationMaster Memory Issues	735
Handling Different Types of Failures	737
Handling Daemon Failures	737
Starting Failures for Hadoop Daemons	737
Task and Job Failures	738
Troubleshooting Spark Jobs	739
Spark's Fault Tolerance Mechanism	740
Killing Spark Jobs	740
Maximum Attempts for a Job	740
Maximum Failures per Job	740

Debugging Spark Applications	740
Viewing Logs with Log Aggregation	740
Viewing Logs When Log Aggregation Is Not Enabled	741
Reviewing the Launch Environment	741
Summary	742

22 Installing VirtualBox and Linux and Cloning the Virtual Machines 743

Installing Oracle VirtualBox	744
Installing Oracle Enterprise Linux	745
Cloning the Linux Server	745

Index	747
--------------	------------

Foreword

Apache Hadoop 2 and the upcoming 3 were a major step forward in moving beyond the paradigm of MapReduce. At the core of this is the new YARN (Yet Another Resource Negotiator) processing framework for creating APIs and processing engines on top of Hadoop and HDFS, including the original MapReduce paradigm. Hadoop 2 is a significant upgrade to Hadoop 1, requiring updates to how a cluster is set up, managed and administered. This book provides everything a developer, operator or administrator would need to manage a production Hadoop 2 cluster of any size.

While Hadoop 2 and 3 at the core are HDFS and YARN, there are many other projects that are included in a typical production Hadoop cluster. For example, Hive, Pig, Spark, Flume and Kafka are often paired with the core Hadoop infrastructure to provide additional functionality and features. This book includes coverage of many of these complementary projects with introductory materials good for developers and administrators alike.

Sam Alapati is the principal Hadoop administrator at Sabre Holdings and has been working with production Hadoop clusters for the last six years. He's uniquely qualified to cover the administration of production clusters and has pulled everything together in this single resource. The depth of experience that Sam brings to this book has enabled him to write much more than a simple introduction to Hadoop and Spark. While it does provide that introductory material, it will be the go-to resource for administrators looking to spec, size, expand and secure their production Hadoop clusters.

—Paul Dix, Series Editor

This page intentionally left blank

Preface

Apache Hadoop is a popular open-source software framework for storing and processing large sets of data on a platform consisting of clusters of commodity hardware. The main idea behind Hadoop is to move computation to the data, instead of the traditional way of moving data to computation. Scalability lies at the heart of Hadoop, and one of the big reasons for its considerable popularity in the big data world we live in today is its extreme cost effectiveness owing to the use of commodity servers and open-source software.

I started working on this book in the fall of 2014. Hadoop 2 had come out a few months earlier, and there were numerous interesting changes in the Hadoop architecture in the new release. There was one very good book on administering generic (without the use of a third-party vendor's tools) Hadoop clusters (*Hadoop Operations* by Eric Sammer), but, over time, it became outdated in several areas (it was published in 2012). Tom White's book *Hadoop: The Definitive Guide* of course is wonderful, and it contains several useful discussions pertaining to Hadoop administration, but it's a book more geared toward developers and architects than cluster administrators. I decided to write this book to provide Hadoop users a comprehensive guide to administering, securing, and optimizing their Hadoop clusters.

As I progressed with the book, Spark became the most important processing framework for Hadoop. I therefore added four chapters to discuss the architecture of Spark, the nature of Spark applications and how to manage and optimize Spark jobs running in a Hadoop cluster.

In this book, I explain how to manage, optimize, and secure Hadoop environments by working directly with the Hadoop configuration files. You may wonder if you really need to learn how to administer Hadoop from the ground level up. Like many of the people that manage Hadoop environments, I use third-party Hadoop distributions such as Cloudera and Hortonworks. Of course, using a tool such as Cloudera Manager or Apache Ambari to manage a Hadoop cluster makes your life really easy. However, I realized that in order to master Hadoop environments, and to get the most out of your Hadoop cluster, you must understand what actually happens behind the scenes when you work with a management tool to administer your cluster. This is possible only if you learn how to build a cluster from scratch and learn how to configure it for various purposes—high availability, performance, security, encryption—as you go along.

Hadoop comes with a large number of configurable properties. In order to take advantage of Hadoop's powerful capabilities, you must understand the critical performance, security, high-availability and other configuration parameters and know how to tune

them. To this end, I've explained all of the key administration-related Hadoop configuration properties in this book, along with plenty of examples, so you can configure, secure, and optimize your cluster with confidence.

Hadoop is an exciting area to work in, with its interactions with software that fall under the umbrella of the “Hadoop ecosphere.” In this book, my main focus is on core Hadoop itself, specifically on HDFS, the Hadoop distributed file system, and YARN, the processing framework of Hadoop. I do discuss several members of the Hadoop—ecosphere, such as Apache Sqoop, Apache Flume and Apache Spark—but the emphasis is mostly on how to manage the Hadoop infrastructure itself. To this end, I spend quite a bit of time discussing the architecture of both HDFS and YARN in this book.

Who This Book Is For

I wrote this book with the Hadoop administrator in mind. However, you do not need to be a full-time Hadoop administrator to benefit from this book. If you're a big data architect, developer, or analyst, there are several things in this book that'll prove to be of use to you.

How This Book Is Structured and What It Covers

This book is divided into 5 parts, spread over 21 chapters. Following is a chapter-by-chapter summary of what this book covers.

Part I: Introduction to Hadoop—Architecture and Hadoop Clusters

- Chapter 1, “Introduction to Hadoop and Its Environment,” introduces you to Hadoop and big data in general. You learn how Hadoop differs from traditional databases and about the concept of a data lake. You also learn where Hadoop fits in with big data and data science. It also introduces the concept of a Hadoop cluster.

The chapter outlines the roles of the key Hadoop components and members of the Hadoop ecosphere, such as ZooKeeper, Apache Sqoop, Apache Flume and Apache Kafka.

Although Hadoop 1 belongs to history now, it offers a convenient means of tracing the evolution of Hadoop to its current incarnation, especially how it separates processing and scheduling and allows multiple processing engines beyond just MapReduce. I therefore review the key differences between Hadoop 1 and Hadoop 2 to put things in perspective and to help you understand where Hadoop might be headed.

This chapter provides a very brief introduction to MapReduce and Apache Spark, the two main computational frameworks for Hadoop, as well Pig and Hive. The chapter also describes popular Hadoop data ingestion frameworks such as Apache Flume and Apache Kafka. The chapter wraps up with a review of the main areas

of focus for Hadoop administrators, such as resource allocation, job scheduling, performance tuning and security.

- Chapter 2, “An Introduction to the Architecture of Hadoop,” introduces the architecture of Hadoop and explains how HDFS supports data storage and YARN, the other main component of Hadoop, provides the data processing capability.
- Chapter 3, “Creating and Configuring a Simple Hadoop Cluster,” explains, step by step, how to create and configure a single node, pseudo-distributed cluster. While you can’t do a whole lot of big data processing with a single node cluster, I do this so you learn the installation procedures without worrying about setting up multiple nodes right at the beginning. Everything you learn in this chapter carries over to the installation and configuring of a “real,” multinode Hadoop cluster.
- Chapter 4, “Planning for and Creating a Fully Distributed Cluster,” explains how to plan for a Hadoop cluster and how to size one. I show you the step-by-step procedures involved in creating a multinode Hadoop cluster.

Once you learn how to create a Hadoop cluster, you need to know how to modify the default Hadoop configuration. Hadoop comes with a large number of configurable properties for all its capabilities, such as storage, processing, resource allocation and security.

One of the key functions of a Hadoop administrator is to know how to configure, tune and optimize their cluster by setting the correct values for a large number of configuration properties. This chapter shows you how you get started with the configuration of Hadoop. You’ll also learn about how to configure Hadoop services, its web interfaces and the various Hadoop ports.

Part II: Hadoop Application Frameworks

- Chapter 5, “Running Applications in a Cluster—The MapReduce Framework (and Hive and Pig),” explains the main concepts of MapReduce, which for many years was the only major processing framework available in Hadoop. With Hadoop 2, MapReduce isn’t the only processing framework but is still used heavily in many Hadoop environments. The chapter shows the well-known WordCount program and how to run it in MapReduce.

The chapter also introduces you to Apache Hive and Apache Pig, two popular data processing frameworks in many Hadoop shops.

- Chapter 6, “Running Applications in a Cluster—The Spark Framework,” introduces Apache Spark, which is poised to take over from MapReduce as Hadoop’s main processing framework. This chapter focuses on the architecture and installation of Spark, as well as how to load data into Spark from various sources.
- Chapter 7, “Running Spark Applications,” explains what Spark resilient distributed datasets (RDDs) are and shows how to work with them. This chapter also shows

you how to run Spark jobs interactively, through the `spark-submit` command. You also learn the various ways to configure Spark applications and how to monitor Spark applications.

This chapter also introduces Spark Streaming, for handling streaming data, and Spark SQL, for handling structured data.

Part III: Managing and Protecting Hadoop Data and High Availability

- Chapter 8, “The Role of the NameNode and How HDFS Works,” is a deep dive into how the NameNode and the DataNodes interact. You also learn how to configure rack awareness in your cluster.

Data replication is the calling card of HDFS, and you’ll learn about how HDFS organizes its data and how data replication works. You’ll also learn how clients read data from HDFS and write data to HDFS. Finally, this chapter explains the HDFS recovery processes.

Centralized cache management in HDFS offers key benefits, and this chapter explains the concepts of centralized cache management, as well as how to configure caching and manage it.

- Chapter 9, “HDFS Commands, HDFS Permissions and HDFS Storage,” is about managing HDFS storage with HDFS shell commands. You’ll also learn about the `dfsadmin` utility, a key ally in managing HDFS. The chapter also shows how to manage HDFS file permissions and create HDFS users.

As a Hadoop administrator, one of your key tasks is to manage HDFS storage. The chapter shows how to check HDFS usage and how to allocate space quotas to HDFS users. The chapter also discusses when and how to rebalance HDFS data, as well as how you can reclaim HDFS space.

- Chapter 10, “Data Protection, File Formats and Accessing HDFS,” focuses on safeguarding Hadoop data. In addition, the chapter discusses the compression of data and various Hadoop file formats. Finally, the chapter shows you how to access HDFS data through HTTP, using WebHDFS and HttpFS.
- Chapter 11, “NameNode Operations, High Availability and Federation,” starts off with a detailed explanation of NameNode operations. You’ll also learn about the checkpointing process and how to configure it. The chapter explains how the NameNode enters and leaves the safe mode of operations. You’ll also learn how to back up the NameNode metadata, which is absolutely critical for the functioning of a Hadoop cluster.

The chapter explains how to configure HDFS high availability through setting up a Standby NameNode.

Part IV: Moving Data, Allocating Resources, Scheduling Jobs and Security

- In Chapter 12, “Moving Data Into and Out of Hadoop,” you’ll learn how to move data through built-in HDFS file system commands, as well as through the DistCp utility, which enables you to move data between Hadoop clusters.

The chapter shows you how to move data between a Hadoop cluster and a relational database through the Sqoop utility. You’ll also learn how to ingest data from various external sources through Apache Flume and Apache Kafka.

- Chapter 13, “Resource Allocation in a Hadoop Cluster,” explains the topic of resource allocation in a Hadoop cluster. You’ll learn how to configure resource allocation among users and groups through the two main Hadoop built-in schedulers—the Capacity Scheduler and the Fair Scheduler.
- Chapter 14, “Working with Oozie and Hue to Manage Job Workflows,” shows you how to use two very important components of a typical Hadoop environment—Apache Oozie and Apache Hue—to configure jobs and manage them, as well as to access HDFS, and to work with Hive, Pig, Impala and other processing frameworks.
- Chapter 15, “Securing Hadoop,” is about securing Hadoop environments. The main thrust of this chapter is the setting up of authorization through Kerberos, an open-source security framework used widely in Hadoop environments. You’ll also learn how to set up role-based authentication through Apache Sentry.

This chapter also shows you how to audit Hadoop and YARN operations and how to secure Hadoop data through Hadoop’s HDFS Transparent Encryption feature.

Part V: Monitoring, Optimization and Troubleshooting

- Chapter 16, “Managing Jobs, Using Hue and Performing Routine Tasks,” shows you how to use the `yarn` command to monitor and manage jobs. The chapter explains how to perform various routine management tasks such as decommissioning and recommissioning nodes.

The chapter also shows how to set up ResourceManager high availability.

- Chapter 17, “Monitoring, Metrics and Hadoop Logging,” introduces Hadoop metrics and how to make the most of them. There’s a brief review of how to use Ganglia to monitor Hadoop. The chapter discusses the basics of Linux system monitoring.

The chapter reviews the most frequently used Hadoop web UIs to monitor your cluster. Hadoop logging is an important and complex topic, and the chapter shows you how to view various Hadoop-related logs and how to administer logging.

- Chapter 18, “Tuning the Cluster Resources, Optimizing MapReduce Jobs and Benchmarking,” shows how to benchmark the performance of a Hadoop cluster with the TeraSort and the TestDFSIO testing tools.

The chapter's main focus is on configuring a cluster for optimal performance through setting memory and storage parameters in an efficient manner. The chapter shows how to tune the performance of MapReduce jobs, as well as offers pointers for optimizing Hive and Pig jobs.

- Chapter 19, “Configuring and Tuning Apache Spark on YARN,” and the next chapter are dedicated to the configuration and tuning of Apache Spark running on YARN. The chapter also shows how to configure resources for Spark and how to monitor Spark applications.
- Chapter 20, “Optimizing Spark Applications,” discusses the Spark execution model in detail. The chapter explains key aspects of Spark performance such as partitioning, parallelism, data serialization, compression and caching. You'll learn about shuffle operations and how to minimize them.
- Chapter 21, “Troubleshooting Hadoop—A Sampler,” is a brief review of Hadoop troubleshooting. It discusses space- and memory-related issues, such as JVM garbage collection strategies, and common failures that occur in a Hadoop cluster.

Hadoop is an exciting environment to work in, with new processing frameworks and tools coming on board continuously, keeping you on your toes all the time. It's, indeed, quite an exhilarating journey! I've thoroughly enjoyed writing this book, just as I do administering Hadoop clusters. I hope you enjoy reading and using the book as much as I've enjoyed writing it!

Register your copy of *Expert Hadoop® Administration* at informit.com for convenient access to downloads, updates, and corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account. Enter the product ISBN (9780134597195) and click Submit. Once the process is complete, you will find any available bonus content under “Registered Products.”

Acknowledgments

Writing a book is always the work of a team, of which the author is but one of the members. I'd like to acknowledge the immense help provided during the writing of this book by various people, starting with Debra Williams Cauley, executive editor at Addison-Wesley, who oversaw the writing and production of this book. Debra is probably the hardest working and most earnest editor I've worked with, and her dedication to the project and the sense of urgency with which she managed everything has had a huge influence on the way I approached this project, especially towards the later parts of the project.

I owe an immense debt to Chris Zahn for his astute editing of the book, while being enormously kind and graceful throughout the arduous process. Chris's encouragement and support has helped me immensely while writing this book, and the book has gained immeasurably from his skillful editing and his sharp eye for details, without losing the big picture. I've learned quite a few things about correct style and conventions from going over Chris's edits. It's quite unlikely that any major stylistic errors remain after Chris straightened things out, but if they do, you know who to blame!

I've been quite fortunate to have four great reviewers go through the chapters, all of them seasoned professionals from Hortonworks. Anubhav Awasthi, big data consultant, went through all the chapters, caught several errors, and made several important suggestions that helped me improve the book. Karthik Varakantham, system architect, reviewed the book, corrected several stylistic and technical points and made a number of highly useful suggestions. Kannappan Natarasan, senior consultant, reviewed several of the early chapters and provided an overview of how the chapters looked, as well as suggestions to improve the book. Ron Lee, platform engineering architect, made several great suggestions, especially regarding Chapter 15, as well as Chapters 16 through 21. Ron helped me improve the book considerably based on his detailed comments, stemming from his extensive in-the-trenches experience with Hadoop environments.

Earlier on, both Marina Stephens and John Guthrie reviewed and commented in great detail on several chapters in this book. I was able to improve the style as well as the technical content following these reviews. Thank you, Marina and John, for all of your painstaking work, for the many errors you caught, and for all your suggestions that have led me to improve the clarity of a number of topics!

I work as a big data administrator at Sabre, where I'm fortunate to work together with several amazing team members and great managers. I first of all thank Zeelani Shaik, my manager, for his unfailing courtesy, kindness, understanding and encouragement at work. Amjad Saeed was instrumental in bringing me to Sabre, and his cheerfulness,

kindness and grace have always been a source of great pleasure to me. Zul Sidi, vice president for the Enterprise Data and Analytics (EDA) group, is a tremendous leader for the entire team, motivating us and setting a great example by the way he performs his own job. Zul's openness to suggestions for improving the way we do things and his constant encouragement and support has helped me and other members of the EDA team achieve numerous significant objectives for our customers during the short time he has been here.

I also owe a round of thanks to Sujoe Bose for his kindness and help, as well as to Senthil Selvaraj for his help when he worked with me at Sabre. I've learned a lot about Hadoop from Sadu Hegde, and I thank him for helping me get started at Sabre. Mallik Dontula has always been a source of wisdom regarding all matters concerning Hadoop and big data, and I've benefited from his valuable help and suggestions on several occasions. Chris Morris and Larry Pritchett have both been not only good friends, but also truly great professionals, and I've benefited immensely from working with them. I'd also like to thank Aaron Patenaude for his generosity, and his great help with anything I've ever asked of him. I would like to thank my friends Winfield Geng, for his unstinting help whenever I requested it, and Bob Newman, who is conscientious and keeps us on our toes, for his advice and help. I would like to thank both Mohammed Hossain and Andrew Ahmad for their friendship and help. I'd be remiss if I don't acknowledge the great support from my friend and colleague Vinay Shetty, who is not only amazingly good at his job, but also very helpful while working together. I certainly owe many thanks over the past two years to Linda Phipps, for all the things she helped me with, always with great cheer! Lance Tripp was the person who encouraged me to seek the position I currently hold at Sabre. Since I love my job, I must say a big thank you to Lance for his smart recruiting!

Writing a book always means you basically disappear from the home front, although the physical signs of your existence abound. I appreciate the love and kindness of my wife, Valerie, and the affection of my children, Nina, Nicholas and Shannon, who've always supported my writing and research endeavors. Thank you as well to Dale, Shawn and Keith, who always remain close to me. I also appreciate the affection and kindness of the Dixon family: Stephanie Dixon, as well as Clarence and Elaine, who have always been supportive of everything I've done. The kindness and affection of my brothers Hari and Bujji, my sisters-in-law Aruna and Vanaja, my nieces Aparna and Soumya, and my nephew Teja means everything to me, so thank you to all of you! Special thanks to my nephew Ashwin and his wife SreeVidya, whose kind hospitality during my stay in San Jose during a Hadoop conference helped me develop several key ideas I discuss in this book.

About the Author

Sam R. Alapati is a principal Hadoop administrator at Sabre, headquartered in Southlake, Texas, where he works with multiple Hadoop clusters on a daily basis. As part of his responsibilities as the point person for all Hadoop administration–related work for the Enterprise Data Analytics (EDA) group at Sabre, Sam manages and optimizes multiple critical data science and data analysis related Hadoop job flows. Sam is also an expert Oracle Database administrator, and his vast knowledge of relational databases and SQL contributes to his success in working with Hadoop–related projects. Sam’s accomplishments in the database and middleware area include the publication of 18 well-received books over the past 14 years, mostly on Oracle Database administration and Oracle Weblogic Server. Sam is also the author of a forthcoming book titled *Modern Linux Administration* (O’Reilly, 2017). Sam’s experience dealing with numerous configuration, architecture and performance–related Hadoop issues over the years led him to the realization that many working Hadoop administrators and developers would appreciate having a handy reference, such as this book, to turn to when creating, managing, securing and optimizing their Hadoop infrastructure.

This page intentionally left blank



Managing and Protecting Hadoop Data and High Availability

This page intentionally left blank

HDFS Commands, HDFS Permissions and HDFS Storage

This chapter covers the following:

- Working with HDFS
- Using HDFS shell commands
- Managing HDFS permissions and users
- Managing HDFS storage (including rebalancing of data)
- Granting users permissions and quotas

Working with HDFS is one of the most common tasks for someone administering a Hadoop cluster. Although you can access HDFS in multiple ways, the command line is the most common way to administer HDFS storage.

Managing HDFS users by granting them appropriate permissions and allocating HDFS space quotas to users are some of the common user-related administrative tasks you'll perform on a regular basis. The chapter shows how HDFS permissions work and how to grant and revoke space quotas on HDFS directories.

Besides the management of users and their HDFS space quotas, there are other aspects of HDFS that you need to manage. This chapter also shows how to perform maintenance tasks such as periodically balancing the HDFS data to distribute it evenly across the cluster, as well as how to gain additional space in HDFS when necessary.

Managing HDFS through the HDFS Shell Commands

You can access HDFS in various ways:

- From the command line using simple Linux-like file system commands, as well as through a web interface, called WebHDFS
- Using the HttpFS gateway to access HDFS from behind a firewall

- Through Hue’s File Browser (and Cloudera Manager and Ambari, if you’re using Cloudera, or Hortonwork’s Hadoop distributions)

Figure 9.1 summarizes the various ways in which you can access HDFS. Although you have multiple ways to access HDFS, it’s a good bet that you’ll often be working from the command line to manage your HDFS files and directories. You can access the HDFS file system from the command line with the `hdfs dfs` file system commands.

File Systems other than HDFS

It’s important to keep in mind that HDFS file systems are only one way that Hadoop implements a file system. There are several other Java implementations of file systems that work with Hadoop. These include local file systems (`file`), WebHDFS (WebHDFS), HAR (Hadoop archive files), View (viewfs), S3 (`s3a`) and others. For each file system, Hadoop uses a different URI scheme for the file system instance in order to connect with it. For example, you list the files in the local system by using the `file` URI scheme, as shown here:

```
$ hdfs dfs -ls file:///
```

This will get you a listing of files stored on the local Linux file system.

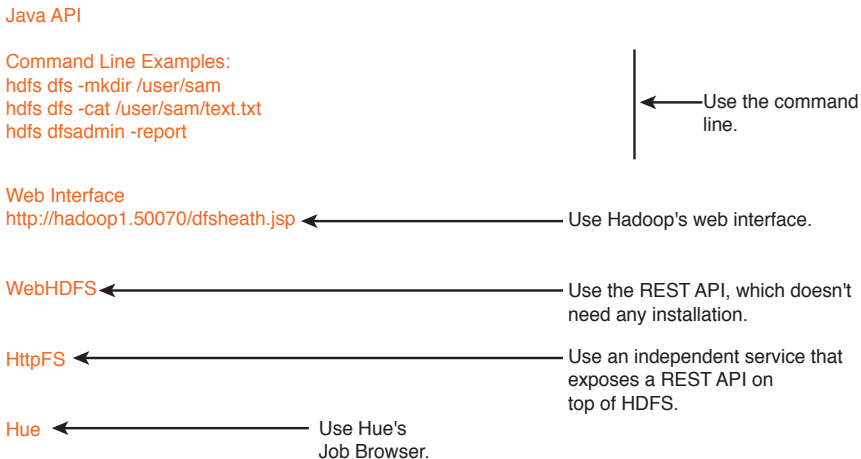


Figure 9.1 The many ways in which you can access HDFS

Using the `hdfs dfs` Utility to Manage HDFS

You use the `hdfs dfs` utility to issue HDFS commands in Hadoop. Here's the usage of this command:

```
hdfs dfs [GENERIC_OPTIONS] [COMMAND_OPTIONS]
```

Using the `hdfs dfs` utility, you can run file system commands on the file system supported in Hadoop, which happens to be HDFS.

You can use two types of HDFS shell commands:

- The first set of shell commands are very similar to common Linux file system commands such as `ls`, `mkdir` and so on.
- The second set of HDFS shell commands are specific to HDFS, such as the command that lets you set the file replication factor.

You can access the HDFS file system from the command line, over the web, or through application code. HDFS file system commands are in many cases quite similar to familiar Linux file system commands. For example, the command `hdfs dfs -cat /path/to/hdfs/file` works the same as a Linux `cat` command, by printing the output of a file onto the screen.

Internally HDFS uses a pretty sophisticated algorithm for its file system reads and writes, in order to support both reliability and high throughput. For example, when you issue a simple `put` command that writes a file to an HDFS directory, Hadoop will need to write that data fast to three nodes (by default).

You can access the HDFS shell by typing `hdfs dfs <command>` at the command line. You specify actions with subcommands that are prefixed with a minus (-) sign, as in `dfs -cat` for displaying a file's contents.

You may view all available HDFS commands by simply invoking the `hdfs dfs` command with no options, as shown here:

```
$ hdfs dfs
Usage: hadoop fs [generic options]
      [-appendToFile <localsrc> ... <dst>]
      [-cat [-ignoreCrc] <src> ...]
```

Figure 9.2 shows all the available HDFS `dfs` commands.

However, it's the `hdfs dfs -help` command that's truly useful to a beginner and even quite a few "experts"—this command clearly explains all the `hdfs dfs` commands. Figure 9.3 shows how the help utility clearly explains the various file copy options that you can use with the `hdfs dfs` command.

Note

Several Linux file and directory commands have analogs in HDFS. These include the familiar `ls`, `cp` and `mv` commands. However, a big difference between Linux file and HDFS file system commands is that there are no directory-location-related commands in HDFS. For example, there's no HDFS `pwd` command or `cd` command.

```

bash-3.2$ hdfs dfs
Usage: hadoop fs [generic options]
[-appendToFile <localsrc> ... <dst>]
[-cat [-ignoreCrc] <src> ...]
[-checksum <src> ...]
[-chgrp [-R] GROUP PATH...]
[-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
[-chown [-R] [OWNER][:[GROUP]] PATH...]
[-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst>]
[-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
[-count [-q] [-h] <path> ...]
[-cp [-f] [-p | -p[topax]] <src> ... <dst>]
[-createSnapshot <snapshotDir> [<snapshotName>]]
[-deleteSnapshot <snapshotDir> <snapshotName>]
[-df [-h] [<path> ...]]
[-du [-s] [-h] <path> ...]
[-expunge]
[-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
[-getfacl [-R] <path>]
[-getfattr [-R] {-n name | -d} [-e en] <path>]
[-getmerge [-nl] <src> <localdst>]
[-help [cmd ...]]
[-ls [-d] [-h] [-R] [<path> ...]]
[-mkdir [-p] <path> ...]
[-moveFromLocal <localsrc> ... <dst>]
[-moveToLocal <src> <localdst>]
[-mv <src> ... <dst>]
[-put [-f] [-p] [-l] <localsrc> ... <dst>]
[-renameSnapshot <snapshotDir> <oldName> <newName>]
[-rm [-f] [-r|-R] [-skipTrash] <src> ...]
[-rmdir [--ignore-fail-on-non-empty] <dir> ...]
[-setfacl [-R] [{-b|-k} {-m|-x <acl_spec>} <path>]|[--set <acl_spec> <pa
th>]]
[-setfattr {-n name [-v value] | -x name} <path>]
[-setrep [-R] [-w] <rep> <path> ...]
[-stat [format] <path> ...]
[-tail [-f] <file>]
[-test [-d] [-d[efsz]] <path>]
[-text [-ignoreCrc] <src> ...]
[-touchz <path> ...]

```

Figure 9.2 The `hdfs dfs` commands.

```

-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst> :
  Identical to the -put command.

-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst> :
  Identical to the -get command.

-count [-q] [-h] <path> ... :
  Count the number of directories, files and bytes under the paths
  that match the specified file pattern. The output columns are:
  DIR COUNT FILE COUNT CONTENT SIZE FILE NAME or
  QUOTA REMAINING QUOTA SPACE QUOTA REMAINING SPACE QUOTA
  DIR COUNT FILE COUNT CONTENT SIZE FILE NAME
  The -h option shows file sizes in human readable format.

-cp [-f] [-p | -p[topax]] <src> ... <dst> :
  Copy files that match the file pattern <src> to a destination. When copying
  multiple files, the destination must be a directory. Passing -p preserves status
  [topax] (timestamps, ownership, permission, ACLs, XAttr). If -p is specified
  with no <arg>, then preserves timestamps, ownership, permission. If -pa is
  specified, then preserves permission also because ACL is a super-set of
  permission. Passing -f overwrites the destination if it already exists. raw
  namespace extended attributes are preserved if (1) they are supported (HDFS
  only) and, (2) all of the source and target pathnames are in the /.reserved/raw
  hierarchy. raw namespace xattr preservation is determined solely by the presence
  (or absence) of the /.reserved/raw prefix and not by the -p option.

```

Figure 9.3 How the `hdfs dfs -help` command helps you understand the syntax of the various options of the `hdfs dfs` command

In the following sections, I show you how to

- List HDFS files and directories
- Use the HDFS `STAT` command
- Create an HDFS directory
- Remove HDFS files and directories
- Change file and directory ownership
- Change HDFS file permissions

Listing HDFS Files and Directories

As with regular Linux file systems, use the `ls` command to list HDFS files. You can specify various options with the `ls` command, as shown here:

```
$ hdfs dfs -usage ls
Usage: hadoop fs [generic options] -ls [-d] [-h] [-R] [<path> ...]
bash-4.2$
Here's what the options stand for:
-d: Directories are listed as plain files.
-h: Format file sizes in a human-readable fashion (eg 64.0m instead of 67108864).
-R: Recursively list subdirectories encountered.
-t: Sort output by modification time (most recent first).
-S: Sort output by file size.
-r: Reverse the sort order.
-u: Use access time rather than modification time for display and sorting.
```

Listing Both Files and Directories

If the target of the `ls` command is a file, it shows the statistics for the file, and if it's a directory, it lists the contents of that directory. You can use the following command to get a directory listing of the HDFS root directory:

```
$ hdfs dfs -ls /
Found 8 items
drwxr-xr-x - hdfs hdfs          0 2013-12-11 09:09 /data
drwxr-xr-x - hdfs supergroup    0 2015-05-04 13:22 /lost+found
drwxrwxrwt - hdfs hdfs          0 2015-05-20 07:49 /tmp
drwxr-xr-x - hdfs supergroup    0 2015-05-07 14:38 /user
...
#
```

For example, the following command shows all files within a directory ordered by filenames:

```
$ hdfs dfs -ls /user/hadoop/testdir1
```

Alternately, you can specify the HDFS URI when listing files:

```
$ hdfs dfs -ls hdfs://<hostname>:9000/user/hdfs/dir1/
```

You can also specify multiple files or directories with the `ls` command:

```
$ hdfs dfs -ls /user/hadoop/testdir1 /user/hadoop/testdir2
```

Listing Just Directories

You can view information that pertains just to directories by passing the `-d` option:

```
$ hdfs dfs -ls -d /user/alapati
drwxr-xr-x - hdfs supergroup          0 2015-05-20 12:27 /user/alapati
$
```

The following two `ls` command examples show file information:

```
$ hdfs dfs -ls /user/hadoop/testdir1/test1.txt
$ hdfs dfs -ls /hdfs://<hostname>:9000/user/hadoop/dir1/
```

Note that when you list HDFS files, each file will show its replication factor. In this case, the file `test1.txt` has a replication factor of 3 (the default replication factor).

```
$ hdfs dfs -ls /user/alapati/
-rw-r--r-- 3 hdfs supergroup      12 2016-05-24 15:44 /user/alapati/test.txt
```

Using the `hdfs stat` Command to Get Details about a File

Although the `hdfs dfs -ls` command lets you get the file information you need, there are times when you need specific bits of information from HDFS. When you run the `hdfs dfs -ls` command, it returns the complete path of the file. When you want to see only the base name, you can use the `hdfs -stat` command to view only specific details of a file.

You can format the `hdfs -stat` command with the following options:

```
%b Size of file in bytes
%F Will return "file", "directory", or "symlink" depending on the type of inode
%g Group name
%n Filename
%o HDFS Block size in bytes ( 128MB by default )
%r Replication factor
%u Username of owner
%y Formatted mtime of inode
%Y UNIX Epoch mtime of inode
```

In the following example, I show how to confirm if a file or directory exists.

```
# hdfs dfs -stat "%n" /user/alapati/messages
messages
```

If you run the `hdfs -stat` command against a directory, it tells you that the name you specify is indeed a directory.

```
$ hdfs dfs -stat "%b %F %g %n %o %r %u %y %Y" /user/alapati/test2222
0 directory supergroup test2222 0 0 hdfs 2015-08-24 20:44:11 1432500251189
$
```

The following examples show how you can view different types of information with the `hdfs dfs -stat` command when compared to the `hdfs dfs -ls` command. Note that I specify all the `-stat` command options here.

```
$ hdfs dfs -ls /user/alapati/test2222/true.txt
-rw-r--r-- 2 hdfs supergroup      12 2015-08-24 15:44 /user/alapati/test2222/true.txt
$
```

```
$ hdfs dfs -stat "%b %F %g %n %o %r %u %y %Y" /user/alapati/test2222/true.txt
12 regular file supergroup true.txt 268435456 2 hdfs 2015-05-24 20:44:11 1432500251189
$
```

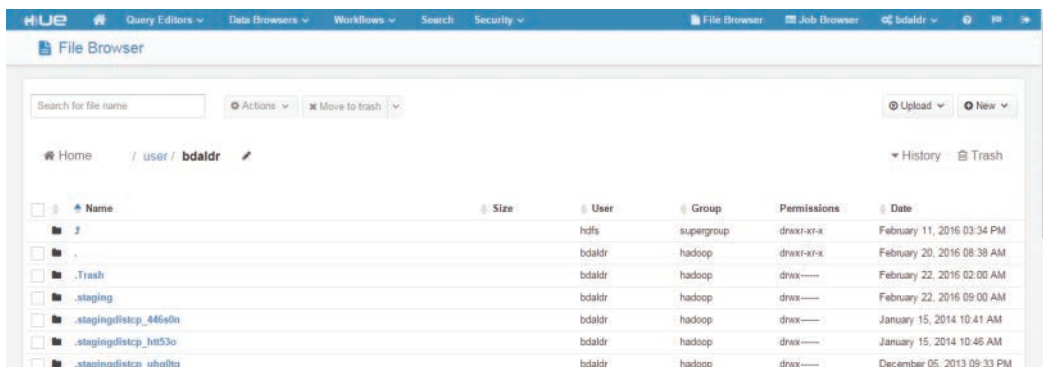


Figure 9.4 Hue's File Browser, showing how you can access HDFS from Hue

I'd be remiss if I didn't add that you can also access HDFS through Hue's Job Browser, as shown in Figure 9.4.

Creating an HDFS Directory

Creating an HDFS directory is similar to how you create a directory in the Linux file system. Issue the `mkdir` command to create an HDFS directory. This command takes path URIs as arguments to create one or more directories, as shown here:

```
$ hdfs dfs -mkdir /user/hadoop/dir1 /user/hadoop/dir2
```

The directory `/user/hadoop` must already exist for this command to succeed.

Here's another example that shows how to create a directory by specifying a directory with a URI.

```
$ hdfs dfs -mkdir hdfs://nn1.example.com/user/hadoop/dir
```

If you want to create parent directories along the path, specify the `-p` option, with the `hdfs dfs -mkdir` command, just as you would do with its cousin, the Linux `mkdir` command.

```
$ hdfs dfs -mkdir -p /user/hadoop/dir1
```

In this command, by specifying the `-p` option, I create both the parent directory `hadoop` and its subdirectory `dir1` with a single `mkdir` command.

Removing HDFS Files and Directories

HDFS file and directory removal commands work similar to the analogous commands in the Linux file system. The `rm` command with the `-R` option removes a directory and everything under that directory in a recursive fashion. Here's an example.

```
$ hdfs dfs -rm -R /user/alapati
15/05/05 12:59:54 INFO fs.TrashPolicyDefault: Namenode trash configuration:
Deletion interval = 1440 minutes, Emptier interval = 0 minutes.
Moved: 'hdfs://hadoop01-ns/user/alapati' to trash at: hdfs://hadoop01-ns/user/
hdfs/.Trash/Current
$
```

I issued an `rm -R` command, and I can verify that the directory I want to remove is indeed gone from HDFS. However, the output of the `rm -R` command shows that the directory is still saved for me in case I need it—in HDFS’s trash directory. The trash directory serves as a built-in safety mechanism that protects you against accidental file and directory removals. If you haven’t already enabled trash, please do so ASAP!

Even when you enable trash, sometimes the trash interval is set too low, so make sure that you configure the `fs.trash.interval` parameter in the `hdfs-site.xml` file appropriately. For example, setting this parameter to 14,400 means Hadoop will retain the deleted items in trash for a period of ten days.

You can view the deleted HDFS files currently in the trash directory by issuing the following command:

```
$ hdfs dfs -ls /user/sam/.Trash
```

You can use the `-rmdir` option to remove an empty directory:

```
$ hdfs dfs -rmdir /user/alapati/testdir
```

If the directory you wish to remove isn’t empty, use the `-rm -R` option as shown earlier.

If you’ve configured HDFS trash, any files or directories that you delete are moved to the trash directory and retained in there for the length of time you’ve configured for the trash directory. On some occasions, such as when a directory fills up beyond the space quota you assigned for it, you may want to permanently delete files immediately. You can do so by issuing the `dfs -rm` command with the `-skipTrash` option:

```
$ hdfs dfs -rm /user/alapati/test -skipTrash
```

The `-skipTrash` option will bypass the HDFS trash facility and immediately delete the specified files or directories.

You can empty the trash directory with the `expunge` command:

```
$ hdfs dfs -expunge
```

All files in trash that are older than the configured time interval are deleted when you issue the `expunge` command.

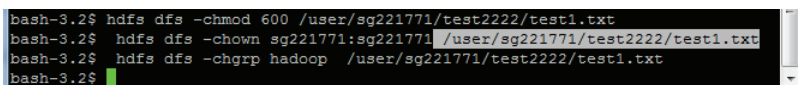
Changing File and Directory Ownership and Groups

You can change the owner and group names with the `-chown` command, as shown here:

```
$ hdfs dfs -chown sam:producers /data/customers/names.txt
```

You must be a super user to modify the ownership of files and directories.

HDFS file permissions work very similar to the way you modify file and directory permissions in Linux. Figure 9.5 shows how to issue the familiar `chmod`, `chown` and `chgrp` commands in HDFS.



```
bash-3.2$ hdfs dfs -chmod 600 /user/sg221771/test222/test1.txt
bash-3.2$ hdfs dfs -chown sg221771:sg221771 /user/sg221771/test222/test1.txt
bash-3.2$ hdfs dfs -chgrp hadoop /user/sg221771/test222/test1.txt
bash-3.2$
```

Figure 9.5 Changing file mode, ownership and group with HDFS commands

Changing Groups

You can change just the group of a user with the `chgrp` command, as shown here:

```
$ sudo -u hdfs hdfs dfs -chgrp marketing /users/sales/markets.txt
```

Changing HDFS File Permissions

You can use the `chmod` command to change the permissions of a file or directory. You can use standard Linux file permissions. Here's the general syntax for using the `chmod` command:

```
hdfs dfs -chmod [-R] <mode> <file/dir>
```

You must be a super user or the owner of a file or directory to change its permissions.

With the `chgrp`, `chmod` and `chown` commands you can specify the `-R` option to make recursive changes through the directory structure you specify.

In this section, I'm using HDFS commands from the command line to view and manipulate HDFS files and directories. However, there's an even easier way to access HDFS, and that's through Hue, the web-based interface, which is extremely easy to use and which lets you perform HDFS operations through a GUI. Hue comes with a File Browser application that lets you list and create files and directories, download and upload files from HDFS and copy/move files. You can also use Hue's File Browser to view the output of your MapReduce jobs, Hive queries and Pig scripts.

While the `hdfs dfs` utility lets you manage the HDFS files and directories, the `hdfs dfsadmin` utility lets you perform key HDFS administrative tasks. In the next section, you'll learn how to work with the `dfsadmin` utility to manage your cluster.

Using the `dfsadmin` Utility to Perform HDFS Operations

The `hdfs dfsadmin` command lets you administer HDFS from the command line. While the `hdfs dfs` commands you learned about in the previous section help you manage HDFS files and directories, the `dfsadmin` command is useful for performing general HDFS-specific administrative tasks. It's a good idea to become familiar with all the options that are available for the `dfsadmin` utility by issuing the following command:

```
$ hdfs dfsadmin -help
hdfs dfsadmin performs DFS administrative commands.
Note: Administrative commands can only be run with superuser permission.
The full syntax is:
hdfs dfsadmin
    [-report [-live] [-dead] [-decommissioning]]
    [-safemode <enter | leave | get | wait>]
    [-saveNamespace]
...
$
```

```

[-triggerBlockReport [-incremental] <datanode_host:ipc_port>]
[-help [cmd]]

-report [-live] [-dead] [-decommissioning]:
  Reports basic filesystem information and statistics.
  Optional flags may be used to filter the list of displayed DNs.

-safemode <enter|leave|get|wait>: Safe mode maintenance command.
  Safe mode is a Namenode state in which it
  1. does not accept changes to the name space (read-only)
  2. does not replicate or delete blocks.
  Safe mode is entered automatically at Namenode startup, and
  leaves safe mode automatically when the configured minimum
  percentage of blocks satisfies the minimum replication
  condition. Safe mode can also be entered manually, but then
  it can only be turned off manually as well.

-saveNamespace: Save current namespace into storage directories and reset edits log.
  Requires safe mode.

-rollEdits: Rolls the edit log.

-restoreFailedStorage: Set/Unset/Check flag to attempt restore of failed storage replicas
  if they become available.

-refreshNodes: Updates the namenode with the set of datanodes allowed to connect to the
  namenode.

```

Figure 9.6 The `dfsadmin -help` command reveals useful information for each `dfsadmin` command.

Note

You've already seen a couple of the `dfsadmin` administrative commands in action (such as `dfsadmin -report` and `dfsadmin -printTopology`) in earlier chapters. This book explains the rest of the `dfsadmin` commands in the appropriate context in various chapters.

If you issue the `dfsadmin` command with no options, it will list all the options that you can specify with the command. The `dfsadmin -help` command is highly useful, since it not only lists the command options, but also shows you what they are for and their syntax as well. Figure 9.6 shows a portion of the `dfsadmin -help` command.

There are several useful `dfsadmin` command options. In the next few sections, let's look at the following command options (other sections of this chapter and other chapters will discuss several other command options).

- `dfsadmin -report`
- `dfsadmin -refreshNodes`
- `dfsadmin -metasave`

The `dfsadmin -report` Command

The `dfsadmin` tool helps you examine the HDFS cluster status. The `dfsadmin -report` command produces useful output that shows basic statistics of the cluster, including the

status of the DataNodes and NameNode, the configured disk capacity and the health of the data blocks. Here's a sample `dfsadmin -report` command:

```
$ hdfs dfsadmin -report

Configured Capacity: 2068027170816000 (1.84 PB)           #A
Present Capacity: 2068027170816000 (1.84 PB)
DFS Remaining: 562576619120381 (511.66 TB)             #A
DFS Used: 1505450551695619 (1.34 PB)                   #B
DFS Used%: 72.80%                                       #B
Under replicated blocks: 1                              #C
Blocks with corrupt replicas: 0
Missing blocks: 1
Missing blocks (with replication factor 1): 9           #C

-----
Live datanodes (54):                                    #D

Name: 10.192.0.78:50010 (hadoop02.localhost)            #E
Hostname: hadoop02.localhost.com
Rack: /rack3                                           #E
Decommission Status : Normal                          #F
Configured Capacity: 46015524438016 (41.85 TB)        #G
DFS Used: 33107988033048 (30.11 TB)
Non DFS Used: 0 (0 B)
DFS Remaining: 12907536404968 (11.74 TB)
DFS Used%: 71.95%
DFS Remaining%: 28.05%                                #G
Configured Cache Capacity: 4294967296 (4 GB)          #H
Cache Used: 0 (0 B)
Cache Remaining: 4294967296 (4 GB)
Cache Used%: 0.00%
Cache Remaining%: 100.00%                             #H
Xceivers: 71
Last contact: Fri May 01 15:15:59 CDT 2015

...

```

Notes

- #A Configured capacity for HDFS in this cluster
- #B HDFS used storage statistics
- #C Shows if there are any under-replicated, corrupt or missing blocks
- #D Shows how many DataNodes in the cluster are alive and available
- #E The hostname and rack name
- #F Status of the DataNode (decommissioned or not)
- #G Configured and used capacity for this DataNode
- #H Cache usage statistics (if configured)

Note

You can view the same information as that shown by the `dfsadmin -report` command on the NameNode web status page, which is at <http://<namenode IP>:50070/dfshealth.jsp>.

The `dfsadmin -report` command shows HDFS details for the entire cluster, as well as separately for each node in the cluster. The output of the DFS command shows the following at the cluster and the individual DataNode levels:

- A summary of the HDFS storage allocation, including information about the configured, used and remaining space
- If you've configured centralized HDFS caching, the used and remaining percentages of cache
- Missing, corrupted and under-replicated blocks

As you'll learn later in this book, the `dfsadmin -report` command's output helps greatly in examining how balanced the HDFS data is, as well as helps you find out the extent of HDFS corruption (if it exists).

The `dfsadmin -refreshNodes` Command

The `dfsadmin -refreshNodes` command updates the NameNode with the list of DataNodes that are allowed to connect to the NameNode.

The NameNode reads the hostnames of the DataNode from the files pointed to by the `dfs.hosts` and the `dfs.hosts.exclude` configuration parameters in the `hdfs-site.xml` file. The `dfs.hosts` file lists all the hosts that are allowed to register with the NameNode. Any entries in the `dfs.hosts.exclude` file point to DataNodes that need to be decommissioned (you finalize the decommissioning after all the replicas from the node that is being decommissioned are replicated to other DataNodes).

The `dfsadmin -metasave` Command

The `dfsadmin -metasave` command provides more information than that provided by the `dfsadmin -report` command. This command gets you various block-related pieces of information such as:

- Total number of blocks
- Blocks waiting for replication
- Blocks that are currently being replicated

Here's how you run the `dfsadmin -metasave` command:

```
$ sudo -u hdfs hdfs dfsadmin -metasave test.txt
Created metasave file test.txt in the log directory of namenode hadoop1
.localhost.com/10.192.2.21:8020
Created metasave file test.txt in the log directory of namenode hadoop02
.localhost.com/10.192.2.22:8020
$
```

When you run the `dfsadmin -metasave` command, it creates a file in the `/var/log/hadoop-hdfs` directory on the server where you executed the command. The output file will contain the following information regarding the blocks:

```
58 files and directories, 17 blocks = 75 total
Live Datanodes: 1
Dead Datanodes: 0
```

```
Metasave: Blocks waiting for replication: 0
Mis-replicated blocks that have been postponed:
Metasave: Blocks being replicated: 0
Metasave: Blocks 0 waiting deletion from 0 datanodes.
Metasave: Number of datanodes: 1
127.0.0.1:50010 IN 247241674752(230.26 GB) 323584(316 KB) 0% 220983930880(205.81 GB)
Sat May 30 18:52:49 PDT 2015
```

Managing HDFS Permissions and Users

HDFS as a file system is somewhat similar to the POSIX file system in terms of the file permissions it requires. However, HDFS doesn't have the concept of users and groups as in the other file systems. It's important to understand the nature of the HDFS super user and how to manage the granting of permissions to users. You also need to learn how to set up users so they're ready to read data and write to the HDFS file system.

In the following sections, I explain these topics:

- HDFS file permissions
- Creating HDFS users

HDFS File Permissions

In a Linux system, you create OS users and make them members of an existing operating system group. In Hadoop, you associate a directory with an owner and a group. You need not actually “create” either the users or the groups. Rather, you use the concept of users and groups to set file and directory permissions. The following sections show how file and directory permissions work in HDFS.

HDFS Permission Checking

The HDFS configuration parameter `dfs.permissions.enabled` in the `hdfs-site.xml` file determines whether permission checking is enabled in HDFS:

```
<property>
<name>dfs.permissions.enabled</name>
<value>true</value>
</property>
```

The default value of the parameter is `true`, meaning permission checking is enabled. If you set this parameter to `false`, you turn HDFS permission checking off. Obviously, you can do this in a development environment to overcome frequent permission-related error messages, but in a production cluster, you need to keep it at its default setting.

HDFS File and Directory Permissions

HDFS uses a symbolic notation (r, w) to denote the read and write permissions, just as a Linux operating system does.

- When a client accesses a directory, if the client is the same as the directory's owner, Hadoop tests the owner's permissions.
- If the group matches the directory's group, then Hadoop tests the user's group permissions.

- If neither the owner nor the group names match, Hadoop tests the “other” permission of the directory.
- If none of the permissions checks succeed, the client’s request is denied.

Although there’s an **execute** (x) permission for a file, it’s ignored for files, and as far as directories go, the execute permission implies that you can access the subdirectories of that directory. Unlike in the underlying Linux operating system, Hadoop has nothing like the UIDs (User IDs) or GIDs (Group IDs) to identify users and groups. HDFS simply stores users and groups of a directory or file as strings.

A user can write to an HDFS directory only if that user has the correct permissions. In this example, the Linux root user tries to copy a file to a user’s HDFS directory and fails due to lack of permissions.

```
[root@hadoop01]# hdfs dfs -put test.txt /user/alapati/test222/
put: Permission denied: user=root, access=WRITE, inode="/user/alapati/
test222":hdfs:supergroup:drwxr-xr-x
[root@hadoop01]#
```

Permission Denied Errors in HDFS

You may receive the permission denied error when you’re issuing an HDFS command from the command line, as in the previous example, or even when you’re trying to browse the HDFS file system through the NameNode web page. For example, you may receive the following error when you try to browse files through the web UI.

```
Permissiondenied:user=alapati,access=READ_EXECUTE,inode="/user":hadoop:hdfs:drwx.-----
```

In this case, you need to change the access privileges on the HDFS directory /user, after logging in as the user hdfs, from the command line:

```
$ hdfs dfs -chmod -R 755 /user
```

Running administrative commands as the root user or any other non-privileged (from the perspective of Hadoop) user will result in errors. If you run the Hadoop file system checking command fsck as the root user, you’ll get the following error:

```
$ su root
$ hdfs fsck /
...
FSCK ended at Sun May 29 14:46:27 CDT 2016 in 39473 milliseconds
Permissiondenied:user=root,access=READ_EXECUTE,inode="/lost+found/user":hdfs:supergroup:drwxr--r--
Fsck on path '/' FAILED
#
```

The FAILED result you get from running the fsck command here doesn’t mean the file system is corrupt! It simply means that you failed to execute the fsck command. A similar thing happens when you run the dfsadmin -report command as any user other than the HDFS super user, hdfs:

```
$ hdfs dfsadmin -report
-----
report: Access denied for user root. Superuser privilege is required
#
```

In both the cases described here, the right thing to do is to either log in as the user `hdfs` and execute the commands, or if you have the `sudo` privileges to the `hdfs` user account, run the commands as follows:

```
$ sudo -u hdfs hdfs fsck /
$ sudo -u hdfs hdfs dfsadmin -report
```

Using Access Control Lists (ACLs) to control permissions

Unlike the regular Linux or UNIX permissions mode, Access Control Lists (ACLs) let you define permissions for some of a group's members. For example, you can grant or deny write permissions on a file only to specific users or groups. ACLs are disabled by default, but you can enable them by configuring the NameNode appropriately with the `dfs.namenode.acls.enabled` configuration parameter.

Chapter 15, "Securing Hadoop," which deals with Hadoop security, discusses ACLs in more detail.

HDFS Users and Super Users

Typically, database administrators create users in their databases, with each user having specific privileges and/or roles that enable them to perform various actions in the database. In the context of Hadoop, *creating* a user is kind of a misnomer, as HDFS really doesn't have anything that lets you create user identities as you would on Linux systems. It also doesn't enable you to create any groups.

In the default mode of authentication, called simple authentication, Hadoop relies on the underlying operating system to determine client identities. If you set up a **Kerberized** system (a system that has been set up to authenticate connections through Kerberos), then Kerberos will determine the client identities. Chapter 15 shows how to set up Kerberos for user authentication.

Note that you don't need to create an operating system account on the underlying Linux system for your HDFS users to be able to access and use HDFS. It's a good practice to create OS accounts for all Hadoop users who'll be using the local file system on the gateway servers for their Hadoop-related work.

Creating HDFS (and Hadoop) Users

In order to enable new users to use your Hadoop cluster, follow these general steps.

1. Create an OS account on the Linux system from which you want to let a user execute Hadoop jobs. Before creating the user, you may have to create the group as well:

```
$ group add analysts
$ useradd -g analysts alapati
$ passwd alapati
```

Here, **analysts** is an OS group I've created for a set of users. The `passwd` command lets me set a password for the user.

2. Make sure that you've set the permissions on the Hadoop temp directory you've specified in the **core-site.xml** file, so all Hadoop users can access it:

```
<property>
  <name>hadoop.tmp.dir</name>
  <value>/tmp/hadoop-$(user.name)</value>
</property>
```

3. If the file permissions on the HDFS temp directory aren't 777, make them so:

```
$ hdfs -dfs -chmod -R 777 //tmp/hadoop-alapati
```

4. In order to “create” a new HDFS user, you need to create a directory under the /user directory. This directory will serve as the HDFS “home” directory for the user.

```
$ hdfs dfs -mkdir /user/alapati
```

5. By default, when you create a directory or a file, the owner is the user that creates the directory (or file) and the group is the group of that user, as shown here.

```
# sudo -u hdfs
# hdfs dfs -ls /user
Found 135 items
drwxr-xr-x  - hdfs      supergroup          0 2016-05-28 08:18 /user/alapati
....
```

In this case, I used the hdfs account to create the directory, so the owner is hdfs and the group is supergroup. Change the ownership of the directory, since you don't want to use the default owner/group (hdfs/supergroup) for this directory.

```
$ su hdfs
$ hdfs dfs -chown -R alapati:analysts
$ hdfs dfs -ls /user/
$ drwxr-xr-x  - alapati  analysts          0 2016-04-27 12:40 /user/alapati
```

6. You can check the new directory structure for the user with the following command:

```
$ hdfs dfs -ls /user/alapati
```

User alapati can now store the output of his MapReduce and other jobs under that user's home directory in HDFS.

7. Refresh the user and group mappings to let the NameNode know about the new user:

```
$ hdfs dfsadmin -refreshUserToGroupMappings
```

8. Set a space quota for the new directory you've created:

```
$ hdfs dfsadmin -setSpaceQuota 30g /user/alapati
```

The new user can now log into the gateway servers and execute his or her Hadoop jobs and store data in HDFS.

User Identities

Hadoop supports two modes of operation—**simple** and **Kerberos**—to determine user identities. The simple mode of operation is the default. You specify the mode of operation with the `hadoop.security.authentication` property in the `hdfs-site.xml` file.

When operating in a non-Kerberos (or non-Kerberized) cluster, the host operating system determines the client identities. In a Kerberized cluster, user identities are based on the user's Kerberos credentials, as explained in Chapter 15. Users determine their current Kerberos principal through the `kinit` utility, and the Kerberos principal is then mapped to an HDFS username.

The HDFS Super User

Since Hadoop doesn't have the concept of a user identity, there's no fixed super user for Hadoop. The system super user for Hadoop is simply the operating system user that starts the NameNode. The HDFS super user doesn't have to be the root user of the NameNode host. If you wish, you can allocate a set of users to a separate super user group.

You can make a set of users members of a super user group by setting the `dfs.permissions.superusergroup` configuration parameter in the `hdfs-site.xml` file, as shown here.

```
<property>
  <name>dfs.permissions.superusergroup</name>
  <value>superusergroup</value>
</property>
```

In this example, `superusergroup` is the name of the group of super users in the cluster. The following example shows that the user `hdfs` belongs to the group `superusergroup`:

```
# hdfs dfs -ls /
Found 7 items
drwxr-xr-x - hdfs hdfs 0 2014-06-25 16:39 /data
drwxr-xr-x - hdfs supergroup 0 2015-05-05 15:46 /system
drwxrwxrwt - hdfs hdfs 0 2015-05-09 09:33 /tmp
drwxr-xr-x - hdfs supergroup 0 2015-05-05 13:20 /user
...
#
```

A lot of the administrative HDFS commands need to be run as the “`hdfs`” OS user, which is the default HDFS super user. If you run these commands as any other user, including the root user in a Linux system, you'll get the following error:

```
Access denied for user root. Superuser privilege is required.
```

The root user in Linux is indeed a super user but only for the local file system. It's user `hdfs` who's king when it comes to the HDFS file system. You can perform administration-related HDFS commands only as the `hdfs` user or by `sudo`ing to that user. You can use the Linux `sudo` command to use the privileged administrative commands, as shown in the following example.

```
$ sudo -u hdfs hdfs dfs -rm /user/test/test.txt
```

In this example, the OS user was granted `sudo` privileges to the HDFS account and thus is able to run HDFS file commands as the HDFS super user `hdfs`.

Managing HDFS Storage

You deal with very large amounts of data in a Hadoop cluster, often ranging over multiple petabytes. However, your cluster is also going to use a lot of that space, sometimes with several terabytes of data arriving daily. This section shows you how to check for used and free space in your cluster, and manage HDFS space quotas. The following section shows how to balance HDFS data across the cluster.

The following subsections show how to

- Check HDFS disk usage (used and free space)
- Allocate HDFS space quotas

Checking HDFS Disk Usage

Throughout this book, I show how to use various HDFS commands in their appropriate contexts. Here, let's review some HDFS space and file related commands. You can view the `help` facility for any individual HDFS file command by issuing the following command first:

```
$ hdfs dfs -usage
```

Let's review some of the most useful file system commands that let you check the HDFS usage in your cluster. The following sections explain how to

- Use the `df` command to check free space in HDFS
- Use the `du` command to check space usage
- Use the `dfsadmin` command to check free and used space

Finding Free Space with the `df` Command

You can check the free space in an HDFS directory with a couple of commands. The `-df` command shows the configured capacity, available free space and used space of a file system in HDFS.

```
# hdfs dfs -df
Filesystem                Size      Used    Available  Use%
hdfs://hadoop01-ns  2068027170816000  1591361508626924  476665662189076   77%
#
```

You can specify the `-h` option with the `df` command for more readable and concise output:

```
# hdfs dfs -df -h
Filesystem                Size      Used    Available  Use%
hdfs://hadoop01-ns  1.8 P    1.4 P    433.5 T    77%
#
```

The `df -h` command shows that this cluster's currently configured HDFS storage is 1.8PB, of which 1.4PB have been used so far.

Finding the Used Space with the `du` Command

You can view the size of the files and directories in a specific directory with the `du` command. The command will show you the space (in bytes) used by the files that match

the file pattern you specify. If it's a file, you'll get the length of the file. The usage of the `du` command is as follows:

```
$ hdfs dfs -du URI
```

Here's an example:

```
$ hdfs dfs -du /user/alapati
67545099068 67545099068 /user/alapati/.Trash
212190509 328843053 /user/alapati/.staging
26159 78477 /user/alapati/catalyst
3291761247 6275115145 /user/alapati/hive
$
```

You can view the used storage in the entire HDFS file system with the following command:

```
$ hdfs dfs -du /
414032717599186 883032417554123 /data
0 0 /home
0 0 /lost+found
111738 335214 /schema
1829104769791 5401313868645 /tmp
325747953341360 690430023788615 /user
$
```

The following command uses the `-h` option to get more readable output:

```
$ hdfs dfs -du -h /
353.4 T 733.6 T /data
0 0 /home
0 0 /lost+found
109.1 K 327.4 K /schema
2.1 T 6.1 T /tmp
277.3 T 570.9 T /user
$
```

Note the following about the output of the `du -h` command shown here:

- The first column shows the actual size (raw size) of the files that users have placed in the various HDFS directories.
- The second column shows the actual space consumed by those files in HDFS.

The values shown in the second column are much higher than the values shown in the first column. Why? The reason is that the second column's value is derived by multiplying the size of each file in a directory by its replication factor, to arrive at the actual space occupied by that file.

As you can see, directories such as `/schema` and `/tmp` reveal that the replication factor for all files in these two directories is three. However, not all files in the `/data` and the `/user` directories are being replicated three times. If they were, the second column's value for these two file systems would also be three times the value of its first column.

If you sum up the sizes in the second column of the `dfs -du` command, you'll find that it's identical to that shown by the `Used` column of the `dfs -df` command, as shown here:

```
$ hdfs dfs -df -h /
Filesystem      Size      Used Available Use%
hdfs://hadoop01-ns 553.8 T  409.3 T   143.1 T   74%
$
```

Getting a Summary of Used Space with the `du -s` Command

The `du -s` command lets you summarize the used space in all files instead of giving individual file sizes as the `du` command does.

```
$ hdfs dfs -du -s -h /
131.0 T  391.1 T /
$
```

How to Check Whether Hadoop Can Use More Storage Space

If you're under severe space pressure and you can't add additional `DataNodes` right away, you can see if there's additional space left on the local file system that you can commandeer for HDFS use immediately. In Chapter 3, I showed how to configure the HDFS storage directories by specifying multiple disks or volumes with the `dfs.data.dir` configuration parameter in the `hdfs-site.xml` file. Here's an example:

```
<property>
<name>dfs.data.dir</name>
<value>/u01/hadoop/data, /u02/hadoop/data, /u03/hadoop/data</value>
</property>
```

There's another configuration parameter you can specify in the same file, named `dfs.datanode.du.reserved`, which determines how much space Hadoop can use from each disk you list as a value for the `dfs.data.dir` parameter. The `dfs.datanode.du.reserved` parameter specifies the space reserved for non-HDFS use per `DataNode`. Hadoop can use all data in a disk above this limit, leaving the rest for non-HDFS uses. Here's how you set the `dfs.datanode.du.reserved` configuration property:

```
<property>
<name>dfs.datanode.du.reserved</name>
<value>10737418240</value>
<description>Reserved space in bytes per volume. Always leave this much space
free for non-dfs use.
</description>
</property>
```

In this example, the `dfs.datanode.du.reserved` parameter is set to 10GB (the value is specified in bytes). HDFS will keep storing data in the data directories you assigned to it with the `dfs.data.dir` parameter, until the Linux file system reaches a free space of 10GB on a node. By default, this parameter is set to 10GB. You may consider lowering the value for the `dfs.datanode.du.reserved` parameter if you think there's plenty of unused space lying around on the local file system on the disks configured for Hadoop's use.

Storage Statistics from the `dfsadmin` Command

You've seen how you can get storage statistics for the entire cluster, as well as for each individual node, by running the `dfsadmin -report` command. The `Used`, `Available` and `Use%` statistics from the `dfs -du` command match the disk storage statistics from the `dfsadmin -report` command, as shown here:

```
bash-3.2$ hdfs dfs -df -h /
Filesystem      Size  Used  Available  Use%
hdfs://hadoop01-ns  1.8 P  1.5 P    269.6 T   85%
```

In the following example, the top portion of the output generated by the `dfsadmin -report` command shows the cluster's storage capacity:

```
bash-3.2$ hdfs dfsadmin -report
Configured Capacity: 2068027170816000 (1.84 PB)
Present Capacity: 2067978866301041 (1.84 PB)
DFS Remaining: 296412818768806 (269.59 TB)
DFS Used: 1771566047532235 (1.57 PB)
DFS Used%: 85.67%
...
```

You can see that both the `dfs -du` command and the `dfsadmin -report` command show identical information regarding the used and available HDFS space.

Testing for Files

You can check whether a certain HDFS file path exists and whether that path is a directory or a file with the `test` command:

```
$ hdfs dfs -test -e /users/alapati/test
```

This command uses the `-e` option to check whether the specified path exists.

You can create a file of zero length with the `touchz` command, which is identical to the Linux `touch` command:

```
$ hdfs dfs -touchz /user/alapati/test3.txt
```

Allocating HDFS Space Quotas

You can configure quotas on HDFS directories, thus allowing you to limit how much HDFS space users or applications can consume. HDFS space allocations don't have a direct connection to the space allocations on the underlying Linux file system. Hadoop lets you actually set two types of quotas:

- **Space quotas:** Allow you to set a ceiling on the amount of space used for an individual directory
- **Name quotas:** Let you specify the maximum number of file and directory names in the tree rooted at a directory

The following sections cover

- Setting name quotas
- Setting space quotas

- Checking name and space quotas
- Clearing name and space quotas

Setting Name Quotas

You can set a limit on the number of files and directory names in any directory by specifying a **name quota**. If the user tries to create files or directories that go beyond the specified numerical quota, the file/directory creation will fail. Use the `dfsadmin` command `-setQuota` to set the HDFS name quota for a directory. Here's the syntax for this command:

```
$ hdfs dfsadmin -setQuota <max_number> <directory>
```

For example, you can set the maximum number of files that can be used by a user under a specific directory by doing this:

```
$ hdfs dfsadmin -setQuota 100000 /user/alapati
```

This command sets a limit on the number of files user `alapati` can create under that user's home directory, which is `/user/alapati`. If you grant user `alapati` privileges on other directories, of course, the user can create files in those directories, and those files won't count against the name quota you set on the user's home directory. In other words, name quotas (and space quotas) aren't **user specific**—rather, they are **directory specific**.

Warning

If you create a user's home directory but fail to grant the user a space quota, the user has unlimited storage in HDFS. Not good!

Setting Space Quotas on HDFS Directories

A **space quota** lets you set a limit on the storage assigned to a specific directory under HDFS. This quota is the number of bytes that can be used by all files in a directory. Once the directory uses up its assigned space quota, users and applications can't create files in the directory.

Note

HDFS space quotas are based on limits on HDFS storage that can be used by a directory—and not by a user.

A space quota sets a hard limit on the amount of disk space that can be consumed by all files within an HDFS directory tree. You can restrict a user's space consumption by setting limits on the user's home directory or other directories that the user shares with other users. If you don't set a space quota on a directory it means that the disk space quota is unlimited for that directory—it can potentially use the entire HDFS.

Hadoop checks disk space quotas recursively, starting at a given directory and traversing up to the root. The quota on any directory is the minimum of the following:

- Directory space quota
- Parent space quota

- Grandparent space quota
- Root space quota

Managing HDFS Space Quotas

It's important to understand that in HDFS, there must be enough quota space to accommodate an entire block. If the user has, let's say, 200MB free in their allocated quota, they can't create a new file, regardless of the file size, if the HDFS block size happens to be 256MB. You can set the HDFS space quota for a user by executing the `setSpaceQuota` command. Here's the syntax:

```
$ hdfs dfsadmin -setSpaceQuota <N> <dirname>...<dirname>
```

The space quota you set acts as the ceiling on the total size of all files in a directory. You can set the space quota in bytes (b), megabytes (m), gigabytes (g), terabytes (t) and even petabytes (by specifying p—yes, this is big data!). And here's an example that shows how to set a user's space quota to 60GB:

```
$ hdfs dfsadmin -setSpaceQuota 60G /user/alapati
```

You can set quotas on multiple directories at a time, as shown here:

```
$ hdfs dfsadmin -setSpaceQuota 10g /user/alapati /test/alapati
```

This command sets a quota of 10GB on two directories—`/user/alapati` and `/test/alapati`. Both the directories must already exist. If they do not, you can create them with the `dfs -mkdir` command.

Caution

The space quota includes all replicated data. If you set the quota at 30GB for a user, that user can exhaust her quota by storing 10GB of actual data in her HDFS directory (using the default replication factor of three, HDFS stores $10 \times 3 = 30$ GB of data).

You use the same command, `-setSpaceQuota`, both for setting the initial limits and modifying them later on. When you create an HDFS directory, by default, it has no space quota until you formally set one.

You can remove the space quota for any directory by issuing the `-clrSpaceQuota` command, as shown here:

```
$ dfsadmin -clrSpaceQuota /user/alapati
```

If you remove the space quota for a user's directory, that user can, theoretically speaking, use up all the space you have in HDFS. As with the `-setSpaceQuota` command, you can specify multiple directories in the `-clrSpaceQuota` command.

Things to Remember about Hadoop Space Quotas

Both the Hadoop block size you choose and the replication factor in force are key determinants of how a user's space quota works. Let's suppose that you grant a new user a space quota of 30GB and the user has more than 500MB still free. If the user

tries to load a 500MB file into one of his directories, the attempt will fail with an error similar to the following, even though the directory had a bit over 500MB of free space.

```
org.apache.hadoop.hdfs.protocol.DSQuotaExceededException: The DiskSpace quota
of /user/alapati is exceeded: quota = 32212254720 B = 30 GB but
diskspace consumed = 32697410316 B = 30.45 GB
```

In this case, the user had enough free space to load a 500MB file but still received the error indicating that the file system quota for the user was exceeded. This is so because the HDFS block size was 128MB, and so the file needed 4 blocks in this case. Hadoop tried to replicate the file three times since the default replication factor was three and so was looking for $128 \times 12 = 1556$ MB of space, which clearly was over the space quota left for this user.

Note

The disk space quota is deducted based not only on the size of the file you want to store in HDFS but also the number of replicas. If you've configured a replication factor of three and the file is 500MB in size, three block replicas are needed, and therefore, the total quota consumed by the file will be 1,500MB, not 500MB.

The administrator can reduce the space quota for a directory to a level below the combined disk space usage under a directory tree. In this case, the directory is left in an indefinite **quota violation state** until the administrator or the user removes some files from the directory. The user can continue to use the files in the overfull directory but, of course, can't store any new files there since their quota is violated.

Checking Current Space Quotas

You can check the size of a user's HDFS space quota by using the `dfs -count -q` command as shown in Figure 9.7.

When you issue a `dfs -count -q` command, you'll see eight different columns in the output. This is what each of the columns stands for:

- `QUOTA`: Limit on the files and directories
- `REMAINING_QUOTA`: Remaining number of files and directories in the quota that can be created by this user
- `SPACE_QUOTA`: Space quota granted to this user
- `REMAINING_SPACE_QUOTA`: Space quota remaining for this user
- `DIR_COUNT`: The number of directories
- `FILE_COUNT`: The number of files
- `CONTENT_SIZE`: The file sizes
- `PATH_NAME`: The path for the directories

The `-count -q` command shows that the space quota for user `bdaldr` is about 100TB. Of this, the user has about 67TB left as free space.


```

bash-3.2$ hdfs dfs -count -q /user/bdald
      none          inf 109951162777600  67751254421557      15870      283346
14058618193064 /user/bdaldr
bash-3.2$ █

```

Figure 9.7 How to check a user's current space usage in HDFS against their assigned storage limits

Clearing Current Space Quotas

You can clear the current space quota for a user by issuing the `clrSpaceQuota` command as shown here:

```
$ hdfs dfsadmin -clrSpaceQuota
```

Here's an example showing how to clear the space quota for a user:

```

$ hdfs dfsadmin -clrSpaceQuota /user/alapati
$ hdfs dfs -count -q /user/alapati
      none          inf          none          inf          2
0
0 /user/alapati
$

```

The user still can use HDFS to read files but won't be able to create any files in that user's HDFS "home" directory. If the user has sufficient privileges, however, she can create files in other HDFS directories. It's a good practice to set HDFS quotas on a per-user basis. You must also set quotas for data directories on a per-project basis.

Rebalancing HDFS Data

Over time, the data in the HDFS storage can become skewed, in the sense that some of the DataNodes may have more data blocks compared to the rest of the cluster's nodes. In cases of extreme skew, the read and write activity is overly busy on the nodes with more data, and the sparsely populated nodes remain underutilized.

HDFS data also gets unbalanced when you add new nodes to your cluster. Hadoop doesn't automatically move existing data around to even out the data distribution among a cluster's DataNodes. It simply starts using the new DataNode for storing fresh data.

Note

It's a good practice to run the HDFS balancer regularly in a cluster.

Hadoop doesn't seek to achieve a fully balanced cluster. This state of affairs is quite hard to achieve in a cluster with continuous data flows. Instead, Hadoop is satisfied when the space usage on each DataNode is within a certain percentage of space used by the other DataNodes. In addition, it also makes use of a threshold size to give you flexibility with the balancing of data.

Hadoop makes available a useful tool, called the **balancer**, to let you rebalance a cluster's block distribution so all DataNodes store roughly equal amounts of data.

The following sections cover

- Reasons for an unbalanced HDFS
- Using Hadoop’s balancer tool
- Setting the proper threshold value
- When to run the balancer
- Making the balancer run faster

Reasons for HDFS Data Imbalance

There’s no guarantee that HDFS will automatically distribute data evenly among the DataNodes in a cluster. For example, when you add a new node to the cluster, all new blocks could be allocated to that node, thus making the data distribution lopsided. When the NameNode allocates data blocks to the nodes, it considers the following criteria to determine which DataNodes get the new blocks.

- Uniformly distributing data across the cluster’s DataNodes
- Keeping one of the replicas of a data block on the node that’s writing the block
- Placing one of the replicas on the same rack as the node writing the block, to minimize cross-rack network I/O
- Spreading the block replicas across racks to support redundancy and survive the loss of an entire rack

Hadoop considers a cluster balanced when the percentage of space in a given DataNode is a little bit above or below the average percentage of space used by the DataNodes in that cluster. What this “little bit” is, is defined by the parameter `threshold size`.

Running the Balancer Tool to Balance HDFS Data

The aforementioned HDFS **balancer** is a tool provided by Hadoop to balance the data spread across the DataNodes in a cluster by moving data blocks from the over-utilized to the under-utilized DataNodes. Figure 9.8 shows the idea behind the balancer tool. Initially Rack 1 and Rack 2 have data blocks. The new rack, Rack 3, has no data initially—only newly added data will be placed there. This means adding nodes leads to an unbalanced cluster. Data is moved from the nodes with data to the new nodes, which have no data until you move data over to them from the current DataNodes or wait for new data to come in. When you run the balancer, Hadoop moves data blocks from their existing locations to the nodes that have more free space, all nodes will have roughly the same amount of used space.

You can run the balancer manually from the command line by invoking the balancer command. The `start-balancer.sh` command invokes the balancer. You can also

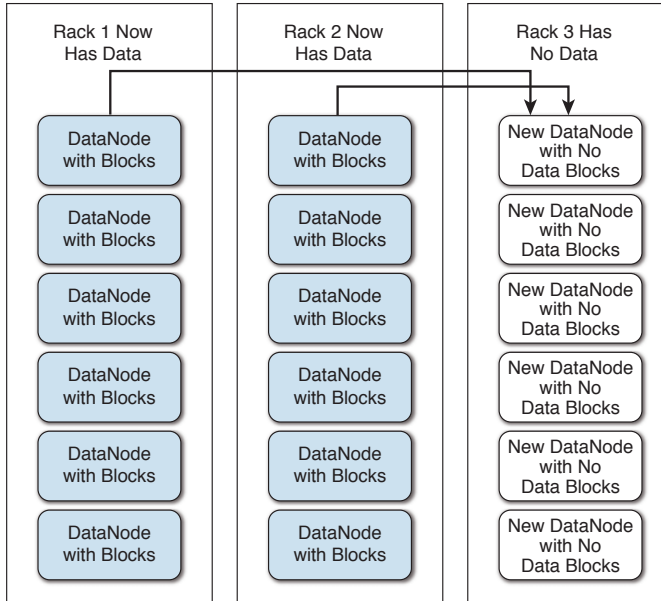


Figure 9.8 How the balancer moves data blocks to the under-utilized nodes from the over-utilized nodes

run it by issuing the command `hdfs -balancer`. Here's the usage of the balancer command:

```
$ hdfs balancer --help
Usage: java Balancer
      [-policy <policy>]      the balancing policy: datanode or blockpool
      [-threshold <threshold>] Percentage of disk capacity
      [-exclude [-f <hosts-file> | comma-separated list of hosts]] Excludes
the specified datanodes.
      [-include [-f <hosts-file> | comma-separated list of hosts]] Includes
only the specified datanodes.
```

The `threshold` parameter denotes the percentage deviation of HDFS usage of each DataNode from the cluster's average DFS utilization ratio. Exceeding this threshold in either way (higher or lower) would mean that the node will be rebalanced.

The default DataNode policy is to balance storage at the DataNode level. The balancer doesn't balance data among individual volumes of the DataNode, however. The alternative **blockpool policy** applies only to a federated HDFS service.

Setting the Proper Threshold Value for the Balancer

You can run the balancer command without any parameters, as shown here:

```
$ sudo -u hdfs hdfs balancer
```

This `balancer` command uses the default threshold of 10 percent. This means that the balancer will balance data by moving blocks from over-utilized to under-utilized nodes, until each DataNode's disk usage differs by no more than plus or minus 10 percent of the average disk usage in the cluster.

Sometimes, you may wish to set the threshold to a different level—for example, when free space in the cluster is getting low and you want to keep the used storage levels on the individual DataNodes within a smaller range than the default of plus or minus 10 percent. You can do so by specifying the threshold parameter, as shown here:

```
$ hdfs balancer -threshold 5
```

Tip

How long the balancer will run depends on the size of the cluster and how unbalanced the data is. When you run the balancer for the very first time, or you schedule it infrequently, as well as when you run it after adding a set of DataNodes, it will run for a very long time—often several days.

The amount of data moved around during rebalancing depends on the value of the threshold parameter. If you use the default value of 10 and the average DFS usage across the cluster is, for example, 70 percent, the balancer will ensure that each DataNode's DFS usage lies somewhere between 60 and 80 percent of that DataNode's storage capacity, once the balancing of the HDFS data is completed.

When you run the balancer, it looks at two key HDFS usage values in your cluster:

- **Average DFS used percentage:** The average DFS used percentage in the cluster can be derived by performing the following computation:

$$\text{Average DFS Used} = \text{DFS Used} * 100 / \text{Present Capacity}$$
- **A Node's used DFS percentage:** This measure shows the percentage of DFS used per node.

The balancer will balance a DataNode only if the difference between a DataNode's used DFS percentage and the average DFS used (by the cluster) is greater than the threshold value. Otherwise, it won't rebalance the cluster.

As noted previously, if you run the balancer without specifying a threshold value, it'll use the default value of 10 as the threshold. In our case, it won't perform any balancing, ending up as shown here (assuming all the DataNodes have a similar DFS usage as that of Node10):

```
$ hdfs balancer
15/05/04 12:56:36 INFO balancer.Balancer: namenodes = [hdfs://hadoop01-ns]
15/05/04 12:56:36 INFO balancer.Balancer: parameters = Balancer
.Parameters[BalancingPolicy.Node, threshold=10.0, number of nodes to be excluded = 0,
number of nodes to be included = 0]
Time Stamp           Iteration#   Bytes Already Moved   Bytes Left To Move
Bytes Being Moved
'''
```

```
The cluster is balanced. Exiting...
May 4, 2015 12:56:37 PM Balancing took 1.47 seconds
$
```

The balancer ran, but it wound things up pretty quickly, because it found that all nodes in the cluster have a usage that's within the threshold value—the cluster is already balanced!

In our case, for balancing to occur, you must specify a threshold value that's ≤ 2 . Here's one way to run it:

```
$ nohup su hdfs -c "hdfs balancer -threshold 2" > /tmp/balancer.log/stdout.log
2>/tmp/balancer.log/stderr.log &
```

Specifying `nohup` and `&` will run the job in the background and get back control of the shell. Since a balancer job can run for quite a long time in a cluster, it's a good idea to run it in this way.

Using `hdfs dfsadmin` to Make Things Easier

In our example, we used a single node, Node10, to check that node's DFS used percentage. We then figured out that we must set the threshold to a value that is ≤ 2 based on this node's DFS used percentage. But you can't run the balancer on a specific node. So, how do you determine the threshold value when you have a larger number of nodes? It's easy. Just pick the lowest DFS used percentage of a node in the entire cluster. You don't have to spend a lot of time figuring out the DFS used percentages for each node. Use the `hdfs dfsadmin -report` command to find out everything you need in order to figure out the right threshold value.

In this example, there are 50 nodes in the cluster. I can run the `dfsadmin` command as follows, capturing the output in a file, since the command will print out the DFS usage reports for each node separately.

```
[root@hadoop01]# sudo -u hdfs hdfs dfsadmin -report > /tmp/dfsadmin.out
```

Look at the very top of the command's output (in the file `dfsadmin.out`), where you'll find the DFS used statistics for the entire cluster:

```
Configured Capacity: 608922615386112 (553.81 TB)
Present Capacity: 607364914327552 (552.40 TB)
DFS Remaining: 166697481228288 (151.61 TB)
DFS Used: 440667433099264 (400.78 TB)
```

```
DFS Used%: 72.55%
```

The smaller the value of the threshold parameter, the more work the balancer will need to perform and the more balanced the cluster will be. However, there's a catch here: If you have a heavily used cluster with numerous writes and deletes of data, the cluster may never reach a fully balanced state, and the balancer will be merely moving around data from one node to another.

When you start the balancer, you'll see the following type of output. Note how the balancer determines how many nodes are overutilized or underutilized. It'll move data from the overutilized nodes to the rest of the cluster nodes. It also determines

the actual amount of data that needs to be moved around to balance the cluster's data distribution.

```
30/05/2016 10:02:26 INFO balancer.Balancer: 4 over-utilized: #A
[10.192.0.55:50010:DISK, 10.192.0.24:50010:DISK, 10.192.0.54:50010:DISK,
10.192.0.25:50010:DISK]
30/05/2016 10:02:26 INFO balancer.Balancer: Need to move 8.05 TB to make the
cluster balanced. #A
30/05/2016 09:07:21 INFO Balancer: Decided to move 10 GB bytes from #B
10.192.0.55:50010:DISK to 10.192.0.116:50010:DISK
30/05/2016 09:07:21 INFO balancer.Balancer: Decided to move 10 GB bytes from
10.192.0.25:50010:DISK to 10.192.0.115:50010:DISK
30/05/2016 09:07:21 INFO balancer.Balancer: Decided to move 10 GB bytes from
10.192.0.24:50010:DISK to 10.192.0.118:50010:DISK
30/05/2016 09:07:21 INFO balancer.Balancer: Decided to move 10 GB bytes from
10.192.0.54:50010:DISK to 10.192.0.110:50010:DISK
30/05/2016 09:07:21 INFO balancer.Balancer: Will move 40 GB in this iteration
30/05/2016 09:07:22 INFO balancer.Dispatcher: Successfully moved
blk_1155910122_1099683676641 with size=17370340 from 10.192.0.54:50010:DISK to
10.192.0.110:50010:DISK through 10.192.0.54:50010 #B

May 30, 2016 10:34:10 PM Balancing took 14.561533333333334 minutes #C
$
```

Notes

- #A Points out the four DataNodes that are currently overutilized. Their HDFS usage percentage is higher than the average HDFS usage for the cluster.
- #B Shows how the balancer moves the data from overutilized to underutilized DataNodes.
- #C Shows the completion of the balancing once the data is evenly spread across all DataNodes.

Tip

To keep the balancer from running for a very long time, specify a higher threshold first and then drop the threshold to a lower value the next time you run the balancer.

Iterative Movement of Blocks

The goal of the balancer is to move data from the overutilized nodes to the underutilized nodes, thus balancing the DFS usage across the cluster. When you start the balancer, it starts by moving some data from nodes whose DFS usage is higher than the threshold and moves that data to nodes whose DFS usage is below the threshold. The balancer is rack aware and thus will generate minimal inter-rack traffic. The balancer works in an iterative fashion, moving a certain amount of data per iteration as the output of the balancer run shows (e.g., "Will move 40GB in this iteration").

When to Run the Balancer

A couple of guidelines as to when to run the balancer are appropriate. In a large cluster, run the balancer regularly. You can schedule a cron job to perform the balancing, instead of manually running it yourself. If a scheduled balancer job is still running when the next job needs to start, no harm's done, as the second balancer job won't start.

It's a good idea to run the balancer right after adding new nodes to the cluster. When you add a large number of nodes at once and run the balancer afterwards, it'll take quite a while to complete its work.

Making the Balancer Run Faster

Ideally you must run the balancer during periods when the cluster is being lightly utilized, but the overhead is usually not high. You can adjust the bandwidth of the balancer to determine the number of bytes per second that each DataNode in the cluster can use to rebalance its data.

The default value for the bandwidth is 10MB per second and you can raise it to make the balancer complete its work faster. You can raise the bandwidth up to about 10 percent of your network speed without any noticeable impact on the cluster's workload. You can set the network bandwidth used by the balancer with the help of the `hdfs dfsadmin` command, as shown here:

```
$ hdfs dfsadmin -setBalancerBandwidth <bandwidth in bytes per second>
```

The `-setBalancerBandwidth` option enables you to change the network bandwidth consumed by each DataNode in your cluster during an HDFS block balancing operation. The bandwidth you specify here is the maximum number of bytes per second that will be used by each DataNode in the cluster. If you're using a shell script to invoke the balancer periodically, you can specify the bandwidth option in the script before invoking the balancer. Here's an example showing how to change the bandwidth to 20MB.

```
$ hdfs dfsadmin -setBalancerBandwidth 20971520
Balancer bandwidth is set to 20971520 for hadoop01.localhost/10.192.0.22:8020
Balancer bandwidth is set to 20971520 for hadoop01.localhost/10.192.0.51:8020
$
```

Make sure that you have adequate bandwidth before increasing the bandwidth. You can find out the speed of your NIC card by issuing the following command:

```
$ ethtool eth0
...
Speed: 1000Mb/s
Duplex: Full
...
$
```

In this example, the network has a speed of 1,000MB per second, so it's safe to set the balancer bandwidth to about 10 percent of it, which is 100MB per second.

When the balancer runs for a long time, you can schedule it to run with different bandwidths during peak and off peak times. You can run it with a low bandwidth during peak times and run it with a higher bandwidth during periods when the cluster is less busy. For example, during peak times, you can schedule a cron job such as the following for the balancer (bandwidth of 10MB):

```
$ su hdfs -c 'hdfs dfsadmin -setBalancerBandwidth 10485760'
$ nohup su hdfs -c 'hdfs balancer' > /tmp/balancerstderr.log 2>
/tmp/balancerstdout.log &
```

You can at the same time schedule a different cronjob to run at off-peak times, with a higher (20MB) setting for the bandwidth parameter:

```
$ su hdfs -c 'hdfs dfsadmin setBalancerBandwidth 20971520>'
$ nohup su hdfs -c 'hdfs balancer' > /tmp/balancerstderr.log 2>
/tmp/balancerstdout.log &
```

Only one balancer job can run at a time. When the second (off-peak) job starts, it stops the first balancer job and starts a new balancer job with the higher bandwidth setting.

Reclaiming HDFS Space

Oftentimes you can conserve HDFS storage space by reclaiming used space where you can. There are two ways in which you can reclaim space allocated to HDFS files:

- You can remove the files or directories once you're done processing them.
- You can reduce the replication factor for a file.

Removing files works well with the raw data files you load into HDFS for processing, and the reduction of the replication factor is a good strategy for handling older and less-critical HDFS files.

Removing Files and Directories

Periodic removal of unnecessary data is an operational best practice. Often, data needs to be retained only for a specific period of time. You can stretch your storage resources by removing any files that are just sitting in HDFS and eating up valuable space.

Decreasing the Replication Factor

You can configure the replication factor at the cluster level by setting the `dfs.replication` parameter in the `hdfs-site.xml` file, as explained in Chapter 4, “Planning for and Creating a Fully Distributed Cluster.” The setting you configure with the `dfs.replication` parameter sets a global replication factor for the entire cluster.

It's important to understand that while you can set the replication factor at the cluster level, you can modify the replication factor for any existing file, with the `-setRep` command. This offers great flexibility, as you can set the replication factor based on

the importance and usage of data. For example, you can lower the replication factor for historical data and raise the replication factor for “hot” data, so more nodes can process the data.

You can change the global replication factor anytime by configuring the `dfs.replication` parameter. Hadoop will either add or remove replicas across the cluster based on whether you increase or decrease the global replication factor.

Note how this behavior is different from how the `fs.block.size` parameter works. The `fs.block.size` parameter sets the block size for the cluster. When you change the value of this parameter, it won't change the block size of files already in HDFS. It'll use the new block size only for new files that are stored in HDFS.

Applications can also specify the replication factor on a per-file basis. You can change the replication factor for a file anytime with the `hdfs dfs -setRep` option. You can change the replication factor for a single file with this command:

```
$ hdfs dfs -setRep -w 2 /data/test/test.txt
```

You can change the replication factor for all files in a directory by adding the `-R` option as shown here:

```
$ hdfs dfs -setRep -w 2 -R /data/test
```

You can reduce the amount of HDFS space occupied by a file by simply reducing the file's replication factor. When you reduce the replication factor using the `hdfs dfs -setrep` option, the NameNode sends the information about the excess replicas to the DataNodes, which will remove the corresponding blocks from HDFS.

Here's an example showing how to reduce the replication factor from the default level of 3 to 2:

1. Issue the following command to check the current replication factor for the file.

```
$ hdfs dfs -ls /user/hive/warehouse/customer/year=2016/month=12/day=31
-rw-r--r--  3 alapati  analysts  60226324 2016-02-01 01:07
/user/hive/warehouse/customer/year=2015/month=01/day=31/
CustRecord-20150131_040_28049_20150131235718-000001-0.avro
```

The number 3 next to the file permission list indicates the replication factor for this file.

2. Change the replication factor from 3 to 2 with the following command:

```
$ hdfs dfs -setrep -R -w 2
/user/hive/warehouse/customer/year=2015/month=12
```

You can check to make sure that the replication factor has been changed to 2 from 3.

```
$ hdfs dfs -ls /user/hive/warehouse/shoprecord/year=2016/month=01/day=31
-rw-r--r--  2 alapati  analysts  60226324 2016-02-01 01:07
/user/hive/warehouse/customer/year=2015/month=01/day=31/CustRecord-
20160131_040_28049_20160131235718-000001-0.avro
```

3. Optionally, you can also add the `-w` flag with this command, to wait for the replication to complete, but this takes a long time for some files. You can see that the replication factor has changed to 2 for the file.

```
$ hdfs dfs -ls
/user/hive/warehouse/customer/year=2016/month=01/day=31
-rw-r--r--  2 alapati analysts  60226324 2015-02-01 01:07
/user/hive/warehouse/customer/year=2015/month=01/day=31/
ShoppingRecord-20160131_040_28049_20160131235718-000001-0.avro
```

In the example here, I changed the replication factor for a file. If you specify a directory instead of a file, the `setrep` command will recursively change the replication factor for all files that are under the directory name you specify.

Although I discussed reducing the replication factor as a way to conserve storage, for important data, you can also try increasing the replication factor. You can also set a higher replication factor for data that's in demand (hot data).

Summary

Here's what you learned in this chapter:

- The `hdfs dfs` command is your ally in performing day-to-day work with HDFS files and directories
- The `hdfs dfsadmin` command is highly useful for checking the status of the DataNodes and the way HDFS data is spread across the DataNodes
- By granting space and file quotas, you can control HDFS usage.
- The `hdfs du` and `hdfs df` commands are handy for finding out how your cluster is using its storage
- Balancing your cluster's data on a regular basis provides computational benefits by evenly spreading HDFS data across all the nodes of your Hadoop cluster.

Index

- | (pipe symbol)
 - piping data into HDFS files, 360
 - reviewing files, 359
- (minus sign), in dfs subcommands, 245
- * (asterisk), wildcard when copying multiple files, 358
- 256-byte encryption, enabling/disabling, 490
- A**
- Acceptance filters, 496
- “Access denied for user root” message, 259
- Accounting. *See* Auditing.
- Accumulators, 702–703
- acl_file parameter, 490
- ACLs (access control lists). *See also*
 - Authorization.
 - authorization, 507–509
 - blocking, 512
 - configuring service level authorization, 510–511
 - permissions, 257, 507–509
 - specifying for UPNs, 490
- Action nodes, Oozie workflows
 - configuring, 449–454
 - description, 438, 448
 - fs actions, 454
 - for Hive jobs, 451–452
 - for MapReduce jobs, 450–451
 - for Pig, 452–453
 - Shell actions, 453
 - types of, 449–450. *See also specific types.*
- Actions. *See also* RDD (resilient distributed dataset), actions.
 - Sentry authorization, 513
 - Spark programming, 170
- Active NameNode
 - checkpointing with a secondary NameNode, 328
 - HA (high availability), 335, 336–337, 345–346
 - monitoring with ZKFC, 335
- Active NameNode failover
 - fencing mechanism, 340–341
 - role of the JournalNodes, 336
- AD (Active Directory)
 - integrating with Hadoop, 504–505
 - Kerberized clusters, setting up one-way trust, 503–504
- addDirective attribute, 230
- addPool attribute, 230
- addprinc command, 492–493
- admin command, 472
- Administration, key areas of
 - allocating cluster resources, 28–29
 - authentication, 30
 - authorization, 30
 - Capacity Scheduler, 29
 - cronning jobs, 29–30, 474
 - default security, 30
 - DRF (Dominant Resource Fairness), 28–29
 - FairScheduler, 28–29
 - Kerberos, 30
 - Knox, 31
 - managing cluster capacity, 28–29
 - managing cluster storage, 28
 - Oozie workflows, 29–30
 - Ranger, 31
 - scheduling jobs, 29–30
 - securing data, 30–31
 - Sentry, 30
- Administrative protocols, 511
- Administrators
 - skills required, 20–21
 - toolset, 21
- Administrators, duties of
 - assisting developers, 19
 - backups, 20
 - disaster recovery, 20
 - installation and upgrades, 19
 - overview, 18–19
 - performance tuning and optimization, 20
- Advanced execution engine, Spark, 150–151

- Agent nodes, 389. *See also* Flume agents.
- Agents. *See* Flume agents.
- aggregateByKey operator, 702
- Aggregating data. *See* Log aggregation.
- Alerting and monitoring. *See* Monitoring.
- Alerting tool, 582
- Allocating YARN memory
 - configuring MapReduce memory, 615–617
 - configuring YARN memory, 613–615
 - overview, 612–613
- Allocation files, Fair Scheduler, 428
- allocations.xml file, configuring Fair Scheduler, 75
- allowSnapshot command, 281
- ALL_SSD, storage policies, 237
- All-to-all operations, 695
- Amazon Elastic MapReduce (EMR), 307
- Amazon Simple Storage Service (S3), 165
- Amazon Web Services, Hadoop distribution, 60
- Ambari, 570, 576
- ANY, data locality level, 715
- Apache products. *See specific products.*
- application command, 531
- Application logs
 - creating, 590–592
 - definition, 584
 - HDFS directories, 585
 - log aggregation, 585, 592
 - logging levels, 591–592
 - NodeManager local directories, 585
 - NodeManager log directories, 585–586
 - retention duration, setting, 592
 - storing, 585–586
 - storing in HDFS. *See* Log aggregation.
 - viewing, 584–585, 596–597
- Application preemption, Fair Scheduler, 431–432
- applicationattempt command, 532
- ApplicationMaster
 - allocating resources, 53–56
 - vs.* ApplicationsManager, 51
 - configuring MapReduce memory, 617–618
 - crashes, troubleshooting, 738
 - in Hadoop clusters, 36
 - JobHistoryServer, 54
 - main functions, 52–53
 - memory issues, troubleshooting, 735–736
 - YARN, 52–56
- Applications. *See also* Jobs.
 - in Hadoop clusters, coordinating execution of, 36. *See also* Hive; MapReduce framework; Pig.
 - limiting number of, 420–421
 - moving between queues, Fair Scheduler, 434
 - preempting, 421–422
 - Spark. *See* Spark applications.
 - status, checking, 532
 - YARN. *See* YARN commands for managing applications.
- ApplicationsManager, 51
- apt-get utility, 63
- Architecture
 - fully distributed clusters, 93
 - fully distributed clusters, single rack to multiple racks, 95–96
 - Ganglia, 580
 - Hadoop clusters, 35
 - HDFS transparent encryption, 521
 - YARN, 49–50
- Architecture, Hadoop 2
 - computation and storage, 34–35
 - redundancy of data, 34
- Architecture, HDFS
 - DataNodes, 38–39
 - master nodes, 38–39
 - NameNodes, 38–39
 - worker nodes, 38–39
- Archival disk-bound storage, 236
- Archival storage
 - cold data, 232, 233–234
 - on DataNodes, 240–241
 - on each DataNode, 240–241
 - fallback storage media, 235
 - frozen data, 232, 233–234
 - heterogeneous HDFS storage, 233–234
 - hot data, 232, 233–234
 - implementing, 240–241
 - Mover tool, 240
 - moving data around, 239–240
 - storage architecture, 234–235
 - storage preferences for files, 235

- storage types, performance characteristics, 233
 - warm data, 232, 233–234
- Archival storage, setting up
 - ALL_SSD, storage policies, 237
 - archival disk-bound storage, 236
 - ARCHIVE storage type, 236
 - cold data, storage policies, 237
 - configuring multiple storage tiers, 235–236
 - DISK storage type, 236
 - flash storage, storage policies, 237
 - hot data, storage policies, 237
 - in-memory storage, 237
 - Lazy_Persist, storage policies, 237
 - ONE_SSD, storage policies, 237
 - RAM_DISK storage type, 237
 - SSD storage type, 237
 - standard disk-based storage, 236
 - storage types, 236–239
 - temporary data, storage policies, 237
 - warm data, storage policies, 237
- Archival storage, storage policies
 - architecture, 238
 - listing, 238
 - managing, 239
 - specifying, 239
 - summary of, 237
- ARCHIVE storage type, 236
- argument element, 453
- AS (Authentication Service), Kerberos, 482–483
- Assisting developers, administrator duties, 19
- Asterisk (*), wildcard when copying multiple files, 358
- Auditing
 - definition, 478
 - HDFS operations, 519
 - log files, 519–520
 - overview, 481, 518–519
 - YARN operations, 519
- Authenticating
 - users, Kerberized clusters, 502
 - users and services, 501
- Authentication. *See also* Kerberos; Sentry.
 - administration, 30
 - vs.* authorization, 505
 - default mode, 257
 - definition, 480
 - overview, 480
 - user identity, 480
- Authentication process, Kerberos, 480, 483–484
- Authentication server, Kerberos, 483
- Authorization. *See also* ACLs (access control lists).
 - administration, 30
 - vs.* authentication, 505
 - definition, 481
 - overview, 481
 - permissions, 507
- Authorization, HDFS permissions
 - ACLs (access control lists), 507–509
 - changing file permissions, 507
 - checking permissions, 507
 - configuring, 506
 - configuring super users, 506
 - extended attributes, 509–510
 - overview, 505–506
 - raw namespace, 509–510
 - security namespace, 509–510
 - simple security mode, 505–506
 - system namespace, 509–510
 - user namespace, 509–510
- Authorization, Sentry
 - actions, 513
 - configuring Hive, 516–517
 - configuring the server for Hive, 515–516
 - executing Hive queries, 517
 - groups, 513, 514
 - Hive authorization, 514–516
 - key concepts, 513
 - objects, 513
 - overview, 513
 - policies, 513, 517
 - policy administration examples, 517–518
 - policy engine, 513
 - policy providers, 513
 - privilege models, 514
 - privileges, 513, 514
 - roles, 514, 518–519
 - Sentry policy file, 514
 - Sentry service, 514
 - users, 513

- Authorization, service level authorization
 - ACLs, blocking, 512. *See also* banned.users parameter.
 - administrative protocols, 511
 - client protocols, 511
 - configuring with ACLs, 510–511
 - controlling HDFS administrative access, 511
 - enabling, 510
 - refreshing SLA configurations, 511
 - reporting task progress, 511
 - user whitelist, 511
 - Automated deployment tools, 63
 - Automatic failover, 347–348
 - Average DFS Used Percentage, 270–271
 - Avro description, 17
 - Avro files
 - benefits of, 301–302
 - description, 301–302, 679–680
 - loading data from relational databases to HDFS, 373
 - structured format, 290
 - Avro format, in HDFS, 42
- B**
- Backup and recovery. *See also* Fault tolerance.
 - NameNode operations, safe mode, 332–334
 - work preserving recovery, 739
 - Backup and recovery, backups. *See also* Snapshots; Trash directory.
 - administrator duties, 20
 - fetchimage command, 552–553
 - HDFS metadata, 552–553
 - metastore databases, 553
 - Backup and recovery, recovery. *See also* Fault tolerance.
 - block recovery, 226
 - close stage, 227
 - data streaming stage, 227
 - disaster recovery, 20. *See also* Snapshots.
 - GS (Generation Stamp), 224
 - lease recovery, 224–225
 - pipeline recovery, 226–227
 - pipeline setup stage, 227
 - RUR (Replica Under Recovery) replica state, 216
 - UNDER_RECOVERY block state, 218–219
 - work preserving recovery, 739
 - Backup Node, checkpointing, 324–325
 - balancer command, 269
 - Balancer tool, 267, 268–271
 - Bandwidth, monitoring, 572
 - banned.users parameter, 498. *See also* ACLs (access control lists), blocking.
 - Batch intervals, 195, 689
 - Batch processing time, reducing, 688–689
 - Beeline, 192, 517
 - Beeswax, configuring, 560
 - Benchmarking clusters. *See also* Hadoop metrics.
 - Folder, 643–644
 - generating job traces, 643–644
 - with GridMix, 644–646
 - with HiBench, 642–643
 - overview, 638
 - read tests, 640
 - with Rumens, 643–644
 - scaling trace runtime, 643–644
 - TeraSort, 640–643
 - with TeraSort, 640–643
 - testing I/O performance, 638–640
 - tiny jobs, 646
 - Trace Builder, 643–644
 - uberized jobs, 646
 - write tests, 639
 - Benchmarks, TeraSort, 642
 - BI (business intelligence), 10, 191, 198
 - Bigtop, Hadoop distribution, 60
 - Binary formats
 - description, 298
 - in HDFS, 42
 - loading data from relational databases to HDFS, 373
 - Blade servers on fully distributed clusters, single rack to multiple racks, 97
 - Block access tokens, 501
 - Block locations, printing, 287
 - “Block MISSING” messages, 287–288
 - Block recovery, 226
 - Block replication, setting, 107–108
 - Block reports
 - generating, 287
 - NameNode operations, 322

- Block size
 - client HDFS, determining, 222
 - default, 213
 - setting, 107
 - Block states, data replication, 218–219
 - Block Storage Service, 350
 - blocks option, 287
 - Breaking up large workloads, cluster
 - computing, 12
 - Broadcast variables, 672, 702–703
 - Brokers, Kafka, 400, 402
 - Bucketing Hive jobs, 635
 - Bundle jobs, Oozie workflows, 439
 - Bundles, Oozie, 473
 - Business data, traditional database systems, 7–8
 - Business intelligence (BI), 10, 191, 198
 - Bypassing the trash directory, 280
 - bzip2 format, 290, 291
- C**
- Cache directives, 228, 230–231
 - cache() method, 718
 - Cache pools, 228, 230–231
 - Cache status, displaying, 684
 - cacheadmin command line interface, 229, 230
 - Caching RDD data
 - cache() method, 718
 - checking the cache, 721
 - DISK_ONLY storage level, 719
 - fault tolerance, 718
 - lineage information, 718
 - MEMORY_AND_DISK storage level, 719
 - MEMORY_AND_DISK_SER storage level, 719
 - MEMORY_ONLY storage level, 719
 - MEMORY_ONLY_SER storage level, 719
 - optimizing, 717–723
 - overview, 717–718
 - persist() method, 719–721
 - in serialized format, 722
 - setting storage levels, 720–721, 721–722
 - Caching Spark tables, 723
 - Call center data, definition, 6
 - Capacity and elasticity, tradeoffs, 419
 - Capacity guarantees, Capacity Scheduler, 412, 414
 - Capacity Scheduler. *See also* Fair Scheduler; Oozie.
 - administration, 29
 - capacity guarantee, 412
 - configuring, 75
 - description, 409, 411–412
 - enabling, 422
 - vs.* Fair Scheduler, 435–436
 - fair share preemption, 421–422
 - maximum capacity, 412
 - minimum share preemption, 421–422
 - preempting applications, 421–422
 - Capacity Scheduler, allocating resources
 - capacity and elasticity, tradeoffs, 419
 - number of applications, limiting, 420–421
 - overview, 418
 - user capabilities, limiting, 419–420
 - Capacity Scheduler, examples
 - administering queues, 424–425
 - code sample, 422–424
 - modifying queue configuration, 424
 - resource limits, setting, 423–424
 - Capacity Scheduler, queues
 - administering, 424–425
 - capacity guarantees, 414
 - configuring capacity, 417
 - creating, example, 413–414
 - diagram, 418
 - elasticity, 414
 - hierarchical, 414, 416
 - importance of, 409–410
 - leaf, 414
 - modifying configuration, 424
 - overview, 412
 - queue element, 415
 - resource limits, setting, 423–424
 - setting up, 415–416
 - Capacity Scheduler, subqueues. *See also* Hierarchical queues; Leaf queues.
 - configuring, 414
 - creating, 413–414
 - diagram, 418
 - setting up, 415–416
 - capacity-scheduler.xml file, configuring
 - Capacity Scheduler, 75
 - capture-output element, 453
 - case statements, Oozie workflows, 460

- Cassandra, 150, 152, 171, 200, 391, 400
- cat command, 356–357
- Catalog, description, 17
- cd command, 245
- Centralized cache management
 - cache directives, 228, 230–231
 - cache pools, 228, 230–231
 - configuring caching, 229
 - functional description, 229
 - Hadoop, and OS page caching, 228
 - key principles, 228
 - overview, 227–228
 - short-circuit local reads, 231–232
- Channel selectors, 390
- Channels, 389–390
- check-column parameter, 378–379
- checkHealth command, 349, 535
- Checkpoint node, checkpointing, 324
- Checkpointing
 - extra edit logs, 326
 - failure, consequences of, 326
 - metadata files, 36
 - overview, 323
 - performance, 327
 - with a Secondary NameNode, 328–329
 - with a Standby NameNode, 327–328
- Checkpointing, configuring
 - backup node, 324–325
 - checkpoint node, 324
 - frequency, 325–327
 - “replaying edit logs” message, 326
 - Secondary NameNodes, 324
 - Standby NameNode, 325
- Checkpoints, NameNode operations, 319
- Chef, 569
- chgrp command, 250–251
- chmod command, 251
- Chokepoints, preventing, 395
- chown command, 250–251
- Chukwa, 576
- CLI (command line interface)
 - cacheadmin, 229, 230
 - fully distributed clusters, 104–105
 - managing HDFS. *See* dfsadmin utility.
- Clickstream data, definition, 6
- Client mode
 - vs.* cluster mode, 674–676
 - Spark applications, 186–187, 189, 190
- Client protocols, 511
- Client server, Oozie architecture
 - description, 440
 - installing, 445–446
- Clients
 - Hadoop, modifying ports in fully distributed clusters, 124–126
 - YARN, 49
- Clients, HDFS
 - block size, determining, 222
 - client interactions, 206
 - default behavior, settings for, 191
 - reading HDFS data, 219–220
 - replication factor, determining, 222
 - write considerations, 223–224
 - writing HDFS data, 221–224
- Cloning, Linux servers, 745–746
- Close stage, 227
- Cloudera, Hadoop distribution, 60
- Cloudera Manager, 434
- clrQuota command, 346
- clrSpaceQuota command, 346
- Cluster capacity, managing, 28–29
- Cluster computing. *See also* Hadoop clusters.
 - breaking up large workloads, 12
 - data redundancy, 12–13
 - description, 12
 - DFS (distributed file system), 13
 - embarrassingly parallel algorithms, 12
 - hardware racks, 12
 - tasks, 13
- Cluster managers, Spark applications, 180
- Cluster mode
 - vs.* client mode, 674–676
 - Spark, 158–159, 186–187, 189, 190–191
- Clusters
 - administering with Hue. *See* Hue, administering a cluster.
 - rack information, finding, 210–211, 212
 - redundancy, rack awareness, 209–210
 - resources, allocating, 28–29
 - shutdown/startup scripts, 546
 - storage, managing, 28
 - usage, displaying, 530
- coalesce operator, 708–709
- Code generation, Spark SQL query optimizer, 713–714
- Codecs, 293–294

- cogroup operator, 702
- Cold data
 - archival storage, 232, 233–234
 - storage policies, 237
- collect(0) operation, 720
- Collecting data. *See* Flume; Log aggregation.
- Collector nodes, 389
- Combiners, optimizing MapReduce, 652–654
- Command line interface (CLI)
 - cacheadmin, 229, 230
 - fully distributed clusters, 104–105
 - managing HDFS. *See* dfsadmin utility.
- Commands. *See also specific commands.*
 - executing remotely, 63
 - Oozie, 471
 - YARN. *See* YARN commands.
- Commands, help for
 - dfsadmin utility, 251–252
 - file commands, 260
 - hdfs dfs utility, 245–247
 - HDFS storage, 260
 - managing HDFS with hdfs dfs utility, 245–247
 - spark-submit script, 187–188
 - Sqoop, 368
 - YARN commands, 530
- Commit log abstraction, 399
- COMMITTED block state, 218–219
- Common, in the Hadoop ecosphere, 15
- Compactness, Spark, 152
- COMPLETE block state, 218–219
- Completed jobs, monitoring with web UIs, 604–606, 606–607
- Compression. *See* Data compression.
- Computation and storage, Hadoop 2
 - architecture, 34–35
- \$CONDITIONS parameter, 376
- Configuration files, precedence among, 76–78
- Configuration parameters
 - monitoring, 682
 - variable expansion, 78–79
- Configuring
 - authorization, 506
 - Beeswax, 560
 - caching, 229
 - capacity, queues, 417
 - control nodes, 456–460
 - decision control nodes, 459–460
 - desktop features, 559
 - end control nodes, 456
 - error nodes, 458–459
 - Fair Scheduler, 428–430
 - fork control nodes, 456–457
 - Hadoop, 557–560
 - Hadoop daemons, 79–81
 - Hadoop for Oozie, 444–446
 - Hadoop metrics, 75
 - Hadoop-specific environment, 80
 - HDFS storage directories, 262
 - HDFS transparent encryption, 522
 - Hive for Sentry, 516–517
 - Hue, 558–559
 - join control nodes, 456–457
 - KDC (Key Distribution Center), 489–490
 - Kerberos, 487–489
 - kill nodes, 458–459
 - KMS (Key Management Server), 522
 - log retention, 594–595
 - MapReduce. *See* Modifying fully distributed clusters, MapReduce configuration.
 - multiple archival storage tiers, 235–236
 - MySQL databases, 445, 548–549
 - ok control nodes, 458–459
 - Oozie, 560
 - Oozie action nodes, 449–454
 - Oozie workflow jobs, 460–461
 - permissions, 506
 - queues, Capacity Scheduler, 424
 - queues, Fair Scheduler, 429–430
 - rack awareness, 210
 - Sentry server for Hive, 515–516
 - shuffle parameters, 697
 - start control nodes, 456
 - subqueues, Capacity Scheduler, 414
 - super user permissions, 506
 - super users, 506
 - trash directory, 278–279
 - YARN, 559–560
 - YARN memory, 613–615
 - ZooKeeper, 560
- Configuring clusters. *See also* Installing pseudo-distributed clusters; Modifying fully distributed clusters; Planning fully distributed clusters.
 - basic HDFS parameters, 81

- Configuring clusters (*continued*)
 - Capacity Scheduler, 75
 - configuration parameters, variable expansion, 78–79
 - core Hadoop properties, 81
 - data block replication factor, changing, 85
 - data storage, 73–74
 - DataNode storage location, specifying, 85
 - default user name, 81
 - environment configuration, 73–74
 - Fair Scheduler, 75
 - file system, base temporary directory, 81
 - Hadoop daemons environment, 79–81
 - Hadoop metrics, 75
 - Hadoop-related configuration, 74–76
 - Hadoop-specific environment, 80
 - HDFS, 85–86
 - HDFS, base temporary directory, 81
 - HDFS daemons, setting up, 73–74
 - heap size, adjusting for the simple cluster, 80–81
 - including/excluding hosts, 75
 - initial Hadoop configuration, 71–72
 - job processing, 73–74
 - logging, 75
 - mapred environment, 80
 - MapReduce, 82–83
 - NameNode metadata file location, specifying, 85
 - NameNode service, file system, host, and port information, 81
 - NameNode URI, specifying, 85
 - precedence among configuration files, 76–78
 - read-only default configuration, 74
 - single-node. *See* Configuring pseudo-distributed Hadoop clusters.
 - site-specific configuration, 74
 - Standby NameNode metadata file location, specifying, 85
 - trash retention interval, setting, 81
 - YARN, configuring, 83–86
 - YARN daemons, setting up, 73–74
 - YARN environment, 80
- Configuring MapReduce, mapreduce shuffle, 83–84
- Configuring MapReduce, memory
 - ApplicationMaster, 617–618
 - for containers, 614
 - JVM heap size, 616–617
 - for map and reduce tasks, 615–616
 - memory-related configuration properties, 618–620
 - NodeManager, 617–618
 - ratio of physical memory to virtual, 617
 - virtual memory for map and reduce tasks, 617
- Configuring pseudo-distributed Hadoop clusters, 74
- Configuring Spark applications
 - configuration properties, 192–193
 - local file storage, specifying, 193
 - memory allocation, specifying, 193
 - spark.executor.memory property, 193
 - sparklocal.dir property, 193
 - with spark-submit script, 193–194
- Connectivity, checking, 68
- Connectors, Sqoop, 367
- Consumers, Kafka, 400, 403
- Containers
 - configuring MapReduce memory, 614
 - YARN, 50
- Context switches, monitoring, 571, 575
- Control nodes
 - configuring, 456–460
 - Oozie workflows, 446–447, 456–460
- Coordinator jobs, Oozie workflows, 438–439
- Coordinator status, checking, 472
- Coordinators. *See* Oozie.
- copyFromLocal command, 358
- Copying
 - data between hosts, 63
 - files from snapshots, 283
 - fsimage files, controlling transfer speed, 327
- copyToLocal command, 359
- core-site.xml file
 - core Hadoop properties, configuring, 81–82
 - default file system name, setting, 191
 - fs.defaultFS, 81
 - fs.trash.checkpoint.interval parameter, 278–279
 - fs.trash.interval parameter, 278

- hadoop.security.authorization property, 510
 - trash feature, configuring, 278
 - core-site.xml file, configuration parameters, 497–498
 - Counters. *See* Hadoop counters.
 - cp command, HDFS analog, 245
 - CPU
 - configuring virtual cores, 620–621
 - fully distributed clusters, single rack to multiple racks, 96–99
 - relationship between memory and virtual cores, 621
 - CPU usage
 - monitoring, 570–573
 - Spark on YARN, configuring resource allocation, 660
 - CPU_MILLISECONDS counter, 650
 - create command, 491
 - createSnapshot command, 281–282
 - Creating
 - application logs, 590–592
 - Capacity Scheduler queues, 413–414
 - Capacity Scheduler subqueues, 413–414
 - DataFrames, 198, 200–201
 - directories, 71, 249, 312
 - files, WebHDFS API, 312
 - fsimage files. *See* Checkpointing.
 - Hadoop clusters. *See* Installing Hadoop clusters.
 - Hadoop users, 70–71
 - HAR files, 305–306
 - Kerberos databases, 491
 - map/reduce containers, 590
 - queues, Capacity Scheduler, 413–414
 - snapshots, 281–282
 - SparkContext objects, 182
 - Sqoop jobs, 377
 - topics, Kafka clusters, 403
 - user accounts, 554–556
 - Creating fully distributed clusters. *See also* Modifying fully distributed clusters.
 - command line interface, 104–105
 - /etc/hosts file, editing, 105–106
 - overview, 102
 - passwordless SSH, configuring, 105
 - pdsh utility, installing, 102–106
 - setting up the test cluster, 102–106
 - Creating RDDs
 - from existing RDDs, 170
 - new, 178
 - subsets of other RDDs, 178
 - from text files, 175
 - cron scheduling. *See also* Time-based scheduling.
 - administration, 29–30
 - Oozie, 474
 - Crowbar, 63
 - CSV files, 679
 - curl tool, 63, 310–311
 - Custom Java counters, 651
- D**
- Daemon failures, troubleshooting, 737
 - Daemon logs
 - definition, 584
 - deleting log files, 598
 - location for, specifying, 597–598
 - log level, setting, 598–599
 - rotating log files, 598
 - DAG (directed acyclic graph), Spark
 - execution model, 693
 - DAG page, 684
 - Dashboard, Oozie, 555
 - Data access, with Spark, 164–166
 - Data at rest, encrypting, 520
 - Data block replication factor, changing, 85
 - Data blocks, data replication, 213
 - Data compression
 - codecs, 293–294
 - data serialization, 295
 - enabling, 293–294
 - file formats, 297
 - file sizes, 680
 - MapReduce, 133, 291–294
 - optimizing, 711–712
 - optimizing MapReduce, 654–655
 - optimizing shuffle operations, 697–698
 - overview, 289–290
 - SerDe module, 295
 - Spark, 295
 - stages of MapReduce, 292–293
 - table data, 373–374
 - tuning map tasks, 628
 - uses for, 681

- Data compression, common formats
 - Avro files, 679–680
 - bzip2, 290, 291
 - comparison of, 291
 - CSV files, 679
 - gzip, 290, 291
 - JSON files, 679
 - list of, 290
 - LZO, 290, 291
 - most common, 297
 - Parquet files, 680
 - SequenceFiles, 679
 - Snappy, 290, 291
- Data consistency model, HDFS, 38
- Data directories, specifying, 108
- Data formats. *See also* File formats.
 - HDFS, 42
 - most common, 297
- Data formats, compression
 - Avro files, 679–680
 - bzip2, 290, 291
 - CSV files, 679
 - gzip, 290, 291
 - JSON files, 679
 - LZO, 290, 291
 - Parquet files, 680
 - SequenceFiles, 679
 - Snappy, 290, 291
- Data in transit, encrypting, 520, 523–524
- Data ingestion. *See also* Flume; Kafka.
 - data science component, 11
 - parallelizing, 688
- Data integration
 - Flume, 27
 - Kafka, 27
 - overview, 27–28
- Data lakes, 9–11
- Data locality
 - Spark SQL query optimizer, 715–716
 - tuning map tasks, 626–627
- Data mining. *See* Data science.
- Data modeling, data science component, 11
- Data organization
 - data replication, 213
 - Hive, 142
- Data processing
 - Hadoop ecosystem, 16
 - parallelizing, 689
- Data redundancy. *See* Redundancy of data.
- Data replicas, distributing, 211
- Data replication
 - block states, 218–219
 - data blocks, 213
 - data organization, 213
 - default block size, 213
 - distributing data replicas, 211
 - fault tolerance, 43
 - finalizing an upgrade, 217
 - functional description, 213–214
 - HDFS data block storage in the Linux file system, 217
 - HDFS replication factor, 214–215
 - overview, 43
 - replica states, 216
- Data replication factors
 - decreasing, 274–275
 - default value, 85
 - determining, 222
 - effects on space quotas, 265–266
- Data science
 - components of, 11
 - definition, 11
- Data serialization, 295, 710–711
- Data storage. *See also* HDFS (Hadoop Distributed File system).
 - amount, single rack to multiple racks, 96
 - in a central location. *See* Data lakes.
 - configuring Hadoop clusters, 73–74
 - Hadoop ecosystem, 15
- Data streaming stage, 227
- Data transfer tools, 355–356
- Data types, 6
- Data wrangling, data science component, 11
- Database systems, traditional, 7–9
- Data-based Oozie coordinators, 467–469
- DataFrames. *See* Spark SQL, DataFrames.
- DATA_LOCAL_MAPS counter, 649
- DataNode web interface, fully distributed clusters, 120–121
- DataNodes
 - archival storage, 240–241. *See also* Archival storage.
 - balancing. *See* Rebalancing HDFS data.
 - communication with NameNodes, 207–208
 - extending clusters, 101
 - function of, 40

- in Hadoop clusters, 36
- HDFS, 38–39, 44
- large cluster guidelines, 101–102
- NameNode operations, 322–323
- network traffic issues, 97–98
- no longer alive, 207–208, 213–214
- periodic heartbeats, 207–208, 213–214
- planning for fully distributed clusters, 100–101
- relation to NameNodes, 44
- securing, 500
- starting, 87–88
- storage location, specifying, 85
- YARN, 49
- Debugging Spark applications, from the command line, 686
- Debugging Spark applications, viewing logs.
 - See also* Troubleshooting Spark jobs.
 - from HDFS, 741
 - with log aggregation, 740–741
 - reviewing the launch environment, 741
 - from the Spark web UI, 741
 - without log aggregation, 741
- decision, configuring, 459–460
- decision control nodes, Oozie workflows, 438, 446–447
- Decommissioning nodes. *See* Nodes, decommissioning and recommissioning.
- Default context metrics, 577
- default rule, 430
- Default user name, configuring Hadoop clusters, 81
- defaultQueueSchedulingPolicy, 430–431
- Delegation tokens, 501
- DELETE operation, 308, 312–313
- delete option, 286, 289
- deleteSnapshot command, 282, 283
- Deleting
 - daemon logs, 598
 - files, 278–279. *See also* Trash directory.
 - log files, 598
 - snapshots, 281–282
 - SPNs (service principal names), 493
- Dell Crowbar, 63
- delprinc command, 493
- Deploying
 - HA (high availability), 342–345
 - a high availability cluster, 544
 - Oozie workflow jobs, 463
 - Sqoop, 367
- Deploying, Oozie
 - configuring Hadoop for Oozie, 444–446
 - installing Oozie, 441–442
 - installing Oozie server, 442–444
 - MySQL database, configuring, 445
 - overview, 441–442
 - workflow jobs, 463
- Desktop features, configuring, 559
- df command, 260
- DFS (distributed file system), 13
- DFS metrics, 577
- dfs -setRep option, 275
- dfs utility
 - checking space quotas, 266
 - count q command, 266
 - createSnapshot command, 281–282
 - deleteSnapshot command, 282, 283
 - snapshots, creating/deleting, 282
- dfsadmin commands, HA (high availability), 346
- dfsadmin utility
 - allowSnapshot command, 281
 - checking for free space, 260
 - clearing space quotas, 265, 267
 - clrQuota command, 346
 - clrSpaceQuota command, 265, 267, 346
 - disallowSnapshot command, 281
 - examining HDFS cluster status, 252–255
 - fetchimage command, 320
 - help command, 251–252
 - metasave command, 254
 - name quotas, specifying, 264
 - printTopology command, 211
 - rack awareness, 211–212
 - refreshNodes command, 254
 - setQuota command, 264, 346
 - setSpaceQuota command, 265, 346
 - setting space quotas, 265
 - snapshots, enabling/disabling, 281
 - updating NameNodes, 254
- dfsadmin utility, report command
 - “Access denied...” message, 256–257
 - calculating threshold values, 271–272
 - overview, 252–254
- dfs.block.size parameter, 107

- dfs.client.read.shortcircuit.streams.cache.expiry.ms parameter, 564
- dfs.client.read.shortcircuit.streams.cache.size parameter, 564
- dfs.data.dir parameter, 262
- dfs.datanode.available-space-volume-choosing-policy.balanced-spacepreference-fraction property, 548
- dfs.datanode.available-space-volume-choosing-policy.balanced-spacethreshold property, 548
- dfs.datanode.data.dir parameter, 85, 108, 235–236
- dfs.datanode.du.reserved parameter, 107
- dfs.datanode.du.reserved parameter, setting, 730
- dfs.datanode.fsdataset.volume.choosing.policy property, 548
- dfs.datanode.kerberos.principal parameter, 499
- dfs.datanode.keytab.file parameter, 500
- dfs.datanode.reserved parameter, 262
- dfs.ha.fencing.methods attribute, 340–341
- dfs.image.transfer.bandwidthPerSec parameter, 327
- dfs.image.transfer.timeout parameter, 327
- dfs.journalnode.kerberos.keytab.file parameter, 499
- dfs.namenode.checkpoint.dir parameter, 85
- dfs.namenode.checkpoint.period parameter, 325
- dfs.namenode.checkpoint.txns parameter, 325
- dfs.name.node.dir parameter, 108
- dfs.namenode.http-bind-host parameter, 124, 563
- dfs.namenode.https-bind.host parameter, 124, 563
- dfs.namenode.kerberos.internal.spnego.principal parameter, 499
- dfs.namenode.kerberos.principal parameter, 499
- dfs.namenode.keytab.file, 499
- dfs.namenode.max.extra.edits.segments.retained parameter, 327
- dfs.namenode.name.dir parameter, 85–87
- dfs.namenode.num.extra.edits.retained parameter, 326
- dfs.namenode.rpc.bind-host parameter, 563
- dfs.namenode.rpc.bind-host parameter, 124
- dfs.namenode.servicerpc-bind.host parameter, 124, 563
- dfs.permissions.enabled parameter, 255, 506
- dfs.permissions.supergroup parameter, 259
- dfs.permissions.superusergroup parameter, 108
- dfs.replication parameter, 107–108
- dfs.secondary.namenode.kerberos.internal.spnego.principal parameter, 499
- dfs.secondary.namenode.kerberos.principal parameter, 499
- dfs.secondary.namenode.keytab.file, 500
- dfs.storage.policy.enabled parameter, 235
- dfs.web.authentication.kerberos.keytab parameter, 499
- dfs.web.authentication.kerberos.principal parameter, 499
- dfw.replication parameter, 275
- dict_file parameter, 490
- direct parameter, 382
- Directed acyclic graph (DAG), Spark execution model, 693
- Directories. *See also* Files and directories.
 - creating in WebHDFS, 312
 - renaming, 283
 - snapshottable, removing, 283
- Directory quotas, checking, 313
- Directory specific space quotas, 264
- disallowSnapshot command, 281
- Disaster recovery, administrator duties, 20.
 - See also* Backup and recovery; Snapshots.
- Discretized Stream (DStream), 196, 688
- Disk configuration for fully distributed clusters, single rack to multiple racks, 97–98
- Disk failure risk, fully distributed clusters, 98
- Disk I/O, optimizing shuffle operations, 696–697
- Disk sizing for fully distributed clusters, single rack to multiple racks, 97–98
- Disk speed, testing, 65
- Disk storage, monitoring, 571–572
- DISK storage type, 236
- Disk usage, checking, 260
- Disk volume failure toleration, troubleshooting, 729–730
- Disk-based archival storage, 236

- DISK_ONLY storage level, 719
 - DistCp
 - default behavior, 364
 - description, 356
 - moving data between clusters, 361–363
 - moving data within a cluster, 363
 - overwriting target files, 364–365
 - potential problem, 356
 - updating target files, 364–365
 - distcp command
 - example, 362–363
 - moving data between clusters, 361–363
 - moving data within a cluster, 361–363
 - options, 363–364. *See also specific options.*
 - overwrite option, 364–365
 - syntax, 361
 - update option, 364–365
 - distinct transformation, 178
 - Distributed computing, fault tolerance requirements, 33
 - Distributed data processing
 - Hive, 26
 - HiveQL, 26
 - MapReduce, 24–25
 - Pig, 26
 - Spark, 25–26
 - Distributed file system (DFS), 13
 - Distributing data replicas, rack awareness, 211
 - DNS, checking, 65–66
 - Double RDDs, 179
 - Downloading, fsimage files, 320–321
 - DRF (Dominant Resource Fairness)
 - scheduler, 28–29, 426. *See also Fair Scheduler.*
 - Drivers
 - Spark applications, 180
 - Sqoop, 367
 - standalone cluster manager, 159
 - Drivers, Spark on YARN
 - in client mode, 664–665
 - in cluster mode, 665–666
 - duties, 663–664
 - Dry runs, Oozie, 472
 - ds.replication parameter, 85
 - dstat command, 576
 - DStream (Discretized Stream), 196, 688
 - du command, 260–262
 - du -h command, 261
 - du -s command, 262
 - Dumping a file's contents, 356–357
 - Duplicate RDDs, filtering out, 178
 - Dynamic resource allocation, 667, 676–678
 - Dynamic resource allocation, enabling, 677–678
 - Dynamic workflows, 463–464
- ## E
- Ease of use, Spark, 151–152
 - Edge servers, 13
 - Edit logs
 - definition, 318
 - extra, 326
 - overview, 320–321
 - Elasticity, queues, 414
 - Elasticsearch. *See* ELK (Elasticsearch/Logstash/Kibana).
 - ELK (Elasticsearch/Logstash/Kibana), 27
 - Email data, definition, 6
 - Embarrassingly parallel algorithms, 12
 - Emptying the trash directory, 250
 - EMR (Amazon Elastic MapReduce), 307
 - Enabling, trash directory, 278
 - Encrypting, HDFS, 523
 - Encryption
 - 256-byte encryption, enabling/disabling, 490
 - data at rest, 520
 - data in transit, 520, 523–524
 - HDFS data transfer protocol, 524
 - Encryption, HDFS transparent architecture, 521
 - configuring encryption, 522
 - configuring KMS, 522
 - dedicated server, 521
 - encrypting HDFS, 523
 - encryption zones, 521
 - functional description, 521
 - KMS (Key Management Server), 521
 - Encryption zones, 521
 - end control nodes
 - configuring, 456
 - Oozie workflows, 438, 446–447, 448, 456
 - Environment configuration, configuring
 - Hadoop clusters, 73–74

- Environment tab, web UI, 682
 - Environment variables, setting, 87
 - env-var element, 453
 - error nodes, configuring, 458–459
 - /etc/hosts file, editing, 105–106
 - Events, 390
 - Exec, 394
 - exec element, 453
 - Execute (x) permission, 506
 - Executing, Spark applications, 187–189
 - Executing remote commands, 63
 - executor-cores flag, 661
 - Executors. *See* Spark executors.
 - export command, 382–383
 - Exporting data. *See* Sqoop, exporting data.
 - expunge command, 250, 279
 - Extended attributes, 509–510
 - Extending clusters, single rack to multiple racks, 101
 - extJS, 440, 443
 - Extracting configuration files, 581
- F**
- Failed jobs, monitoring with web UIs, 601–602
 - “Failed to find any kerberos tgt” message, 502
 - Failover. *See* HA (high availability), failover.
 - failover command
 - manual failover, 348–349, 545–546
 - YARN, 535
 - Fair Scheduler. *See also* Capacity Scheduler; DRF (Dominant Resource Fairness) scheduler; Oozie.
 - allocation files, 428
 - application preemption, 431–432
 - vs.* Capacity Scheduler, 435–436
 - configuring, 428–430
 - configuring Hadoop clusters, 75
 - description, 409
 - fair-scheduler.xml file, example, 432–434
 - monitoring, 434
 - overview, 426–427
 - preemption, 409
 - priorities, 409
 - queues, 409–410
 - security, 432
 - Fair Scheduler, queues
 - configuring, 429–430
 - leaf queues, 428
 - moving applications between, 434
 - overview, 427–428
 - rules for placing jobs into, 430–431
 - scheduling policy, configuring, 431
 - submitting jobs to, 434
 - Fair share preemption, Capacity Scheduler, 421–422
 - FairScheduler, 28–29
 - fair-scheduler.xml file
 - configuring queues in the Fair Scheduler, 429–430
 - example, 432–434
 - Fallback storage media, archival storage, 235
 - Fault tolerance. *See also* Rack awareness; Recovery process.
 - caching RDD data, 718
 - data replication, 43
 - HDFS, 37–38
 - Spark jobs, troubleshooting, 740
 - Federated NameNodes
 - architecture, small files problem, 304
 - description, 349–350
 - Federation. *See* Federated NameNodes.
 - Fedora Linux, package manager for, 63
 - Fencing, configuring, 340–341
 - fetchimage command, 320, 552–553
 - FIFO (first-in, first-out) scheduler, 409, 410–411
 - File formats. *See also* Data formats.
 - changing, 302
 - compatibility with processing tools, 297
 - compression capability, 297
 - file size, 297
 - flexibility, 296
 - overview, 295–296
 - performance, 297
 - selecting, 296–297
 - splittability, 297
 - File sizes
 - choosing a file format, 297
 - data compression, 680
 - File system checks. *See also* fsck command.
 - block locations, printing, 287
 - “block MISSING” messages, 287–288
 - block reports, generating, 287
 - detecting data corruption, 285–288
 - fully distributed clusters, 118

- removing corrupt files, 286
- under-replicated files, 289
- unrecoverable files, 288–289
- File system counters, 649
- File system organization, HDFS, 42
- FILE_BYTES_READ counter, 649
- FILE_BYTES_WRITTEN counter, 649
- Filecrush project, 307
- filecrusher utility, 306
- Files
 - archival storage preferences for, 235
 - dumping contents of, 356–357
 - sending and getting, 63
 - small. *See* Small files.
 - testing for, 357
- Files and directories. *See also* HDFS storage,
 - files and directories; Linux file and directory commands.
 - change directory, 245
 - copying, 245
 - listing files, 244–245
 - moving, 245
 - permissions for. *See* HDFS permissions.
 - print working directory, 245
 - setting space quota limits on directories, 264–266
- File system in Userspace (FUSE), 564–566
- filter() operation, 200
- filter(function) transformation, 178
- Filtering
 - DataFrame rows, 200
 - duplicate RDDs, 178
 - lists of applications, 531–532
- FINALIZED replica state, 216
- Finalizing a data replication upgrade, 217
- Firewall, turning off, 67
- First-in, first-out (FIFO) scheduler, 409, 410–411
- Flash storage, storage policies, 237
- flatMap, transformation, 178
- Flink, 25
- Flume. *See also* Log aggregation.
 - architecture, 389–391
 - channel selectors, 390
 - channels, 389–390
 - collector nodes, 389
 - description, 17
 - events, 390
 - examples, 392–394, 395–398
 - intercepts, 390
 - key components, 389–390
 - memory channels, 392
 - moving data to HDFS, 394–395
 - overview, 388–389
 - preventing chokepoints, 395
 - sink processors, 390
 - sinks, 389–390, 395
 - sources, 389–390, 395
- Flume agents
 - agent nodes, 389
 - configuring, 391–392, 396–398
 - definition, 389
 - description, 390–391
- Folder utility, 643–644
- Fork actions, Oozie workflows, 448
- fork control nodes
 - configuring, 456–457
 - Oozie workflows, 438, 446–447, 448, 456–457
- Formats. *See* Data formats; File formats.
- Fraud detection, advantages of Hadoop, 9
- free command, 573
- Free form import, 375–376
- fromSnapshot parameter, 282–283
- Frozen data, archival storage, 232, 233–234
- fs actions, Oozie action nodes, 454
- fs.block.size parameter, 275
- fsck command. *See also* File system checks.
 - blocks option, 287
 - delete option, 286, 289
 - detecting data corruption, 285–288
 - FAILED error, 256
 - file system check, 118
 - file system check options, 288
 - vs.* Linux fsck command, 284
 - list-corruptfileblocks option, 286
 - locations option, 287
 - move option, 288–289
 - options, summary of, 288. *See also specific options.*
 - rack awareness, 211
 - removing corrupt files, 286
- fs.defaultFs parameter, 81
- fs.default.name parameter, 85

- fsimage files. *See also* Snapshots.
 - copying, controlling transfer speed, 327
 - definition, 318
 - downloading, 320–321
 - importance of updating, 323–324
 - location, specifying, 108
 - loss or corruption, 319
 - overview, 320–321
 - viewing contents of, 321
 - fs.trash.checkpoint.interval parameter, 278–279
 - fs.trash.interval parameter, 81, 278
 - FTP protocol, enabling, 63
 - Full garbage collection, 687
 - Fully distributed clusters, description, 61–62.
 - See also* Creating fully distributed clusters;
 - Modifying fully distributed clusters;
 - Planning fully distributed clusters.
 - FUSE (File system in Userspace), 564–566
- G**
- Ganglia, installing, 580–581. *See also* Monitoring with Ganglia.
 - Garbage collection. *See* GC (garbage collection).
 - Gateway machines, fully distributed clusters, 119
 - GC (garbage collection)
 - collecting statistics about, 687–688
 - Full GC, 687
 - for the JVM (Java Virtual Machine). *See* JVM garbage collection.
 - mechanics of, 687
 - Minor GC, 687
 - monitoring with web UIs, 684, 685
 - Old Generation, 687
 - optimizing shuffle operations, 697
 - tuning, 686–689
 - Young Generation, 687
 - GC_TIME_MILLIS counter, 650
 - generate option, 645
 - Generation Stamp (GS), 224
 - Generations, JVM garbage collection, 732–733
 - Geographic data, definition, 6
 - get command, 359–360
 - GET operation, 308
 - getconf command, 333–334
 - getMerge command, 360
 - getServiceState command, 349, 535, 545
 - Getting, files, 63
 - gmetad daemon, 580–581
 - gmond daemon, 580–581
 - Graphs, Spark, 155
 - GraphX, 155
 - GridMix, benchmarking clusters, 644–646
 - gridmix command, 645
 - gridmix.compression-emulation.enable parameter, 645
 - gridmix.job-submission.policy parameter, 645
 - gridmix.job.type parameter, 645
 - gridmix.output.directory parameter, 645
 - Group metrics, 577
 - groupBy operation, 200
 - groupByKey operator, 700–702
 - Grouping DataFrame data, 200
 - Groups, Sentry authorization, 513, 514
 - Growth patterns of fully distributed clusters,
 - single rack to multiple racks, 96
 - GS (Generation Stamp), 224
 - gweb process, 580
 - gzip format, 290, 291
- H**
- H option, 310
 - HA (high availability)
 - functional description, 336–337
 - Hadoop 2 *vs.* Hadoop 1, 22
 - MySQL databases, 549–551
 - Standby NameNode, 46–47
 - HA (high availability), configuring
 - JournalNodes, role of, 336
 - overview, 335
 - QJM (Quorum Journal Manager), 335
 - ZooKeeper as a coordinator, 335
 - HA (high availability), failover
 - attributes, configuring, 340–341
 - automatic, configuring, 347–348
 - manual, 348–349, 545–546
 - NameNode health, checking, 349
 - NameNode status, displaying, 349
 - ResourceManager, 543–544
 - transitioning node status, 349
 - ZKFC (ZKFailoverController), 347–348
 - HA (high availability), NameNode setup
 - deploying, 342–345

- dfsadmin commands, 346
- managing, 345–346
- Standby NameNode, query errors, 346
- testing, 345
- HA (high availability), ResourceManager
 - architecture, 541–542
 - commands, 545
 - current state, getting, 545
 - current state, transitioning, 545
 - deploying a high availability cluster, 544
 - failover, 543–544
 - failover command, 545
 - getServiceState command, 545
 - Restart feature, 543
 - setting up, 542–543
 - transitionToStandby command, 545
- HA (high availability) quorum cluster
 - failover attributes, configuring, 340–341
 - fencing, configuring, 340–341
 - logical NameNode ID, 337
 - logical nameservice, 337
 - name and address, configuring, 338–340
- haadmin commands, 348–349
- Hadoop
 - block sizes, effects on space quotas, 265–266
 - configuring, 557–560
 - daemons, configuring, 79–81
 - distributions, 60–61
 - integrating with Kafka, 404–406
- Hadoop 2
 - architecture. *See* Architecture, Hadoop 2.
 - common uses for, 6
 - components of. *See* Hadoop ecosphere.
 - ease of adoption, 12
 - handling large datasets, 11
 - key success factors, 8–9
 - scale up architecture *vs.* scale out, 8
 - unique features, 5
 - user identities, determining, 258–259
- Hadoop 2 *vs.* Hadoop 1
 - applications supported, 23
 - architectural differences, 22
 - high availability features. *See* HA (high availability).
 - multiple processing engines, 23
 - resource allocation, 24
 - separation of processing and scheduling, 23
 - YARN, 21–22
- Hadoop Archives (HAR)
 - caveats, 306
 - file types, 305
 - .har file extension, 305
 - HAR files, creating, 305–306
 - HAR files, reading, 306
 - managing small files, 304–306
 - overview, 304–306
- Hadoop clusters. *See also* Cluster computing.
 - allocating resources, 36
 - ApplicationMaster, 36
 - architecture, 13–14, 35
 - checkpointing the metadata file, 36
 - components of, 13
 - configuring. *See* Configuring clusters.
 - coordinating application execution, 36
 - creating. *See* Installing Hadoop clusters.
 - DataNodes service, 36
 - definition, 13
 - edge servers, 13
 - Hadoop services, 36
 - HDFS services, 36
 - HDFS storage metadata, 36
 - master nodes, 36
 - NameNode service, 36
 - NodeManager, 37
 - operating. *See* Operating Hadoop clusters.
 - ResourceManager, 36
 - Secondary NameNode service, 36
 - Standby NameNode service, 36
 - worker nodes, 36
 - YARN (Yet Another Resource Negotiator) services, 36–37
- Hadoop counters. *See also* Benchmarking
 - clusters; Hadoop metrics.
 - custom Java counters, 651
 - file system counters, 649
 - job counters, 649–650
 - limiting the number of, 651–652
 - MapReduce framework counters, 650–651
 - overview, 647–648
- Hadoop counters, key counters
 - CPU_MILLISECONDS, 650
 - DATA_LOCAL_MAPS, 649
 - FILE_BYTES_READ, 649

- Hadoop counters, key counters (*continued*)
 - FILE_BYTES_WRITTEN, 649
 - GC_TIME_MILLIS, 650
 - HDFS_BYTES_READ, 649
 - HDFS_BYTES_WRITTEN, 649
 - MAP_INPUT_RECORDS, 650
 - MAP_OUTPUT_RECORDS, 650
 - MILLIS_MAPS, 650
 - MILLIS_REDUCE, 650
 - NUM_KILLED_MAPS, 649
 - NUM_KILLED_REDUCE, 649
 - PHYSICAL_MEMORY_BYTES, 650
 - REDUCE_SHUFFLE_BYTES, 650
 - SPILED_RECORDS, 650
 - TOTAL_LAUNCHED_MAPS, 649
 - TOTAL_LAUNCHED_REDUCE, 649
- Hadoop daemon starting failures,
 - troubleshooting, 737–738
- Hadoop Distributed File system (HDFS). *See* HDFS (Hadoop Distributed File system).
- Hadoop ecosystem
 - Avro, 17
 - base utilities, 15
 - Catalog, 17
 - Common, 15
 - coordinating distributed applications. *See* ZooKeeper.
 - data processing, 16
 - data storage, 15
 - diagram of, 16
 - Flume, 17
 - HBase, 17
 - HDFS (Hadoop Distributed File system), 15
 - Hive, 17
 - Hue, 17
 - Kafka, 17
 - Mahout, 17
 - management tools, 16. *See also* Ambari.
 - managing resources, 15
 - MapReduce, 15
 - monitoring tools, 16
 - Oozie, 17
 - operating system, 15
 - Pig, 17
 - scheduling jobs, 15
 - Sqoop, 17
 - Storm, 17
 - summary of components, 17
 - Tez, 17
 - YARN, 15
 - ZooKeeper, 17
- Hadoop metrics. *See also* Benchmarking
 - clusters; Hadoop counters; Monitoring.
 - capturing to a file system, 578–579
 - configuring, 75
 - default context, 577
 - DFS, 577
 - group, 577
 - JVM, 577
 - overview, 576
 - RPC, 577
 - sinks, 578–579
 - sources, 578–579
 - types of, 577
 - user, 577
 - uses for, 578
 - YARN, 577
- Hadoop Process Definition Language (hPDL), 447
- Hadoop services, in Hadoop clusters, 36
- Hadoop Streaming
 - definition, 139–140
 - functional description, 140
 - Java classes, 140
- Hadoop web interfaces, fully distributed clusters, 120
- HADOOP_CLASSPATH environment variable, 73
- HADOOP_CONF_DIR environment variable, 163
- hadoop.encrypted.key.provider.path parameter, 522
- hadoop.encrypted.key.provider.url parameter, 522
- hadoop-env.sh file, 79
- HADOOP_HEAPSIZE environment variable, 73
- hadoop.http.staticuser, setting default user name, 81
- HADOOP_LOG_DIR environment variable, 73
- HADOOP_LOG_DIR parameter, 597
- hadoop-metrics.properties file, configuring Hadoop metrics, 75

- HADOOP_PID_DIR environment variable, 73
- Hadoop-related configuration, 74–76
- hadoop.rpc.protection parameter, 497–498
- hadoop.security.authentication parameter, 497–498
- hadoop.security.authentication property, 258–259
- hadoop.security.authorization parameter, 497–498
- hadoop.security.auth_to_local parameter, 495–498
- hadoop.security.group.mapping parameter, 496
- Hadoop-specific environment, configuring, 80
- hadoop.tmp.dir, 81
- HAR (Hadoop Archive)
 - caveats, 306
 - file types, 305
 - .har file extension, 305
 - HAR files, creating, 305–306
 - HAR files, reading, 306
 - managing small files, 304–306
 - overview, 304–306
 - .har file extension, 305
- HAR file system, 244
- HAR files, 305–306
- hard limit settings, 67–68
- Hardware racks, 12. *See also* Planning fully distributed clusters, single rack to multiple racks.
- HashPartitioner partitions, 709
- HBase, 17
- HCatalog, 558
- HDFS (Hadoop Distributed File system)
 - accessing from behind a firewall. *See* HttpFS gateway.
 - administrative access, service level authorization, 511
 - alternate file systems, 244
 - architecture. *See* Architecture, HDFS.
 - Avro format, 42
 - base temporary directory, configuring, 81
 - binary formats, 42
 - cache management. *See* Centralized cache management.
 - client interactions. *See* Clients, HDFS.
 - cluster status, examining, 252–255
 - configuring for fully distributed clusters, HDFS configuration, modifying fully distributed clusters
 - configuring Hadoop clusters, 85–86
 - daemons, configuring in Hadoop clusters, 73–74
 - data block storage in the Linux file system, 217
 - data consistency model, 38
 - data formats, 42
 - data replication. *See* Data replication.
 - data transfer protocol, 524
 - DataNode services, starting, 87–88
 - DataNodes, 44
 - distributed synchronization and group services. *See* ZooKeeper.
 - fault tolerance, 37–38. *See also* Recovery process.
 - federation. *See* Federated NameNodes.
 - file system organization, 42
 - formatting, 86–87
 - in Hadoop clusters, 36
 - Hadoop ecosphere, 15
 - handling large datasets, 37
 - high availability. *See* HA (high availability).
 - loading data into. *See* Loading data.
 - managing. *See* Managing HDFS.
 - metadata, 319–321. *See also* NameNodes.
 - metadata, backing up, 552–553
 - mountable file systems, 564–566
 - in a multihomed network, 124, 562–563
 - NameNode operations, 43–48
 - NameNodes, communication with DataNodes, 207–208
 - NameNodes, starting, 87–88
 - NFSv3 gateway, configuring, 566–567
 - operations, auditing, 519
 - parameters, configuring Hadoop clusters, 81
 - permissions. *See* Authorization, HDFS permissions.
 - ports, modifying in fully distributed clusters, 123–124
 - reading, 219–220
 - remote communication. *See* HttpFS gateway; WebHDFS.
 - replication factor, 214–215

HDFS (*continued*)

- Secondary NameNodes, 46–47, 87–88
- SequenceFile format, 42
- services, starting, 87–88
- short-circuit local reads, 563–564
- space issues, troubleshooting, 727
- special features, 562–567
- storage metadata in Hadoop clusters, 36
- storage usage, monitoring with web UIs, 608–609
- storing data, 40–42
- streaming access to data, 38
- transparent encryption. *See* Encryption, HDFS transparent.
- unbalanced data. *See* Rebalancing HDFS data.
- unique features, 37–38
- write considerations, 223–224
- writing data, 221–224
- writing to an HDFS file, 42–43
- hdfs dfs utility. *See* Managing HDFS with hdfs dfs utility.
- hdfs dfsadmin command, 118
- HDFS files
 - concatenating, 360
 - description, 206
 - listing, 247, 248
 - piping data into, 360
 - viewing first and last portions of, 360
- HDFS permissions
 - ACLs (access control lists), 256
 - checking, 255
 - default authentication mode, 257
 - enabling new users, 257–258
 - Kerberized systems, 257
 - overview, 255
 - Permission denied errors, 256
 - r (read), 255–256
 - super users, designating, 259
 - user identities, 258–259
 - w (write), 255–256
 - x (execute), 256
- HDFS storage
 - additional space, checking for, 262
 - checking disk usage, 260
 - decreasing the replication factor, 274–275
 - df command, 260
 - dfs -setRep option, 275
 - dfsadmin command, 260
 - dfw.replication parameter, 275
 - displaying storage statistics, 263
 - du command, 260–262
 - du -h command, 261
 - du -s command, 262
 - free space, checking, 260
 - fs.block.size parameter, 275
 - help for file commands, 260
 - reclaiming used space, 274–276
 - report command, 263
 - reserving space for non-HDFS data use, 262
 - test command, 263
 - used space, checking, 260–262
- HDFS storage, files and directories
 - checking the existence of files, 263
 - distinguishing directories from files, 263
 - removing, 274
- HDFS storage, space quotas
 - checking, 266–267
 - Hadoop block sizes, effects of, 265–266
 - managing, 265
 - name quotas, setting, 264
 - vs.* name quotas, 263
 - quota violation state, 266
 - removing, 265
 - replication factors, effects of, 265–266
 - setting limits on directories, 264–266
 - user specific *vs.* directory specific, 264
- hdfs user, setting up, 70–71
- HDFS_BYTES_READ counter, 649
- HDFS_BYTES_WRITTEN counter, 649
- hdfs-site.xml file
 - configuring archival storage tiers, 235–236
 - configuring HDFS storage directories, 262
 - configuring pseudo-distributed Hadoop clusters, 74
 - decommissioning a DataNode service, 536
 - default behavior for HDFS client, 191
 - dfs.data.dir parameter, 262
 - dfs.datanode.reserved parameter, 262
 - dfs.hosts.exclude parameter, 536
 - dfs.namenode.http-bind-host parameter, 124, 563

- dfs.namenode.https-bind.host parameter, 124, 563
- dfs.namenode.rpc.bind-host parameter, 124, 563
- dfs.namenode.servicerpc-bind.host parameter, 124, 563
- dfs.permissions.enabled parameter, 255
- dfs.permissions.supergroup parameter, 259
- hadoop.security.authentication property, 258–259
- HDFS in a multihomed network, 124
- modifying HDFS configuration, 106–109
- permission checking, enabling/disabling, 255
- super users, designating, 259
- user identities, determining, 258–259
- YARN in a multihomed network, 124
- hdfs-site.xml file, configuration parameters, 499–500
- hds-audit.log file, 519
- head command, 360
- Heap dumps, 734
- Heap size, adjusting for the simple cluster, 80–81
- Heartbeats
 - DataNodes, 207–208, 213–214
 - frequency, configuring, 321
 - overview, 322
 - piggybacking, 322
 - stopped, 322
- help command
 - dfsadmin utility, 251–252
 - hdfs dfs utility, 245–247
- Help feature, Sqoop, 368
- Help for commands
 - dfsadmin utility, 251–252
 - file commands, 260
 - hdfs dfs utility, 245–247
 - HDFS storage, 260
 - managing HDFS with hdfs dfs utility, 245–247
 - spark-submit script, 187–188
 - Sqoop, 368
 - YARN commands, 530
- Heterogeneous HDFS storage, archival storage, 233–234
- HiBench, benchmarking clusters, 642–643
- Hierarchical queues, 414, 416. *See also* Leaf queues.
- High availability (dfsadmin commands, HA), 346
- High availability (HA). *See* HA (high availability).
- History file directory, specifying, 114
- History files, managing, 114
- Hive
 - alternative to MapReduce, 25
 - authorization, 514–516
 - connecting to Spark SQL, 199
 - data organization, 142
 - definition, 141
 - description, 9, 17
 - executing under Sentry, 517
 - loading data into, 142–143
 - monitoring, 609–610
 - overview, 141–142
 - partitioned Hive tables, 381
 - querying data, 143
 - SQL features. *See* HiveQL.
- Hive jobs
 - Oozie action nodes, 451–452
 - optimizing. *See* Optimizing Hive jobs.
- Hive Query Language (HQL), 164, 452
- Hive tables, 142
- HiveContext, 198–199
- hive-partition-key parameter, 381
- hive-partition-value parameter, 381
- HiveQL, 26
- hive.sentry.provider property, 515
- HiveServer2, 514–517
- Hortonworks, 60
- Hosts, including/excluding, 75
- Hot data
 - archival storage, 232, 233–234
 - storage policies, 237
- Hot swapping a disk drive, 729
- HotSpot, 624
- hPDL (Hadoop Process Definition Language), 447
- HQL (Hive Query Language), 164, 452
- HSQldb, 372
- HTTP protocol, enabling, 63

- HttpFS gateway
 - accessing HDFS, 314–315
 - configuring, 313–314
 - overview, 313
 - vs.* WebHDFS, 315
- Hue
 - configuring, 558–559
 - in the Hadoop ecosystem, 17
- Hue, administering a cluster
 - administrative tasks, 561–562
 - Beeswax, configuring, 560
 - configuring Hue, 558–559
 - creating user accounts, 554–556
 - desktop features, configuring, 559
 - Hadoop, configuring, 557–560
 - installing, 556–557
 - managing Hue, 561
 - managing workflows, 561–562
 - Oozie, configuring, 560
 - overview, 553–554
 - starting the Hue server, 561
 - user impersonation, 558
 - YARN, configuring, 559–560
 - ZooKeeper, configuring, 560
- I**
- IBM, Hadoop distribution, 60
- if-then-else actions. *See* decision control nodes.
- Impala, alternative to MapReduce, 25
- import command, 368–370
- Import process
 - incremental imports, 378–379
 - input parsing options, 373
 - overview, 368–371
 - selective import, 374–376
 - into SequenceFiles, 373
- import-all-tables command, 376
- Importing data. *See* Loading data.
- incremental parameter, 378–379
- Ingesting data. *See* Loading data.
- Initializing, Spark SQL, 199
- In-memory archival storage, 237
- In-memory computation, Spark, 151
- Input split size, tuning map tasks, 627–628
- InputFormat, 164
- Input/output
 - delimiters, 372–373
 - directories, MapReduce, 137
 - MapReduce, 132
 - tuning map tasks, 627–630
- Installation and upgrades, administrator duties, 19
- Installing
 - client server, Oozie architecture, 445–446
 - Ganglia, 580–581
 - Hue, 556–557
 - Kafka, 401
 - Kerberos, 486–487
 - OEL (Oracle Enterprise Linux), 745
 - Oozie, 441–442
 - Oozie server, 442–444
 - server, Oozie architecture, 442–444
- Installing fully distributed clusters, 61–62.
 - See also* Modifying fully distributed clusters; Planning fully distributed clusters.
 - checking the new file system, 118
 - overview, 61–62
 - starting up and shutting down the cluster, 114–117
- Installing Hadoop clusters. *See also*
 - Configuring clusters.
 - Hadoop distributions, 60–61
 - installation types, 61–62
 - multinode clusters. *See* Fully distributed clusters.
 - single-node installation. *See* Installing pseudo-distributed clusters.
 - standalone installation, 61–62
- Installing Java, 69–70
- Installing pseudo-distributed clusters. *See also*
 - Configuring pseudo-distributed Hadoop clusters.
 - description, 61–62
 - Hadoop users, creating, 70–71
 - HDFS management, setting up, 70–71
 - hdfs user, setting up, 70–71
 - installing Hadoop software, 70
 - Java requirements, 69–70
 - mapred user, setting up, 70–71
 - MapReduce services, setting up, 70–71
 - overview, 62–63
 - passwordless connection, 68–69
 - required directories, creating, 71

- setting up SSH, 68–69
 - starting up and shutting down the cluster, 114–117
 - utilities, 63
 - YARN services, setting up, 70–71
 - yarn user, setting up, 70–71
 - Installing pseudo-distributed clusters,
 - modifying the Linux kernel
 - connectivity, checking, 68
 - DNS, checking, 65–66
 - increasing file limits, 64
 - IP tables, disabling, 67
 - IPv6, disabling, 67
 - modified parameters, summary of, 64
 - NIC bonding, 65
 - noatime for disk mounts, setting, 65
 - nodiratime for directory mounts, setting, 65
 - NTP, enabling, 65
 - SELinux, disabling, 66
 - server BIOS settings, checking, 65
 - shell limits, setting, 67–68
 - swap, disabling, 66
 - testing disk speed, 65
 - THP compaction, turning off, 68
 - turning off the network firewall, 67
 - Ulimits, setting, 67–68
 - Installing Spark
 - compiling binaries, 157
 - examples, 157
 - key files and directories, 157
 - overview, 155–156
 - reducing verbosity, 158
 - Instant messaging data, definition, 6
 - Integrity checks. *See* File system checks; fsck command.
 - Interactive Spark applications. *See* Spark applications, interactive.
 - Intercepts, 390
 - Interrupts, monitoring, 575
 - I/O load, reducing, 624–625
 - I/O processes, MapReduce, 132–133
 - I/O statistics, monitoring, 573–574
 - iopath option, 645
 - iostat utility, 573–574
 - IP tables, disabling, 67
 - IPv6, disabling, 67
- J**
- jar command, 645
 - JAR files, displaying, 682
 - Java, installing, 69–70
 - Java classes, and Hadoop Streaming, 140
 - Java Database Connectivity (JDBC) server,
 - Spark applications, 191–192
 - Java heap, JVM garbage collection. *See also* Troubleshooting JVM garbage collection.
 - generations, 732–733
 - old generations, 732–733
 - overview, 732–733
 - permanent generations, 732–733
 - sizing, 733
 - young generations, 732–733
 - Java requirements for installing pseudo-distributed clusters, 69–70
 - Java Virtual Machine (JVM)
 - configuring reuse, 623–624
 - heap size, configuring, 616–617
 - and HotSpot, 624
 - metrics, 577
 - off heap usage, 668
 - JAVA_HOME environment variable, 73
 - JBOD disks, single rack to multiple racks, 98
 - JDBC (Java Database Connectivity) server,
 - Spark applications, 191–192
 - JN (JournalNode) daemons
 - configuring HA (high availability), 336
 - large cluster guidelines, 101
 - Job counters, 649–650
 - Job history metadata, YARN, 54
 - job -info command, 471
 - job -kill command, 471
 - Job launchers, Oozie workflows, 449
 - Job logs, reviewing, 602–604
 - Job processing
 - configuring Hadoop clusters, 73–74
 - MapReduce, 133–135
 - Job queue status, checking, 533
 - Job queues. *See* Capacity Scheduler, queues; Fair Scheduler, queues.
 - Job stages, displaying, 682, 684
 - Job tokens, 501
 - Job types, Oozie workflows, 439

- JobHistoryServer
 - description, 54
 - large cluster guidelines, 101
 - as monitoring tool, 606–607
 - port, specifying, 113
 - starting, 88–89
 - job.properties file, 462
 - Jobs. *See also* Applications.
 - completed, monitoring, 684, 686
 - details, displaying with MapReduce, 137–139
 - IDs, troubleshooting, 736
 - information, viewing, 531
 - parallelism, 377–378
 - scheduling. *See* Capacity Scheduler; Fair Scheduler; Oozie.
 - Spark applications, definition, 180–181
 - Spark execution model, 692, 693
 - status, checking, 471
 - tracking from the command line, 686
 - YARN, 49
 - Jobs, failures
 - Oozie, 473
 - troubleshooting, 738–739
 - Jobs tab, 682, 683
 - job.xml file, 589
 - join control nodes, configuring, 456–457
 - Joining two databases, optimizing shuffle operations, 702
 - Joins, Pig jobs, 638
 - JournalNode (JN) daemons
 - configuring HA (high availability), 336
 - large cluster guidelines, 101
 - JSON files, 679
 - Jsvc libraries, 500
 - JVM (Java Virtual Machine)
 - configuring reuse, 623–624
 - heap size, configuring, 616–617
 - and HotSpot, 624
 - metrics, 577
 - off heap usage, 668
 - JVM garbage collection, Java heap. *See also* Troubleshooting JVM garbage collection.
 - generations, 732–733
 - old generations, 732–733
 - overview, 732–733
 - permanent generations, 732–733
 - sizing, 733
 - young generations, 732–733
- K**
- kadm5.acl file, 487
 - kadmin utility, 494, 502
 - kadmin.local utility, 502
 - Kafka
 - benefits of, 398–399
 - brokers, 400, 402
 - commit log abstraction, 399
 - consumers, 400, 403
 - description, 17, 398
 - functional description, 399–400
 - handling large volumes of data, 400
 - installing, 401
 - integrating with Hadoop and Storm, 404–406
 - key components, 400
 - as messaging solution, 400
 - producers, 400, 403–404
 - topics, 400, 403
 - Kafka clusters
 - brokers, starting the, 402
 - creating topics, 403
 - producers, starting, 403–404
 - setting up, 401–404
 - ZooKeeper services, setting up, 402
 - kdadmind daemon, 502
 - kdb5_util utility, 502
 - KDC (Key Distribution Center)
 - authentication, 480
 - encryption types supported, specifying, 489
 - Kerberos security, 482–483
 - TCP ports, specifying, 489
 - UDC ports, specifying, 489
 - kdc.conf file, 487, 489–490
 - kdc_ports parameter, 489
 - kdc_tcp_ports parameter, 489
 - kdestroy command, 503
 - Kerberized clusters, definition, 482
 - Kerberized clusters, managing
 - accessing the Kerberos database, 502
 - AD, integrating with Hadoop, 504–505
 - AD, setting up one-way trust, 503–504
 - adding principals, 502

- administration commands, 502–503. *See also specific commands.*
- authenticating users, 502
- changing passwords, 502
- defining SPNs, 503–504
- granting tickets, 502
- listing a user's ticket cache, 503
- performing HDFS operations, 502
- provisioning UPNs, 503–504
- remote administration, 502
- retrieving TGTs, 502
- setting up one-way trust, 503–505
- utilities and daemons, 502
- viewing tickets, 502
- Kerberized systems, 257
- Kerberos. *See also* Authorization.
 - AS (Authentication Service), 482–483
 - 256-byte encryption, enabling/disabling, 490
 - ACLs, specifying for UPNs, 490
 - administrative domain. *See* Kerberos, realms.
 - authenticating users and services, 501
 - authentication process, 480, 483–484
 - authentication server, 483
 - block access tokens, 501
 - central server. *See* KDC (Key Distribution Center).
 - delegation tokens, 501
 - description, 30, 480
 - determining user identities, 258–259
 - example, 490
 - job tokens, 501
 - KDC (Key Distribution Center), 480, 482–483
 - keytab file, 480
 - name origin, 482
 - overview, 482
 - TGTs (Ticket Granting Tickets), 480, 483
 - tickets, 483
 - tokens, 501
- Kerberos, authorization
 - configuring Kerberos, 487–489
 - configuring the KDC, 489–490
 - installing Kerberos, 486–487
 - kadm5.acl file, 487
 - kdc.conf file, 487, 489–490
 - krb5.conf file, 487–489
 - overview, 486
- Kerberos, passwords
 - changing, 502, 503
 - storage location, 483, 493–495. *See also* Keytab file.
 - weak, dictionary of, 490
- Kerberos, realms
 - definition, 483
 - one-way trust, 485–486, 503–505
 - service principal, 483
 - special principal, 485
 - SPNs (service principal names), 483
 - trusted relationships, 484–485
 - two-way trust, 485–486
 - UPNs (user principal names), 483, 490
 - user principal, 483
- Kerberos, securing a cluster
 - acceptance filters, 496
 - core-site.xml file, configuration parameters, 497–498
 - Hadoop configuration files, 497–500
 - HDFS-related configuration, 499–500
 - hdfs-site.xml file, configuration parameters, 499–500
 - LinuxContainerExecutor, configuring, 498
 - mapping SPNs, 495–497
 - overview, 495
 - securing DataNodes, 500
 - starting the cluster in secure mode, 500–501
 - translating SPNs to operating system names, 495–496
 - YARN-related configuration, 498
 - yarn-site.xml file, configuration parameters, 498
- Kerberos, setting up
 - creating a database, 491
 - deleting SPNs, 493
 - keytab file, 493
 - overview, 490–491
 - SPNs, 492–493
 - starting servers, 492
 - UPNs, 491
- Kerberos databases
 - accessing, 502
 - creating, 491

Kerberos trusts, 484–485
 Key Distribution Center (KDC). *See* KDC
 (Key Distribution Center).
 Keytab files, 480, 493
 KickStart, 63
 kill command, 472, 532–533
 kill control nodes, Oozie workflows, 438
 kill nodes, configuring, 458–459
 Killed jobs, monitoring with web UIs,
 601–602
 Killing a job, 471, 472
 Killing running applications, 532–533
 Killing Spark jobs, 740
 kinit command, 502–503
 kinit utility, 502
 klist command, 494, 503
 klist utility, 502
 KMS (Key Management Server), 521, 522
 Knox, description, 31
 kpasswd command, 503
 kpasswd utility, 502
 krb5.conf file, 487–489
 krb5kdc command, 492
 krb5kdc daemon, 502

L

L option, 310, 312
 Large datasets, handling, 37
 last-value parameter, 378–379
 launch_container.sh script, 589
 Lazy_Persist, storage policies, 237
 LDAP directories
 as Kerberos database, 491
 one-way trusts, 503
 permission checks, 507
 LDAP Synchronization Connector (LSC), 505
 Leaf queues
 Capacity Scheduler, 414
 Fair Scheduler, 428
 Lease recovery, 224–225
 limits.conf file, setting shell limits, 67–68
 Lineage information, 718
 Linux file and directory commands
 cd, 245
 change directory, 245
 copying files and directories, 245
 cp, HDFS analog, 245

HDFS command analogs, 245
 head, HDFS analog, 360
 listing files, 244–245
 ls, HDFS analog, 245
 ls, listing HDFS files, 247, 248
 moving files and directories, 245
 mv, HDFS analog, 245
 print working directory, 245
 pwd, 245
 sudo, 259
 use administrative privileges, 259
 Linux file limits, increasing, 64
 Linux file system 1 full, troubleshooting,
 726
 Linux kernel, modifying. *See* Installing
 pseudo-distributed clusters, modifying
 the Linux kernel.
 Linux servers
 cloning, 745–746
 monitoring. *See* Monitoring Linux servers.
 LinuxContainerExecutor, configuring, 498
 list command, 531–532
 list-corruptfileblocks option, 286
 list-databases command, 368
 Listing
 archival storage policies, 238
 relational databases, 368
 snapshots, 282
 tables in a database, 368
 list-tables command, 368
 Loading data
 bulk data. *See* DistCp.
 copying between clusters. *See* DistCp.
 Hadoop data transfer tools, 355–356. *See*
 also specific tools.
 from HDFS, with Spark, 164–165
 into Hive, 142–143
 messaging systems. *See* Kafka.
 from relational databases. *See* Spark; Sqoop.
 Spark SQL, 199–200
 streaming data. *See* Flume; Kafka; Storm.
 vast amounts of. *See* DistCp.
 Loading data from the command line
 cat command, 356–357
 copy and moving files to and from HDFS,
 358–360
 copyFromLocal command, 358

- copyToLocal command, 359
 - dumping a file's contents, 356–357
 - get command, 359–360
 - getMerge command, 360
 - head command, 360
 - mv command, 360
 - put command, 358
 - specifying files as URIs, 357
 - tail command, 360
 - test command, 357
 - testing for files, 357
 - viewing first and last portions of an HDFS file, 360
 - Local directories
 - out of free space, troubleshooting, 727–729
 - Spark on YARN, setting, 681
 - Local mode
 - Pig, 144
 - Spark, 158
 - LOCAL_DIRS environment variable, 681
 - locations option, 287
 - Log aggregation. *See also* Flume.
 - accessing log files, 595–597
 - configuring log retention, 594–595
 - default retention period, 592
 - error message, 534
 - HDFS storage location, 593–594
 - overview, 592–593
 - viewing application logs, 596–597, 740–741
 - Log directories out of free space,
 - troubleshooting, 727–729
 - Log files. *See also specific log files.*
 - accessing, 595–597
 - auditing, 519–520
 - deleting, 598
 - Hadoop audits, 519–520
 - Oozie, 473
 - rotating, 598
 - Log ingesting tool. *See* Flume.
 - Log level, setting, 598–599
 - Log retention
 - configuring, 594–595
 - default retention period, 592
 - Log4j log file, 519
 - log4j.properties file, configuring logging, 75, 584
 - log_aggregation.retain.seconds parameter, 112
 - Logging
 - configuring, 584
 - configuring Hadoop clusters, 75
 - HDFS staging directories, 587–588
 - Log4j log file, 519, 584
 - “replaying edit logs” message, 326
 - types of logs, 583–584
 - Logging, NodeManager
 - launching, 590
 - local directories, 588–592
 - map/reduce containers, creating, 590
 - Logging levels, 591–592
 - Logging-related parameters, configuring, 111–113
 - Logical NameNode ID, 337
 - Logical nameservice, 337
 - Logs
 - accessing, 583, 584
 - analysis scenario, 7
 - monitoring with web UIs, 686
 - reviewing, 533–534
 - reviewing with ResourceManager, 602–604
 - stderr, 583–584
 - stdout, 583–584
 - syslog, 583–584
 - types of, 583–584. *See also* Application logs; Daemon logs.
 - logs command, 533–534
 - lost+found directory, 288–289. *See also* Trash directory.
 - ls command
 - HDFS analog, 245
 - listing HDFS files, 247, 248
 - listing snapshots, 282, 283
 - LSC (LDAP Synchronization Connector), 505
 - lsSnapshottbleDir command, 282
 - LZO format, 290, 291
- ## M
- m parameter, 382
 - Machine learning algorithms, Spark, 155
 - Mahout, description, 17
 - Managing, archival storage policies, 239

- Managing HDFS
 - from the command line. *See* `dfsadmin` utility.
 - setting up HDFS in pseudo-distributed Hadoop clusters, 70–71
- Managing HDFS with `hdfs dfs` utility
 - (minus sign), in `dfs` subcommands, 245
 - accessing the HDFS shell, 245
 - `chgrp` command, 250–251
 - `chmod` command, 251
 - `chown` command, 250–251
 - displaying all commands, 245
 - emptying the trash directory, 250
 - `expunge` command, 250
 - `help` command, 245–247
 - Linux file and directory command analogs, 245
 - `mkdir` command, 249
 - overview, 245
 - R operation, 251
 - recursive changes, 251
 - `rm` command, 249–250
 - shell commands, types of, 245
 - `skipTrash` option, 250
 - `stat` command, 248–249
 - `stat` command *vs.* `ls` command, 248–249
- Managing HDFS with `hdfs dfs` utility, files and directories
 - confirming existence of, 248
 - contents, displaying, 245, 247
 - creating directories, 249
 - displaying information about, 247, 248
 - groups, changing, 250–251
 - listing, 245, 247–248
 - ownership, changing, 250–251
 - permissions, changing, 251
 - removing, 249–250
- Manual failover, 348–349, 545–546
- Map and reduce tasks, configuring
 - MapReduce memory, 615–616
- Map phase, tuning map tasks, 626, 628–630
- Map step, MapReduce, 130
- MapFiles, small files problem, 300
- `map` (function) transformation, 178
- `MAP_INPUT_RECORDS` counter, 650
- Map-only jobs, 652
- `MAP_OUTPUT_RECORDS` counter, 650
- Mapper tasks, YARN, 49
- Mappers
 - limiting, 656–658
 - minimizing output, 655
 - too many, 655–656
- Mapping, SPNs (service principal names), 495–497
- MapR, Hadoop distribution, 60
- MapReduce environment, configuring
 - Hadoop clusters, 80
- `mapred` user, setting up in pseudo-distributed Hadoop clusters, 70–71
- `mapred-env.sh` file, 79
- `mapred.reduce.slowstart.completed` parameter, 634
- `mapred-site.xml` file
 - configuring MapReduce, 82–83
 - configuring pseudo-distributed Hadoop clusters, 74
 - configuring the reducer initialization time, 634
 - enabling compression, 293–294
 - `mapred.reduce.slowstart.completed` parameter, 634
 - `mapreduce.map.memory.mb`, 615–616
 - `mapreduce.map.output.compress` parameter, 293–294
 - `mapreduce.map.output.compress.codec` parameter, 293–294
 - `mapreduce.map.sort.spill` parameter, 633
 - `mapreduce.map.sort.spill.percent` parameter, 629–630
 - `mapreduce.output.fileoutputformat.compress` parameter, 293–294
 - `mapreduce.output.fileoutputformat.compress.codec` parameter, 294
 - `mapreduce.reduce.input.buffer.percent` parameter, 633
 - `mapreduce.reducer.memory.mb` parameter, 615–616
 - `mapreduce.reduce.shuffle.input.buffer.percent` parameter, 633
 - `mapreduce.reduce.shuffle.memory.limit.percent` parameter, 633–634
 - `mapreduce.reduce.shuffle.merge.percent` parameter, 632, 634
 - `mapreduce.reduce.shuffle.parallelcopies` parameter, 633

- mapreduce.shuffle.transfer.buffer.size parameter, 633–634
- mapreduce.task.io.sort.factor parameter, 628–630
- mapreduce.task.io.sort.mb parameter, 628–630
- tuning MapReduce shuffle process, 632–634
- MapReduce
 - alternatives to, 25
 - configuring Hadoop clusters, 82–83
 - data compression, 133, 291–294
 - displaying job details, 137–139
 - distributed data processing, 24–25
 - drawbacks, 149
 - Hadoop ecosphere, 15
 - input/output directories, 137
 - inputs and outputs, 132
 - I/O processes, 132–133
 - job processing, 133–135
 - key concepts, 131–133
 - map phase, 7
 - map step, 130
 - performance tuning. *See* Optimizing MapReduce; Tuning map tasks; Tuning reduce tasks.
 - programming model, 130
 - reduce phase, 7
 - reduce step, 130
 - sample program, 135–136
 - tasks, reducing. *See* Tuning map tasks.
 - typical scenario for, 7
 - in a YARN-based cluster, 54–56
- MapReduce, Hadoop Streaming
 - definition, 139–140
 - functional description, 140
 - Java classes, 140
- MapReduce, WordCount program
 - description, 130
 - running, 136–137
 - sample program, 135–136
- Map/reduce containers, creating, 590
- MapReduce framework counters, 650–651
- MapReduce jobs, Oozie action nodes, 450–451
- MapReduce mode, Pig, 144
- MapReduce services, setting up in pseudo-distributed Hadoop clusters, 70–71
- mapreduce.jobhistory.address parameter, 113
- mapreduce.jobhistory.done-dir parameter, 114
- mapreduce.jobhistory.intermediate-done-dir parameter, 114
- mapreduce.jobhistory.webapp.address parameter, 114
- mapreduce.job.jvm.numtasks parameter, 624
- mapreduce.job.maps parameter, 656
- mapreduce.job.reduces parameter, 656
- mapreduce.job.running.map.limit parameter, 656
- mapreduce.job.running.reduce.limit parameter, 656
- mapreduce.job.speculative.minimum-allowed-tasks parameter, 622
- mapreduce.job.speculative.slowtaskthreshold parameter, 622
- mapreduce.job.speculative.speculative-cap-running-tasks parameter, 622
- mapreduce.job.speculative.speculative-cap-total-tasks parameter, 622
- mapreduce.job.tags property, 598
- mapreduce.map.cpu.vcores, 621
- mapreduce.map.java.opts parameter, 110–111
- mapreduce.map.log.level property, 591
- mapreduce.map.memory.mb, 615–616
- mapreduce.map.memory.mb parameter, 110
- mapreduce.map.output.compress parameter, 293–294
- mapreduce.map.output.compress.codec parameter, 293–294
- mapreduce.map.sort.spill parameter, 633
- mapreduce.map.sort.spill.percent parameter, 629–630
- mapreduce.map.speculative parameter, 622
- mapreduce.output.fileoutputformat.compress parameter, 293–294
- mapreduce.output.fileoutputformat.compress.codec parameter, 294
- mapreduce.reduce.cpu.vcores, 621
- mapreduce.reduce.input.buffer.percent parameter, 633
- mapreduce.reduce.java.opts parameter, 110–111
- mapreduce.reduce.log.level property, 592
- mapreduce.reduce.memory.mb parameter, 110

- mapreduce.reducer.memory.mb parameter, 615–616
- mapreduce.reduce.shuffle.input.buffer.percent parameter, 633
- mapreduce.reduce.shuffle.memory.limit.percent parameter, 633–634
- mapreduce.reduce.shuffle.merge.percent parameter, 632, 634
- mapreduce.reduce.shuffle.parallelcopies parameter, 633
- mapreduce.reduce.speculative parameter, 622
- mapreduce.shuffle.transfer.buffer.size parameter, 633–634
- mapreduce-site.xml file, 113–114
- mapreduce.task.io.sort.factor parameter, 625, 628–630
- mapreduce.task.io.sort.mb parameter, 628–630
- Master nodes
 - fully distributed clusters, 99–100
 - in Hadoop clusters, 36
 - HDFS, architecture, 38–39
 - planning for fully distributed clusters, 100–101
- Master nodes, configuring, 161
- Master processes, starting/stopping, 161
- master_key-type parameter, 490
- Master-master replication, setting up, 550–551
- Maximum capacity, Capacity Scheduler, 412
- max_life parameter, 490
- max_renewable_life parameter, 490
- Measuring performance. *See* Benchmarking clusters; Hadoop metrics.
- memChannel, 392
- meminfo command, 573
- Memory
 - choosing, 96–99
 - configuring, for MapReduce. *See* Configuring MapReduce, memory.
 - ratio of physical to virtual, 617
 - tuning, 689
 - virtual memory for map and reduce tasks, 617
- Memory, sizing
 - fully distributed clusters, single rack to multiple racks, 98
 - Spark executors, 672
- Memory channels, 392
- Memory related parameters, 109–111
- Memory usage
 - monitoring, 571, 572–573
 - optimizing shuffle operations, 696
 - page ins/outs, 571
 - Spark applications, monitoring, 684, 685
 - Spark executors, configuring, 671–672
 - Spark on YARN, configuring resource allocation, 660, 670–672
 - thrashing, 571
 - troubleshooting, 734
- Memory usage, Spark executors
 - allocating, 660
 - configuring, 671–672
 - finding current, 672
- MEMORY_AND_DISK storage level, 719
- MEMORY_AND_DISK_SER storage level, 711, 719
- MEMORY_ONLY storage level, 719
- MEMORY_ONLY_SER storage level, 711, 719
- merge command, 379
- Merge joins, 638
- Merge phase
 - tuning map tasks, 626
 - tuning reduce tasks, 630–632
- Merging files, small files problem, 303–304
- Mesos
 - running Spark applications, 189
- Mesos, Spark applications, 189
- Mesos clusters, running Spark, 155, 158, 161–162
- Message system. *See* Kafka.
- Metadata files, checkpointing, 36
- Metadata retention, specifying, 700
- metasave command, dfsadmin utility, 254
- Metastore, sharing, 372
- Metastore databases, backing up, 553
- Metrics for Hadoop. *See* Hadoop metrics.
- Metrics REST API, 684
- Microsoft SQL Server, 367
- MILLIS_MAPS counter, 650
- MILLIS_REDUCE counter, 650
- Minimum share preemption, Capacity Scheduler, 421–422
- Minor GC (garbage collection), 687

- Minus sign (-), in dfs subcommands, 245
- min.user.id parameter, 498
- Mkdir command, 249
- MLlib, 155
- Modeling. *See* Data modeling.
- Modifying fully distributed clusters
 - DataNode web interface, 120–121
 - gateway machines, 119
 - Hadoop web interfaces, 120
 - web interfaces, 119–121
 - YARN web interface, 121
- Modifying fully distributed clusters, HDFS
 - configuration. *See also* Creating fully distributed clusters; Installing pseudo-distributed clusters, modifying the Linux kernel; Planning fully distributed clusters.
 - block replication, setting, 107–108
 - block size, setting, 107
 - data directories, specifying, 108
 - dfs.block.size parameter, 107
 - dfs.datanode.data.dir parameter, 108
 - dfs.datanode.du.reserved parameter, 107
 - dfs.name.node.dir parameter, 108
 - dfs.permissions.superusergroup parameter, 108
 - dfs.replication parameter, 107–108
 - fsimage file location, specifying, 108
 - hdfs-site.xml file, 106–109
 - in a multihomed network, 124
 - non-HDFS storage size, setting, 107
 - super user group, specifying, 108
- Modifying fully distributed clusters,
 - MapReduce configuration
 - history file directory, specifying, 114
 - history files, managing, 114
 - JobHistoryServer port, specifying, 113
 - mapreduce.jobhistory.address parameter, 113
 - mapreduce.jobhistory.done-dir parameter, 114
 - mapreduce.jobhistory.intermediate-done-dir parameter, 114
 - mapreduce.jobhistory.webapp.address parameter, 114
 - mapreduce-site.xml file, 113–114
 - staging directory, specifying, 113
 - Web UI, setting, 114
 - yarn.app.mapreduce.am.staging_dir parameter, 113–114
- Modifying fully distributed clusters, ports
 - Hadoop clients, 124–126
 - for HDFS, 123–124
 - port numbers for Hadoop services, setting, 122–123
- Modifying fully distributed clusters, YARN configuration
 - log_aggregation.retain.seconds parameter, 112
 - logging related parameters, 111–113
 - mapreduce.map.java.opts parameter, 110–111
 - mapreduce.map.memory.mb parameter, 110
 - mapreduce.reduce.java.opts parameter, 110–111
 - mapreduce.reduce.memory.mb parameter, 110
 - memory related parameters, 109–111
 - in a multihomed network, 124
 - yarn.application.classpath parameter, 113
 - yarn.log.aggregation-enable parameter, 111–112
 - yarn.nodemanager.aux-services parameter, 109
 - yarn.nodemanager.aux-services.
 - mapreduce_shuffle-class parameter, 109
 - yarn.nodemanager.local-dirs parameter, 112
 - yarn.nodemanager.log-dirs parameter, 112
 - yarn.nodemanager.resource.cpu-vcores parameter, 110
 - yarn.nodemanager.resource.memory-mb parameter, 109–110
 - yarn-site.xml file, 109
 - yarn.xml file, 109
- Modifying the Linux kernel. *See* Installing pseudo-distributed clusters, modifying the Linux kernel.
- Monitoring. *See also* Hadoop metrics.
 - aggregating metrics. *See* Monitoring with Ganglia.
 - collecting metrics. *See* Monitoring with Ganglia.
 - decommissioning and recommissioning nodes, 539

- Monitoring (*continued*)
 - Fair Scheduler, 434
 - Hive, 609–610
 - Spark, 610
 - Spark applications, 193–194
 - tools for, 16
 - tracking metrics. *See* Monitoring with Ganglia.
 - Monitoring Linux servers
 - alerting tool, 582
 - bandwidth, 572
 - context switches, 571, 575
 - CPU usage, 570–573
 - disk storage, 571–572
 - interrupts, 575
 - I/O statistics, 573–574
 - memory usage, 571, 572–573
 - network utilization, 575–576
 - page faults, 572–573
 - processes, 572–573
 - read/write operations, 574–576
 - resource usage, 574–575
 - runnable processes, 571
 - running processes, 572
 - Monitoring Linux servers, tools for
 - dstat command, 576
 - free command, 573
 - iostat utility, 573–574
 - meminfo command, 573
 - Nagios, 582
 - ps command, 572
 - sar utility, 574–576
 - top command, 574–575
 - vmstat utility, 572–573
 - Monitoring with Ganglia
 - architecture, 580
 - gmetad daemon, 580
 - gmond daemon, 580
 - gweb process, 580
 - overview, 579
 - RRDtool, 580
 - Monitoring with Ganglia, setup
 - alerting and monitoring, 582
 - extracting configuration files, 581
 - gmetad daemon, configuring, 581
 - gmond daemon, configuring, 581
 - Hadoop metrics, 582
 - installing Ganglia, 580–581
 - Nagios, 582
 - Monitoring with web UIs
 - completed jobs, 604–606, 606–607
 - failed and killed jobs, 601–602
 - GC (garbage collection), 684, 685
 - HDFS storage usage, 608–609
 - JobHistoryServer, 606–607
 - as monitoring tool, 599–606
 - NameNode, 608–609
 - overview, 599
 - ResourceManager, 599–606
 - reviewing job logs, 602–604
 - running jobs, 604–606
 - Mountable file systems, 564–566
 - move option, 288–289
 - Mover tool, 240
 - moveToTrash() method, 278
 - Moving data
 - within a cluster, 361–363
 - between clusters, 361–363
 - into an HDFS. *See* Loading data.
 - out of an HDFS. *See* Sqoop, exporting data.
 - Moving data around, archival storage, 239–240
 - Moving files to and from HDFS, 358–360
 - Multihomed network, HDFS in, 562–563
 - mv command, 360
 - mv command, HDFS analog, 245
 - MySQL databases
 - configuring, 445, 548–549
 - HA (high availability), 549–551
 - large cluster guidelines, 102
 - master–master replication, setting up, 550–551
 - switching active/passive roles, 551
- ## N
- Nagios, 582
 - Name quotas
 - vs.* space quotas, 263
 - specifying, dfsadmin utility, 264
 - NameNode operations
 - block reports, 322
 - checkpoints, 319
 - DataNode interactions, 322–323
 - HDFS metadata, 319–321

- overview, 318
- startup process, 321
- NameNode operations, edit logs
 - definition, 318
 - merging with fsimage files. *See* Checkpointing.
 - overview, 320–321
- NameNode operations, fsimage files
 - copying, controlling transfer speed, 327
 - creating. *See* Checkpointing.
 - definition, 318
 - downloading, 320–321
 - importance of updating, 323–324
 - loss or corruption, 319
 - merging with edit logs. *See* Checkpointing.
 - overview, 320–321
 - viewing contents of, 321
- NameNode operations, heartbeats
 - definition, 322
 - frequency, configuring, 321
 - overview, 322
 - piggybacking, 322
 - stopped, 322
- NameNode operations, safe mode
 - automatic operations, 328–329
 - backup and recovery, 332–334
 - configuration information, getting, 333–334
 - enabling, 330–331
 - transitioning to open mode, 331–332
- NameNodes
 - backing up HDFS metadata, 552
 - Block Storage Service, 350
 - communication with DataNodes, 207–208
 - configuring file system, host, and port information, 81
 - crashes, troubleshooting, 737–738
 - data stored in, 43–44
 - description, 36, 43
 - federated, 349–350
 - forcing a manual failover, 546
 - function of, 39–40
 - HA (high availability), 546
 - in Hadoop clusters, 36
 - in HDFS architecture, 38–39
 - HDFS operations, 43–45. *See also* Secondary NameNodes.
 - health, checking, 349
 - large cluster guidelines, 101
 - metadata file location, specifying, 85
 - as monitoring tools, 608–609
 - moving to a different host, 545
 - namespace volumes, 350
 - outages. *See* HA (high availability).
 - relation to DataNodes, 44
 - restarting, 46
 - starting, 87–88
 - status, displaying, 349
 - transitioning from Standby to active, 546
 - updating, 254
 - URI, specifying, 85
 - in very large clusters. *See* Federated NameNodes.
- NameNodes, edit logs
 - extra, configuring, 326–327
 - “replaying edit logs” message, 326
- Namespace volumes, 350
- Narrow dependencies, 698–700
- Narrow transformations, 698
- nestedUserQueue rule, 430
- Netezza, 367
- Network considerations, single rack to multiple racks, 99
- Network firewall, turning off, 67
- Network parameters, Spark-related, 676
- Network utilization, monitoring, 575–576
- NFSv3 gateway, configuring, 566–567
- NIC bonding, 65
- “No credentials cache found” message, 494–495
- noatime for disk mounts, setting, 65
- node command, 533
- Node status, checking, 533
- NODE_LOCAL, data locality level, 715
- NodeManager
 - configuring MapReduce memory, 617–618
 - in Hadoop clusters, 37
 - mapreduce shuffle, implementing, 83–84
 - YARN, 49, 52
- NodeManager, logging
 - application logs, 585–586
 - launching, 590
 - local directories, 588–592
 - map/reduce containers, creating, 590
- NodeManager, starting, 88–89

- NodeManager failures, troubleshooting, 738
 - NodeManager log file, 519–520
 - NodeManager services, large cluster guidelines, 101–102
 - Nodes
 - listing, 533
 - for planning fully distributed clusters, choosing, 94
 - removing from a cluster. *See* Nodes, decommissioning and recommissioning.
 - Nodes, decommissioning and recommissioning
 - adding DataNodes, 540–541
 - adding NodeManagers, 540–541
 - decommissioning a NodeManager, 538–539
 - decommissioning DataNodes, 537–538
 - including and excluding hosts, 536
 - monitoring, 539
 - overview, 535
 - recommissioning nodes, 538–539
 - run time, 539
 - tuning the HDFS, 539–540
 - Node's used DFS percentage, 270–271
 - nodirtime for directory mounts, setting, 65
 - nofile attribute, 67–68
 - Non-HDFS storage size, setting, 107
 - NO_PREF, data locality level, 715
 - nproc attribute, 67–68
 - NTP, enabling, 65
 - num-executors command-line flag, 667
 - NUM_KILLED_MAPS counter, 649
 - NUM_KILLED_REDUCEs counter, 649
 - num-mappers parameter, 382
- O**
- Objects, Sentry authorization, 513
 - ODBC server, Spark applications, 191–192
 - OEL (Oracle Enterprise Linux), installing, 745
 - OIV (Offline Image Viewer), 321
 - ok control nodes, configuring, 458–459
 - Old Generation garbage collection, 687
 - Old generations, JVM garbage collection, 732–733
 - ONE_SSD, storage policies, 237
 - One-way trust
 - on Kerberized clusters, 503–505
 - Kerberos realms, 485–486, 503–505
 - Oozie. *See also* Capacity Scheduler; Fair Scheduler.
 - bundles, 473
 - configuration files, 473
 - configuring, 560
 - cron scheduling, 474
 - Dashboard, 555
 - description, 17
 - job failures, 473
 - log files, 473
 - SLAs (service level agreements), 474–475
 - troubleshooting, 473–474
 - Oozie, deploying
 - configuring Hadoop for Oozie, 444–446
 - installing Oozie, 441–442
 - installing Oozie server, 442–444
 - MySQL database, configuring, 445
 - overview, 441–442
 - workflow jobs, 463
 - Oozie, managing and administering
 - checking coordinator status, 472
 - checking job status, 471
 - checking Oozie status, 472
 - common commands, 471
 - dry runs, 472
 - killing a job, 471, 472
 - overview, 470–471
 - resuming a suspended job, 472
 - running Pig jobs through HTTP, 472
 - SLA event records, getting, 472
 - suspending running jobs, 472–473
 - validating a workflow.xml file, 472
 - validating XML schemas, 472
 - Oozie architecture, client
 - description, 440
 - installing, 445–446
 - Oozie architecture, database, 440–441
 - Oozie architecture, server
 - description, 439–440
 - installing, 442–444
 - starting/stopping, 444–445
 - Oozie coordinators
 - data based, 467–469
 - overview, 464–465
 - submitting from the command line, 469–470
 - time based, 465–467, 469. *See also* cron scheduling.

- Oozie status, checking, 472
- oozie utility, 438
- Oozie workflow jobs
 - configuring, 460–461
 - deploying, 463
 - dynamic workflows, 463–464
 - job.properties file, 462
 - properties, specifying, 461–463
 - running, 461–464
- Oozie workflows
 - bundle jobs, 439
 - case statements, 460
 - control nodes, 446–447
 - control nodes, configuring, 456–460
 - coordinator jobs, 438–439
 - decision control nodes, 438, 446–447
 - defining, 447–449
 - description, 437–439, 446
 - end control nodes, 438, 446–447, 448, 456
 - example, 448
 - fork actions, 448
 - fork control nodes, 438, 446–447, 448, 456–457
 - if-then-else actions. *See* decision control nodes.
 - job launchers, 449
 - job types, 439
 - join control nodes, 438, 446–447, 448, 456–457
 - kill control nodes, 438
 - ok control nodes, configuring, 458–459
 - start control nodes, 438, 446–447, 448, 456
 - workflow jobs, 439
 - workflow.xml file, 447–449
- Oozie workflows, action nodes
 - configuring, 449–454
 - description, 438, 448
 - fs actions, 454
 - for Hive jobs, 451–452
 - for MapReduce jobs, 450–451
 - for Pig, 452–453
 - Shell actions, 453
 - types of, 449–450. *See also specific types.*
- Oozie workflows, creating
 - control nodes, configuring, 456–460
 - decision, configuring, 459–460
 - end control nodes, configuring, 456
 - error nodes, configuring, 458–459
 - fork control nodes, configuring, 456–457
 - join control nodes, configuring, 456–457
 - kill nodes, configuring, 458–459
 - ok control nodes, configuring, 458–459
 - overview, 454–455
 - start control nodes, configuring, 456
- Operating Hadoop clusters
 - DataNode services, starting, 87–88
 - formatting the HDFS, 86–87
 - HDFS services, starting, 87–88
 - JobHistoryServer, starting, 88–89
 - NameNode services, starting, 87–88
 - NodeManager, starting, 88–89
 - ResourceManager, starting, 88–89
 - Secondary NameNode services, starting, 87–88
 - services, shutting down, 90
 - services, starting, 87–89
 - setting environment variables, 87
 - YARN services, starting, 88–89
- Operating system. *See* YARN (Yet Another Resource Negotiator).
- Optimization, administrator duties, 20. *See also* Performance; Tuning.
- Optimized row columnar (ORC) files, 297, 300–301, 303, 681
- Optimizing
 - JVM garbage collection, 733–734
 - Spark execution model, 692–694
- Optimizing Hive jobs
 - bucketing, 635
 - built-in capabilities, 636
 - cost-based optimization, 636
 - ORCFILE format for Hive tables, 636
 - overview, 635
 - parallel execution, 635
 - partitioning, 635
 - vectorization, 636–637
- Optimizing MapReduce. *See also* Tuning
 - map tasks; Tuning reduce tasks.
 - balancing work among reducers, 655
 - combiners, 652–654
 - data compression, 654–655
 - limiting mappers or reducers, 656–658
 - map-only jobs, 652
 - minimizing mapper output, 655

- Optimizing MapReduce (*continued*)
 - Partitioners, 654
 - reduce phase, 633–634
 - reducer initialization time, 634
 - shuffle process, 632–634
 - sort, 633
 - spill process, 633
 - too many mappers or reducers, 655–656
 - Optimizing Pig jobs
 - merge joins, 638
 - replicated joins, 638
 - rules for, 637
 - setting parallelism, 637
 - skewed joins, 638
 - specialized joins, 638
 - Optimizing shuffle operations. *See also*
 - Optimizing Spark applications.
 - accumulators, 702–703
 - aggregateByKey operator, 702
 - all-to-all operations, 695
 - avoiding a shuffle, 702–703, 709–710
 - broadcast variables, 702–703
 - cogroup operator, 702
 - compression operations, 697–698
 - configuring shuffle parameters, 697
 - disk I/O, 696–697
 - example, 695–696
 - GC (garbage collection), 697
 - groupByKey operator, 700–702
 - joining two databases, 702
 - memory usage, 696
 - metadata retention, specifying, 700
 - minimizing shuffle operations, 699
 - narrow transformations, 698
 - reduceByKey operator, 694–695, 700–702
 - stage boundaries, 699
 - triggering a shuffle, 698–700
 - wide transformations, 698
 - Optimizing Spark applications. *See also*
 - Optimizing shuffle operations; Tuning
 - Spark streaming applications.
 - caching data, 717–723
 - compression, 711–712
 - data serialization, 710–711
 - number of tasks, 703–710
 - parallelism, 703–710
 - partitioning, 703–710
 - Spark execution model, 692–694
 - SQL query optimizer, 712–716
 - Options file, Sqoop, 371
 - Oracle Enterprise Linux (OEL), installing, 745
 - Oracle products. *See specific products.*
 - ORC (optimized row columnar) files, 297, 300–301, 303, 681
 - ORCFILE format for Hive tables, 636
 - OS page caching, 228
 - Out of memory errors, troubleshooting, 734–735
 - OutputFormat, 164
 - overwrite option, 364–365
- P**
- Package manager for Red Hat, SUSE and Fedora Linux, 63
 - Page faults, monitoring, 572–573
 - Page ins/outs, monitoring, 571
 - Pair RDDs, 179
 - Parallel execution, Hive jobs, 635
 - parallel option, 637
 - Parallelism
 - optimizing, 703–710
 - in Pig jobs, 637
 - in Spark applications, 703–705
 - Parallelizing
 - data ingestion, 688
 - data processing, 689
 - Parquet files, 290, 680
 - Partitioners, 654
 - Partitioning
 - Hive jobs, 635
 - optimizing, 703–710
 - Partitioning in Spark applications
 - avoiding a shuffle, 709–710
 - coalesce operator, 708–709
 - HashPartitioner partitions, 709
 - by key code, 709
 - overview, 703–704
 - by range, 709
 - RangePartitioner partitions, 709
 - repartition operator, 708
 - repartitioning, 708
 - types of partitions, 709

- Partitioning in Spark applications, number of partitions
 - increasing, 707–708
 - in an RDD, changing, 708–709
 - setting default for, 706–707
- Partitions, 708–709
- Passwordless connection, pseudo-distributed
 - Hadoop clusters, 68–69
- Passwords, Kerberos
 - changing, 502, 503
 - storage location, 483, 493–495
 - weak, dictionary of, 490
- Passwords, Sqoop, 372
- pdsh utility, 63, 102–106
- Performance. *See also* Optimization; Tuning.
 - administrator duties, 20
 - checkpointing, 327
 - choosing a file format, 297
 - measuring. *See* Benchmarking clusters; Hadoop metrics.
 - troubleshooting, 682–684
- Performance, improving
 - configuring JVM reuse, 623–624
 - deprecated parameters, 623
 - reducing I/O load, 624–625
 - speculative execution, 621–624
- Permanent generations, JVM garbage collection, 732–733
- Permission checking, enabling/disabling, 255
- “Permission denied” errors, 256
- Permissions
 - ACLs (access control lists), 507–509
 - authorization, 507
 - changing, 507
 - changing file permissions, 507
 - checking, 507
 - checking permissions, 507
 - configuring, 506
 - configuring super users, 506
 - extended attributes, 509–510
 - Hive, 514–515
 - overview, 505–506
 - raw namespace, 509–510
 - security namespace, 509–510
 - simple security mode, 505–506
 - system namespace, 509–510
 - user namespace, 509–510
- Permissions for files and directories. *See* HDFS permissions.
- persist() method, 719–721
- Persistence, RDD, 179
- PHYSICAL_MEMORY_BYTES counter, 650
- Pig
 - description, 17, 26
 - example, 145
 - execution modes, 144
 - local mode, 144
 - MapReduce mode, 144
 - Oozie action nodes, 452–453
 - overview, 144
- Pig jobs
 - optimizing. *See* Optimizing Pig jobs.
 - running through HTTP, 472
- Pig Latin, 144
- Piggybacking heartbeats, 322
- Pipe symbol (|)
 - pipng data into HDFS files, 360
 - reviewing files, 359
- Pipeline recovery, 226–227
- Pipeline setup stage, 227
- Pipelining, 693
- Pivotal HD, Hadoop distribution, 60
- Planning fully distributed clusters
 - choosing nodes, 94
 - form factors, 94
 - general considerations, 92–94
 - master nodes, 99–100
 - overview, 92
 - typical architecture, 93
- Planning fully distributed clusters, servers
 - blade servers, 94, 97
 - commodity servers, 94
 - custom designed rack servers, 97–98
 - DataNodes, 100–101
 - Master Nodes, 100–101
 - rack servers, 94
 - sizing, 100–101
- Planning fully distributed clusters, single rack
 - to multiple racks
 - amount of data storage, 96
 - architecture, 95–96
 - blade servers, 97
 - CPU, choosing, 96–99

- Planning fully distributed clusters, single rack
 - to multiple racks (*continued*)
 - custom designed rack servers, 97–98
 - disk configuration, 97–98
 - disk failure, risk of, 98
 - disk sizing, 97–98
 - extending clusters, 101
 - growth patterns, 96
 - JBOD disks, 98
 - key principles, 96–99
 - large cluster guidelines, 101–102
 - memory, choosing, 96–99
 - memory, sizing, 98
 - network considerations, 99
 - RAID disks, 98
 - sizing the cluster, 96
 - storage, choosing, 96–99
 - type of workload, 96
 - virtualization, 97
 - Policies, Sentry authorization, 513, 517
 - Policy administration examples, 517–518
 - Policy engine, Sentry, 513
 - Policy providers, Sentry, 513
 - Port numbers for Hadoop services, setting, 122–123
 - Ports, modifying in fully distributed clusters
 - Hadoop clients, 124–126
 - for HDFS, 123–124
 - port numbers for Hadoop services, setting, 122–123
 - POST operation, 308
 - PostgreSQL, 445
 - Precedence among configuration files, 76–78
 - Preempting applications, Capacity Scheduler, 421–422
 - Preemption, Fair Scheduler, 409
 - primaryGroup rule, 430
 - Principals, adding to Kerberized clusters, 502
 - printTopology command, 211
 - Priorities, Fair Scheduler, 409
 - Privilege models, Sentry authorization, 514
 - Privileges, Sentry authorization, 513, 514
 - Processes, monitoring, 572–573
 - Processing engines, Hadoop 2 *vs.* Hadoop 1, 23
 - Processing layer. *See* YARN (Yet Another Resource Negotiator).
 - PROCESS_LOCAL, data locality level, 715
 - Producers, Kafka, 400, 403–404
 - Programming model, MapReduce, 130
 - Properties
 - core Hadoop properties, 81
 - Oozie workflow jobs, 461–463
 - precedence, 662
 - Spark, viewing, 713–714
 - Protocol Buffers, 200, 712
 - ps command, 572
 - Pseudo-distributed systems, 19
 - ptopax option, 283
 - Puppet, 569
 - put command, 358
 - PUT operation, 308, 312
 - pwd command, 245
 - pyspark command, 661
 - Python
 - memory resources, 667
 - in the Oozie shell, 453
 - sample programs, 157
 - vs.* Scala, 170–171, 713
 - vs.* Spark, 170–171
 - storage levels, 721
 - Python objects in an RDD, 157
 - Python program, submitting, 187
- Q**
- QJM (Quorum Journal Manager), 335
 - quasiquotes, 713
 - Query plans, 686
 - Querying data
 - with Hive, 143
 - Spark SQL, 200
 - queue command, 533
 - queue element, 415
 - Queues. *See* Capacity Scheduler, queues; Capacity Scheduler, subqueues; Fair Scheduler, queues.
 - Quota violation state, 266
- R**
- R operation, 251
 - r (read) permission, 255–256, 506
 - Rack awareness
 - cluster redundancy, 209–210
 - configuring, 210
 - dfsadmin utility, 211–212

- distributing data replicas, 211
- finding cluster rack information, 210–211, 212
- fsck command, 211
- overview, 209–210
- printTopology command, 211
- report command, 212
- ResourceManager, 209
- topology.py script, 210
- Rack servers, custom designed, 97–98
- RACK_LOCAL, data locality level, 715
- Racks. *See* Hardware racks.
- RAID disks, 98
- RAM_DISK storage type, 237
- RangePartitioner partitions, 709
- Ranger, description, 31
- Raw namespace, 509–510
- RBW (Replica Being Written) replica state, 216
- RCFile format, 290, 300–310
- RDBMS (relational database management system)
 - listing, 368
 - loading unto HDFS. *See* Spark; Sqoop.
 - moving data to and from. *See* Sqoop.
 - querying. *See* Hive.
- RDD (resilient distributed dataset). *See also* Spark applications.
 - caching. *See* Caching RDD data.
 - collect(0) operation, 720
 - contents of, 174
 - creating DataFrames, 200–201
 - Double RDDs, 179
 - narrow dependencies, 698–700
 - number of partitions, changing, 708–709
 - operations, 176–178
 - overview, 173
 - Pair RDDs, 179
 - persistence, 179
 - Spark execution model, 693
 - wide dependencies, 698–700
- RDD (resilient distributed dataset), actions
 - count() operation, 177
 - counting number of elements, 177
 - definition, 170
 - first operation, 177
 - returning arrays of elements, 177
 - returning largest element, 177
 - saveAsTextFile() operation, 177
 - saving as a text file, 177
 - take(n), 177
 - top() operation, 177
- RDD (resilient distributed dataset), creating
 - from existing RDDs, 170
 - with parallelization, 174
 - subsets of RDDs, 178
 - from a text file, 175
 - with transformations, 178
- RDD (resilient distributed dataset), transformations
 - creating new RDDs, 178
 - creating subsets of RDDs, 178
 - definition, 170
 - distinct, 178
 - filter(function), 178
 - filtering out duplicates, 178
 - flatMap, 178
 - map(function), 178
 - sample, 178
 - sortBy, 178
 - sorting, 178
- rdd.getNumPartitions() function, 709
- rdd.partitions.size() function, 709
- Read (r) permission, 255–256, 506
- Read phase, tuning map tasks, 626
- Read tests, benchmarking clusters, 640
- Reading HDFS data, 219–220
- Read-only default configuration, Hadoop clusters, 74
- Read/write operations, monitoring, 574–576
- Realms. *See* Kerberos, realms.
- Rebalancing HDFS data
 - adjusting balancer bandwidth, 273–274
 - amount of data moved, 270
 - average DFS used percentage, 270–271
 - balancer command, 269
 - balancer tool, 267, 268–271, 547
 - balancing storage on DataNodes, 547–548
 - current balance, checking, 547
 - dfsadmin command, 271–272
 - iterative movement of blocks, 272
 - making the balancer run faster, 273–274
 - node's used DFS percentage, 270–271
 - overview, 267–268

- Rebalancing HDFS data (*continued*)
 - run time, 270
 - setBalancerBandwidth option, 273–274
 - start-balancer.sh command, 268–271
 - threshold, setting, 269–270
 - tools for, 267, 271–272
 - unbalanced data, description, 48
 - unbalanced data, reasons for, 268
 - when to run the balancer, 272
- Recommissioning nodes. *See* Nodes, decommissioning and recommissioning.
- Recoverability, distributed computing requirements, 33
- Recovering deleted files, from snapshots, 283–284
- Recovery process. *See also* Backup and recovery; Fault tolerance.
 - block recovery, 226
 - close stage, 227
 - data streaming stage, 227
 - disaster recovery, 20. *See also* Backup and recovery; Snapshots.
 - GS (Generation Stamp), 224
 - lease recovery, 224–225
 - pipeline recovery, 226–227
 - pipeline setup stage, 227
 - RUR (Replica Under Recovery) replica state, 216
 - UNDER_RECOVERY block state, 218–219
 - work preserving recovery, 739
- Recursive changes, 251
- Red Hat, package manager for, 63
- Red Hat Enterprise Linux RPM software packages, 63
- Red Hat products. *See specific products.*
- Reduce phase
 - optimizing MapReduce, 633–634
 - tuning reduce tasks, 630–632
- Reduce step, MapReduce, 130
- Reduce tasks, YARN, 49
- reduceByKey operator, 694–695, 700–702
- Reducer initialization time, optimizing
 - MapReduce, 634
- Reducers
 - balancing work among, 655
 - limiting, 656–658
 - too many, 655–656
- reducers.bytes.per.reducer property, 637
- reducers.max property, 637
- REDUCE_SHUFFLE_BYTES counter, 650
- Redundancy of data
 - cluster computing, 12–13
 - Hadoop architecture, 34
- refreshNodes command, 254, 535
- refreshQueues command, 424
- reject rule, 430
- Relational databases. *See* RDBMS (relational database management system).
- Remote administration, Kerberized clusters, 502
- Removing, space quotas, 265
- Renaming directories, 283
- repartition operator, 708
- Repartitioning, 708
- “Replaying edit logs” message, 326
- Replica Being Written (RBW) replica state, 216
- Replica states, 216
- Replica Under Recovery (RUR) replica state, 216
- Replicas Waiting to be Recovered (RWR) replica state, 216
- Replicated joins, 638
- Replication, troubleshooting, 730
- report command
 - “Access denied...” error, 256–257
 - description, 252
 - displaying HDFS storage, 263
 - displaying rack information, 212
 - examining HDFS cluster status, 252–254
 - sample output, 253–254
- Reporting, data science component, 11
- Representational State Transfer (REST) API. *See* WebHDFS.
- Resilient distributed dataset (RDD). *See* RDD (resilient distributed dataset).
- Resource allocation
 - ApplicationMaster, 53–56
 - Hadoop 2, 407–410. *See also* Resource schedulers.
 - Hadoop 2 *vs.* Hadoop 1, 24
 - Hadoop clusters, 36
 - limits, 661–663

- managing cluster workloads, 408
 - overview, 660–661
 - YARN memory. *See* Allocating YARN memory.
 - Resource management
 - Hadoop ecosystem, 15
 - YARN, 50–56
 - Resource schedulers
 - default, 408
 - list of, 409. *See also specific schedulers.*
 - Resource usage, monitoring, 574–575
 - ResourceManager
 - in Hadoop clusters, 36
 - high availability. *See* HA (high availability), ResourceManager.
 - large cluster guidelines, 101
 - rack awareness, 209
 - Restart feature, 543
 - YARN, 49
 - ResourceManager, starting, 88–89
 - ResourceManager crashes, troubleshooting, 738
 - ResourceManager log file, 519–520
 - REST (Representational State Transfer) API. *See* WebHDFS.
 - Restart feature, 543
 - Restoring deleted files, from the trash directory, 278
 - resume command, 472
 - Resuming a suspended job, 472
 - Retention duration for application logs, setting, 592
 - Retrying jobs after a failure, 738–739
 - rm command, 249–250
 - Role-based authorization. *See* Authorization, Sentry.
 - Roles, Sentry authorization, 514, 518–519
 - Rotating log files, 598
 - RPC metrics, 577
 - rpm, 63
 - RRDtool, 580
 - Rumen, benchmarking clusters, 643–644
 - Runnable processes, monitoring, 571
 - Running
 - Oozie workflow jobs, 461–464
 - Pig jobs through HTTP, 472
 - processes, monitoring, 572
 - Running jobs
 - displaying, 682
 - monitoring, 604–606
 - RUR (Replica Under Recovery) replica state, 216
 - RWR (Replicas Waiting to be Recovered) replica state, 216
- S**
- S3 (Amazon Simple Storage Service), 165
 - S3 (s3a) file system, 244
 - S3DistCp, 307
 - Safe mode. *See* NameNode operations, safe mode.
 - Safeguarding data. *See* Snapshots; Trash directory.
 - safemode wait command, 330–331
 - Sample transformation, 178
 - sar utility, 574–576
 - SASL (Simple Authentication and Security Layer), 477, 501
 - Scala language
 - benefits of, 21, 170–171
 - building Spark applications, 186
 - default persistence level, 721
 - examples, 157
 - running Spark applications on Mesos, 189
 - in the Spark shell, 182–183
 - Scala objects, in RDD files, 170
 - Scalability
 - distributed computing requirements, 33
 - issues with traditional database systems, 9
 - Scale up architecture *vs.* scale out, 8
 - Scaling trace runtime, benchmarking clusters, 643–644
 - Scheduler, YARN, 51
 - Schedulers. *See* Resource schedulers.
 - Scheduling jobs
 - administration, 29–30
 - Hadoop ecosystem, 15
 - Scheduling policies, configuring, 431
 - scp, 63
 - Scripting, in Pig Latin. *See* Pig.
 - Scripts, for starting and stopping a cluster, 116–117
 - Secondary NameNodes
 - checkpointing, 324, 328–329

- Secondary NameNodes (*continued*)
 - HA (high availability), 46–47
 - in Hadoop clusters, 36, 88
 - in HDFS, 46–47
 - restarting NameNodes, 46
 - starting, 87–88
- secondaryGroupExistingQueue rule, 430
- Securing data, administration, 30–31
- Security
 - default, 30
 - determining access to cluster data. *See* Authorization.
 - Fair Scheduler, 432
 - Knox, 524–525
 - overview, 478–480
 - Ranger, 525
 - roles in a cluster, 479–480
 - tracking cluster activity. *See* Auditing.
 - verifying user identities. *See* Authentication.
- Security namespace, 509–510
- SecurityAuth-hdfs.audit log file, 519
- security.job.client.protocol.acl property, 511
- security.job.task.protocol.acl property, 511
- select() operation, 200
- SELinux, disabling, 66
- Sending, files, 63
- Sensor data, definition, 6
- Sentiment data, definition, 6
- Sentry policy file, 514
- Sentry service. *See also* Authentication.
 - description, 30, 514
 - role-based authorization. *See* Authorization, Sentry.
- sentry.metastore.service.users property, 516
- SequenceFiles, 679
 - description, 299–300
 - HDFS, 42
 - small files problem, 299–300
 - structured format, 290
- SerDe (serialization deserialization), 295
- Server, Oozie architecture
 - description, 439–440
 - installing, 442–444
 - starting/stopping, 444–445
- Server BIOS settings, checking, 65
- Server log data, 6. *See also* Logs.
- Service level agreements (SLAs)
 - event records, getting, 472
 - Oozie, 474–475
- Service level authorization. *See* Authorization, service level.
- Service principal, Kerberos realms, 483
- Service principal names (SPNs). *See* SPNs (service principal names).
- Service tickets, 483. *See also* TGTs (Ticket Granting Tickets).
- Services, starting/shutting down, 87–90
- set default_parallel option, 637
- setBalancerBandwidth option, 273–274
- setQuota command, 264, 346
- setSpaceQuota command, 265, 346
- setStoragePolicy command, 239
- Shell actions, Oozie action nodes, 453
- Shell commands, types of, 245
- Shell limits, setting, 67–68
- shell method, 340–341
- Short-circuit local reads, 231–232, 563–564
- show() operation, 200
- Shuffle boundaries, 693
- Shuffle phase, tuning reduce tasks, 630–632
- Shuffle process, optimizing MapReduce, 632–634
- Shuffling data, 693
- Shutting down. *See* Starting up and shutting down.
- Simple Authentication and Security Layer (SASL), 477, 501
- Simple security mode, 505–506
- Simplicity, Spark, 152
- Sink processors, 390
- Sinks, 389–390, 395, 578–579
- Site-specific configuration, configuring
 - Hadoop clusters, 74
- Skewed joins, 638
- skipTrash option, 250, 280
- sla command, 472
- SLAs (service level agreements)
 - event records, getting, 472
 - Oozie, 474–475
- Small files
 - consolidating batch files, 307
 - managing, 304–306

- performance impact, 307
- SequenceFile key/pairs, 307
- Small files problem
 - federated NameNode architecture, 304
 - MapFiles, 300. *See also* SequenceFiles.
 - merging files, 303–304, 306–307
 - overcoming, 303–304
 - overview, 303–304
 - SequenceFiles, 299–300. *See also* MapFiles.
- Snappy format, 290, 291
- snapshotDiff command, 282–283
- Snapshots. *See also* fsimage file.
 - as backups, 284
 - copying files from, 283
 - creating/deleting, 281–282
 - enabling/disabling, 281
 - listing, 282
 - overview, 280–281
 - recovering deleted files, 283–284
 - renaming directories, 283
 - snapshottable directories, removing, 283
 - viewing differences between, 282–283
- soft limit settings, 67–68
- Solr, alternative to MapReduce, 25
- Sort performance, tuning reduce tasks, 632
- Sort process, optimizing MapReduce, 633
- sortBy transformation, 178
- Sorting, RDDs, 178
- Sources, 389–390, 395, 578–579
- Space, storage. *See* HDFS storage.
- Space quotas. *See* HDFS storage, space quotas.
- Spark
 - accessing text files, 164–165
 - alternative to MapReduce, 25
 - cluster mode, 158–159
 - clusters for, 158–159. *See also specific clusters.*
 - data access, 164–166
 - data compression, 295
 - general framework, 152
 - graphs, 155
 - and Hadoop, 153
 - installing. *See* Installing Spark.
 - loading data from a relational database, 166
 - loading data from HDFS, 164–165
 - local mode, 158
 - machine learning algorithms, 155
 - MapReduce drawbacks, 149
 - on Mesos clusters, 155, 158, 161–162
 - overview, 147–149
 - run modes, 158–159
 - standalone clusters, 158
 - streaming data, 155
 - uses for, 152
- Spark API, entry point to, 183
- Spark applications. *See also* RDD (resilient distributed dataset).
 - architecture, 179–181
 - building, 186
 - client mode, 189, 190
 - cluster managers, 180
 - cluster mode, 189, 190–191
 - components of, 180–181
 - configuration properties, 192–193
 - definition, 180
 - driver program, 180
 - executing, 187–189
 - executors, 181
 - JDBC server, 191–192
 - job, definition, 180
 - jobs, description, 181
 - local file storage, specifying, 193
 - memory allocation, specifying, 193
 - on Mesos, 189
 - ODBC server, 191–192
 - running on Mesos, 189
 - running with spark-submit script, 193–194
 - shared variables, 173
 - Spark Shell, 181–185
 - spark.executor.memory property, 193
 - sparklocal.dir property, 193
 - spark-submit script, 187–189
 - stage, definition, 180
 - stages, description, 181
 - streaming, tuning. *See* Tuning Spark streaming applications.
 - task, definition, 180
 - tasks, description, 181
 - worker processes, 180
 - on YARN, 189
- Spark applications, configuring
 - configuration properties, 192–193
 - local file storage, specifying, 193
 - memory allocation, specifying, 193
 - spark.executor.memory property, 193

- Spark applications, configuring (*continued*)
 - sparklocal.dir property, 193
 - with spark-submit script, 193–194
- Spark applications, interactive
 - execution, overview, 185
 - overview, 181
 - Spark Shell, 181–185
- Spark applications, monitoring with web UIs
 - cache status, displaying, 684
 - completed jobs, displaying, 684, 686
 - configuration parameters, displaying, 682
 - DAG page, 684
 - debugging, 686
 - default port, 682
 - Environment tab, 682
 - garbage collection, 684, 685
 - getting logs, 686
 - JAR files used, displaying, 682
 - job stages, displaying, 682, 684
 - Jobs tab, 682, 683
 - memory usage, 684, 685
 - Metrics REST API, 684
 - query plans, 686
 - running jobs, displaying, 682
 - Spark history server, 684, 686
 - Stages tab, 682, 684
 - Storage tab, 684
 - task durations, 684, 685
 - Task Metrics tab, 684, 685
 - tracking jobs from the command line, 686
 - troubleshooting performance, 682–684
 - viewing status of, 194
- Spark applications, running
 - client mode, 186–187
 - cluster mode, 186–187
 - in the standalone Spark cluster, 186–187
- Spark benefits
 - accessibility, 151–152
 - advanced execution engine, 150–151
 - compactness, 152
 - ease of use, 151–152
 - in-memory computation, 151
 - simplicity, 152
 - speed, 149–151
- Spark cluster managers
 - Mesos, 161–162
 - standalone cluster, 159–161
- Spark Core, 154
- Spark execution model
 - DAG (directed acyclic graph), 693
 - execution plan, 693
 - jobs, 692, 693
 - optimizing, 692–694
 - pipelining, 693
 - RDD (resilient distributed dataset), 693
 - shuffle boundaries, 693
 - shuffling data, 693
 - Spark applications, 692
 - stages, 693
 - tasks, 693–694
- Spark executors, configuring resource
 - allocation
 - broadcast variables, 672
 - dynamic allocation, 667
 - memory usage, 671–672
 - number of executors, 667
 - overview, 666–667
 - resources for the executors, 667–669
 - summary of, 669
 - tasks and executors, 669, 672–673
 - for workload types, 674
- Spark history server, 684, 686
- Spark jobs, killing, 740
- Spark jobs, troubleshooting
 - fault tolerance, 740
 - killing Spark jobs, 740
 - maximum attempts, specifying, 740
 - maximum failures per job, specifying, 740
 - maximum launch attempts, specifying, 740
 - task failures, 739
- Spark JVM garbage collection,
 - troubleshooting, 734
- Spark micro-batching, 196
- Spark on YARN, cluster managers
 - compatibility, 158
 - overview, 162–163
 - setting up Spark, 163
 - Spark/YARN interaction, 163
 - standalone scheduler, 155
 - YARN *vs.* standalone cluster manager, 163
- Spark on YARN, configuring resource
 - allocation
 - cluster mode *vs.* client mode, 674–676
 - CPU, 660
 - for drivers, duties, 663–664
 - for drivers, in client mode, 664–665

- for drivers, in cluster mode, 665–666
- dynamic allocation, enabling, 677–678
- dynamic allocation *vs.* static, 676–678
- for executors. *See* Spark executors,
 - configuring resource allocation.
- memory usage, 660, 670–672
- property precedence, 662
- resource allocation limits, 661–663
- resource allocation overview, 660–661
- setting configuration properties, 662–663
- setting local directories, 681
- Spark-related network parameters, 676
- yarn-client mode, 662
- yarn-cluster mode, 662
- Spark programming
 - accumulators, 172
 - anonymous functions, 171
 - broadcast variables, 172
 - chaining transformations, 172
 - Java language, 170
 - languages, 170–171. *See also specific languages.*
 - lazy execution model, 172
 - passing functions as parameters, 171
 - Python language, 170
 - restricted shared variables, 172
 - Scala language, 170
 - shared variables, 172
- Spark programming, RDDs
 - actions, 170
 - creating from existing RDDs, 170
 - definition, 170
 - lineage, 173
- Spark Shell
 - overview, 182–183
 - running programs locally, 183–184
 - running spark-shell on a cluster, 184–185
 - vs.* Spark applications, 181–182
- Spark SQL
 - connecting to Hive, 199
 - HiveContext, 198–199
 - initializing, 199
 - loading data, 199–200
 - overview, 198–199
 - querying data, 200
 - in the Spark stack, 154
 - SQLContext, 198–199
- Spark SQL, DataFrames
 - creating, 198, 200–201
 - creating with RDDs, 200–201
 - description, 198
 - displaying contents, 200
 - filter() operation, 200
 - filtering rows, 200
 - groupBy operation, 200
 - grouping data, 200
 - operations on, 200
 - select() operation, 200
 - selecting fields or functions, 200
 - show() operation, 200
- Spark SQL query optimizer
 - code generation, 713–714
 - data locality, 715–716
 - logical plan, 712–713
 - optimizer steps, 712–714
 - overview, 712
 - physical plan, 713
 - speculative execution, 714
 - viewing Spark properties, 713–714
- Spark Stack
 - components, 154–155
 - GraphX, 155
 - MLib, 155
 - overview, 153–154
 - Spark Core, 154
 - Spark SQL, 154
 - Spark Streaming, 155
 - Standalone Scheduler, 155
- Spark standalone cluster manager, 163
- Spark streaming, 195–197
 - description, 155
 - example, 197–198
 - functional description, 195–196
 - micro-batching, 196
 - overview, 194–195
 - streaming sources, 196
 - windowed computations, 196
- spark.akka.framesize property, 676
- spark.akka.threads property, 676
- spark.cleaner.ttl property, 700
- SparkConf, 193, 661–662
- SparkContext
 - entry point to the Spark API, 183
 - naming conventions, 186

- SparkContext (*continued*)
 - running Spark applications, 185–186
 - in Spark standalone clusters, 159
- SparkContext objects
 - creating, 182
 - in Spark cluster execution, 185
- SparkContext.newAPIHadoopFile method, 165
- SparkContext.wholeTextFiles format, 165
- spark.default.parallelism configuration
 - property, 706
- spark.default.parallelism property, 709
- spark-defaults.conf file, 661
- spark.driver.cores property, 665
- spark.driver.maxResultSize property, 675
- spark.driver.memory property, 665
- spark.dynamicAllocation.enabled property, 667, 677
- spark.dynamicAllocation.
 - executorIdleTimeout property, 677
- spark.dynamicAllocation.
 - schedulerbacklogTimeout property, 677
- spark.dynamicAllocation.
 - sustainedSchedulerBacklogTimeout property, 677
- spark.executor.cores parameter, 667
- spark.executor.cores property, 661
- spark.executor.instances property, 667
- spark.executor.memory parameter, 667
- spark.locality.wait property, 715
- spark.locality.wait.node property, 715
- spark.memory.fraction property, 670
- spark.memory.storageFraction property, 670
- spark.reducer.maxSizeInFlight property, 697
- Spark-related network parameters, 676
- spark-shell command, 661
- spark.shuffle.compress property, 697
- spark.shuffle.file.buffer property, 697
- spark.shuffle.spill.compress property, 697
- spark.speculation.multiplier property, 714
- spark-submit command, 661
- spark-submit script
 - cluster URL, specifying, 188
 - description, 187
 - example, 187
 - help command, 187–188
 - master flag, 188
 - running Spark applications in local mode, 188
 - spark.yarn.am.cores property, 664
 - spark.yarn.am.memory property, 664
- Special principal, Kerberos realms, 485
- specified rule, 430
- Speculative execution, 621–624, 714
- Speed, Spark, 149–151
- Spill phase, tuning map tasks, 626, 628–630
- Spill process, optimizing MapReduce, 633
- SPILLED_RECORDS counter, 650
- split-by id parameter, 376
- Splittability, file formats, 297
- Splitting data along column lines, 376
- Splitting queries into chunks, 376
- SPNs (service principal names)
 - defining, 503–504
 - deleting, 493
 - Kerberos realms, 483
 - mapping, 495–497
 - setting up, 492–493
 - translating to operating system names, 495–496
- SQL query optimizer
 - optimizing, 712–716
 - Spark. *See* Spark SQL query optimizer.
- SQLContext, Spark SQL, 198–199
- Sqoop. *See also* Sqoop 2.
 - architecture, 366–367
 - connectors, 367
 - deploying, 367
 - description, 17, 356
 - drivers, 367
 - help feature, 368
- Sqoop, exporting data
 - functional description, 383–385
 - from Hive to a database, 386
 - number of mappers, specifying, 382, 383
 - overview, 382–383
 - simultaneous update or insertion, 385–386
 - stored procedures, 386
- Sqoop, loading data from relational databases to HDFS
 - into Avro files, 373
 - in binary format, 373
 - combining new datasets with old, 379
 - compressing table data, 373–374
 - creating Sqoop jobs, 377
 - free form import, 375–376

- getting data from all tables, 376–377
- import process, overview, 368–371
- incremental imports, 378–379
- input parsing options, 373
- input/output delimiters, 372–373
- job parallelism, 377–378
- listing relational databases, 368
- listing tables in a database, 368
- metastore, sharing, 372
- options file, 371
- passwords, specifying, 372
- selecting a target directory, 374, 376
- selective import, 374–376
 - into SequenceFiles, 373
 - specifying an access mode, 374
 - splitting data along column lines, 376
 - splitting queries into chunks, 376
- Sqoop, loading data from relational databases
 - to Hive
 - overview, 379–381
 - partitioned Hive tables, 381
- Sqoop 2, 387–388. *See also* Sqoop.
- Sqoop jobs, creating, 377
- SSD storage type, 237
- SSH
 - passwordless SSH, configuring, 105
 - setting up on pseudo-distributed Hadoop clusters, 68–69
- sshfence method, 340–341
- Stage boundaries, 699
- Stages
 - definition, 180
 - description, 181
 - Spark execution model, 693
- Stages tab, 682, 684
- Staging directory, specifying, 113
- staging-table parameter, 382
- Standalone cluster manager
 - architecture, 159–160
 - driver program, 159
 - executor, 159
 - master nodes, configuring, 161
 - master processes, starting/stopping, 161
 - setting up, 159
 - tasks, 159
 - worker nodes, configuring, 161
 - worker processes, starting/stopping, 161
 - vs.* YARN, 163
- Standalone clusters, Spark, 158
- Standalone installation, 61–62
- Standalone Scheduler
 - Spark cluster manager, 155
 - Spark Stack, 155
- Standalone Spark cluster, running Spark applications, 186–187
- Standby NameNode
 - checkpointing, 325, 327–328
 - metadata file location, specifying, 85
 - query errors, 346
- Standby NameNode service, in Hadoop clusters, 36, 88
- start control nodes
 - configuring, 456
 - Oozie workflows, 438, 446–447, 448, 456
- start-balancer.sh command, 268–271
- Starting up and shutting down
 - fully distributed clusters, 114–117
 - Hadoop services, 90
 - shutdown/startup scripts, 546
- Statistical analysis. *See* Data science.
- status option, 532
- stderr logs, 583–584
- stdout logs, 583–584
- Storage
 - architecture, archival storage, 234–235
 - fully distributed clusters, single rack to multiple racks, 96–99
- Storage levels
 - DISK_ONLY, 719
 - MEMORY_AND_DISK, 719
 - MEMORY_AND_DISK_SER, 719
 - MEMORY_ONLY, 719
 - MEMORY_ONLY_SER, 719
 - setting, 720–721, 721–722
- Storage policies, cold data, 237
- Storage preferences for files, archival storage, 235
- Storage tab, 684
- Storage types, archival storage, 236–239
- Storing data. *See* HDFS storage.
- Storm
 - alternative to MapReduce, 25
 - description, 17
 - integrating with Kafka, 404–406
- Streaming access to data, HDFS, 38
- Streaming data. *See* Spark streaming.

- StreamingContext, 195–197
 - Structured data
 - handling. *See* Spark SQL.
 - traditional database systems, 8
 - Subqueues, Capacity Scheduler
 - configuring, 414
 - creating, 413–414
 - diagram, 418
 - setting up, 415–416
 - sudo command, 259
 - Super user group, specifying, 108
 - Super users, designating, 259
 - supported_encetypes parameter, 489
 - SUSE, package manager for, 63
 - suspend command, 472
 - Suspending running jobs, 472–473
 - Swap, disabling, 66
 - syslog logs, 583–584
 - System namespace, 509–510
- T**
- T option, 310
 - Table data, compressing, 373–374
 - Tables in a database, listing, 368
 - Tachyon, 722
 - tail command, 360
 - target-dir parameter, 376
 - Task durations, monitoring, 684, 685
 - Task failures, troubleshooting, 738–739
 - Task IDs, troubleshooting, 736
 - Task Metrics tab, 684, 685
 - Task progress, reporting, 511
 - Tasks
 - cluster computing, 13
 - definition, 180
 - description, 181
 - standalone cluster manager, 159
 - YARN, 49
 - Tasks, in Spark applications
 - optimizing, 703–710
 - overview, 703–704
 - Spark execution model, 693–694
 - too few, 706
 - Temporary data, storage policies, 237
 - TEMPORARY replica state, 216
 - TeraGen, 641–642
 - TeraSort
 - benchmarking clusters, 640–643
 - generating test data, 641–642
 - overview, 640–641
 - sorting test data, 642
 - TeraGen, 641–642
 - TeraSort, 642
 - TeraValidate, 642
 - using benchmarks, 642
 - utility suite, 641
 - validating test output, 642
- TeraValidate, 642
 - test command, 263, 357
 - TestDFSIO, testing I/O performance, 638–640
 - Testing. *See also* Benchmarking clusters.
 - disk speed, 65
 - for files, 357
 - HA (high availability), NameNode setup, 345
 - I/O performance, benchmarking clusters, 638–640
 - Text file type, 290
 - Text files
 - accessing with Spark, 164–165
 - creating RDD files from, 175
 - description, 298
 - Tez, description, 17
 - TGS (Ticket Granting Service)
 - definition, 480
 - maximum life, specifying, 490
 - maximum renewal time, 490
 - TGTs (Ticket Granting Tickets)
 - clearing a ticket cache, 503
 - description, 480
 - “failed to find any kerberos tgt” message, 502
 - listing a user’s ticket cache, 503
 - retrieving, 502
 - service tickets, 483
 - THP compaction, turning off, 68
 - Thrashing, monitoring, 571
 - Thrift protocol, 192
 - Thrift Server, 192
 - Ticket cache
 - clearing a, 503
 - listing, 503
 - Ticket Granting Service (TGS). *See* TGS (Ticket Granting Service).
 - Ticket Granting Tickets (TGTs). *See* TGTs (Ticket Granting Tickets).

- Tickets
 - definition, 483
 - granting tickets, 502
 - viewing, 502
- Time-based scheduling, 465–467, 469
- Tokens, 501
- top command, 530, 574–575
- Topics, Kafka, 400, 403
- topology.py script, 210
- toSnapshot parameter, 282–283
- TOTAL_LAUNCHED_MAPS counter, 649
- TOTAL_LAUNCHED_REDUCES counter, 649
- Trace Builder, benchmarking clusters, 643–644
- trace option, 645
- Transformations. *See* RDD (resilient distributed dataset), transformations.
- transitionToActive command, 349, 535
- transitionToStandby command, 349, 535, 545
- Trash directory. *See also* lost+found directory.
 - bypassing, 280
 - checkpointing interval, setting, 278–279
 - configuring, 278–279
 - data retention interval, setting, 278–279
 - description, 250, 278
 - emptying, 250, 279
 - enabling, 278
 - moveToTrash() method, 278
 - permanently deleting files, 250, 278–279
 - preventing accidental data deletion, 278–280
 - restoring deleted files, 278
 - selectively deleting files, 279
 - viewing contents of, 250
- Trash retention interval, setting, 81
- Troubleshooting
 - Oozie, 473–474
 - performance, 682–684
 - YARN jobs that are stuck, 731–732
- Troubleshooting, failure types
 - ApplicationMaster crashes, 738
 - daemon failures, 737
 - job failures, 738–739
 - NameNode crashes, 737–738
 - NodeManager failures, 738
 - ResourceManager crashes, 738
 - retrying jobs after a failure, 738–739
 - starting failures for Hadoop daemons, 737–738
 - task failures, 738–739
 - work preserving recovery, 739
- Troubleshooting JVM garbage collection
 - optimizing, 733–734
 - overview, 732–733
 - Spark JVM garbage collection, 734
- Troubleshooting JVM memory allocation
 - analyzing memory usage, 734
 - ApplicationMaster memory issues, 735–736
 - heap dumps, 734
 - job IDs, 736
 - out of memory errors, 734–735
 - task IDs, 736
- Troubleshooting space issues
 - disk volume failure toleration, 729–730
 - HDFS issues, 727
 - hot swapping a disk drive, 729
 - Linux file system 1 full, 726
 - local directories out of free space, 727–729
 - log directories out of free space, 727–729
 - overview, 725–726
 - replication, 730
 - setting dfs.datanode.du.reserved parameter, 730
- Troubleshooting Spark jobs. *See also* Debugging Spark applications.
 - fault tolerance, 740
 - killing Spark jobs, 740
 - maximum attempts, specifying, 740
 - maximum failures per job, specifying, 740
 - task failures, 739
- Trusted relationships, Kerberos realms, 484–485
- Tuning. *See also* Optimization; Performance.
 - administrator duties, 20
 - GC (garbage collection), 686–688
- Tuning map tasks. *See also* Optimizing MapReduce.
 - compression, 628
 - data locality, 626–627
 - input split size, 627–628
 - input/output, 627–630
 - map phase, 626, 628–630
 - merge phase, 626
 - overview, 625–626
 - read phase, 626
 - spill phase, 626, 628–630

Tuning reduce tasks. *See also* Optimizing MapReduce.

- merge phase, 630–632
- reduce phase, 630–632
- shuffle phase, 630–632
- sort performance, 632
- write phase, 630–632

Tuning Spark streaming applications. *See also* Optimizing Spark applications.

- garbage collection, 689
- memory, 689
- overview, 688
- parallelizing data ingestion, 688
- parallelizing data processing, 689
- reducing batch processing time, 688–689
- setting the batch interval, 689

twittersource, 394

256-byte encryption, enabling/disabling, 490

Two-way trust, Kerberos realms, 485–486

U

Uberized jobs, 646

Ubuntu Linux, 63, 743

Ulimits, setting, 67–68

UNDER_CONSTRUCTION block state, 218–219

UNDER_RECOVERY block state, 218–219

Under-replicated files, 289

Unrecoverable files, 288–289

Unstructured data, definition, 6

update option, 364–365

update-key parameter, 382, 385–386

update-mode parameter, 382, 386

Upgrades. *See* Installation and upgrades.

UPNs (user principal names)

- Kerberos realms, 483, 490
- provisioning on Kerberized clusters, 503–504
- setting up, 491

User accounts

- creating, 554–556
- functional, 727

User capabilities, limiting, 419–420

User identity, verifying. *See* Authentication.

User impersonation, 558

User metrics, 577

User namespace, 509–510

User principal, Kerberos realms, 483

user rule, 430

User specific space quotas, 264

User whitelist, 511

Users

- enabling new users, 257–258
- Sentry authorization, 513
- super users, designating, 259
- user identities, 258–259
- using administrative privileges, 259

users option, 645

Utilities

- automated deployment tools, 63
- copying data between hosts, 63
- Crowbar, 63
- curl, 63
- executing remote commands, 63
- FTP protocol, 63
- HTTP protocol, 63
- installing pseudo-distributed Hadoop clusters, 63
- KickStart, 63
- package manager for Red Hat, SUSE and Fedora Linux, 63
- pdsh, 63
- Red Hat Enterprise Linux RPM software packages, 63
- rpm, 63
- scp, 63
- sending and getting files, 63
- wget, 63
- yum, 63

V

validate command, 472

Validating

- benchmark test output, 642
- a workflow.xml file, 472
- XML schemas, 472

Vectorization, Hive jobs, 636–637

View (viewfs) file system, 244

Viewing, application logs, 584–585, 596–597

VirtualBox, installing, 744

Virtualization, fully distributed clusters, 97

Visualization, data science component, 11

vmstat utility, 572–573

VMware, Hadoop distribution, 60

W

w (write) permission, 255–256, 506

WANdisco, Hadoop distribution, 60

Warm data

archival storage, 232, 233–234

storage policies, 237

Web interfaces, fully distributed clusters,
119–121

Web UIs

as monitoring tools. *See* Monitoring with
web UIs.

setting, 114

WebHCat Server, 479–480

WebHDFS, 244

WebHDFS API

adding headers, 310–311

checking directory quotas, 313

creating directories, 312

creating files, 312

DELETE operation, 308, 312–313

following redirects, 310–311

GET operation, 308

vs. HttpFS gateway, 315

indicating the HTTP method, 310–311

overview, 308

point to an uploaded file, 310–311

POST operation, 308

PUT operation, 308, 312

reading files, 312

removing directories, 312–313

setting up, 309

using, 308–309

WebHDFS API, HDFS commands

curl tool, 310–311

H option, 310

L option, 310, 312

overview, 309–310

T option, 310

X option, 310

wget, 63

Whitelists, 511

Wide dependencies, 698–700

Wide transformations, 698

WordCount program

description, 130

running, 136–137

sample program, 135–136

Work preserving recovery, 739

Worker nodes

in Hadoop clusters, 36

HDFS, architecture, 38–39

Worker nodes, configuring, 161

Worker processes, 161, 180

Workflows, managing, 561–562

workflow.xml file, validating a, 472

Wrangling data. *See* Data wrangling.

Write (w) permission, 255–256, 506

Write phase, tuning reduce tasks, 630–632

Write tests, benchmarking clusters, 639

Writing, to an HDFS file, 42–43

X

X option, 310

x (execute) permission, 506

Y

YARN (Yet Another Resource Negotiator)

ApplicationMaster, 52–56

ApplicationsManager, 51

architecture, 49–50

clients, 49

component interactions, 54–56

configuring, 559–560

configuring in pseudo-distributed clusters,

80, 83–86. *See also* Modifying

fully distributed clusters, YARN

configuration.

containers, 50

daemons, setting up, 73–74

DataNodes, 49

Hadoop 2 *vs.* Hadoop 1, 21–22

Hadoop ecosphere, 15

job history metadata, 54

JobHistoryServer, starting, 88–89

jobs, 49

mapper tasks, 49

metrics, 577

in a multihomed network, 124, 562–563

NodeManager, 49, 52

NodeManager, starting, 88–89

operations, auditing, 519

overview, 48

reduce tasks, 49

resource management, 50–56

- YARN (*continued*)
 - ResourceManager, 49
 - ResourceManager, starting, 88–89
 - Scheduler, 51
 - services, starting, 88–89
 - setting up on pseudo-distributed Hadoop clusters, 70–71
 - Spark applications, 189
 - vs.* standalone cluster manager, 163
 - tasks, 49
 - web interface, fully distributed clusters, 121
- YARN commands for managing applications
 - administrative commands, 534–535
 - application command, 531
 - applicationattempt command, 532
 - checkHealth command, 535
 - displaying cluster usage, 530
 - failover command, 535
 - filtering lists of applications, 531–532
 - getServiceState command, 535
 - help for, 530
 - job queue status, checking, 533
 - kill command, 532–533
 - killing, 532–533
 - list command, 531–532
 - logs, reviewing, 533–534
 - logs command, 533–534
 - node command, 533
 - node status, checking, 533
 - nodes, listing, 533
 - overview, 530–531
 - queue command, 533
 - refreshNodes command, 535
 - status, checking, 532
 - status option, 532
 - top, 530
 - transitionToActive command, 535
 - transitionToStandby command, 535
 - viewing job information, 531
- yarn user, setting up, 70–71
- yarn.application.classpath parameter, 113
- yarn.app.mapreduce.am.command-opts parameter, 618
- yarn.app.mapreduce.am.resource.mb parameter, 618
- yarn.app.mapreduce.am.staging_dir parameter, 113–114
- yarn-client mode, 662
- yarn-cluster mode, 662
- YARN_CONF_DIR environment variable, 163
- yarn-env.sh file, 79
- yarn.exclude file, 536
- yarn.include file, 536
- yarn.log.aggregation-enable parameter, 111–112
- yarn.log.aggregation.retain-seconds parameter, 595
- YARN_LOG_DIR parameter, 597
- yarn.log.server.url parameter, 595
- yarn.nodemanager.aux-services parameter, 109
- yarn.nodemanager.aux-services.property, 83
- yarn.nodemanager.aux-services.mapreduce_shuffle-class parameter, 109
- yarn.nodemanager.aux-services.mapreduce_shuffle.class property, 84
- yarn.nodemanager.container parameter, 498
- yarn.nodemanager.disk-health-checker.max-disk-utilization-perdisk-percentage parameter, 727–729
- yarn.nodemanager.disk-health-checker.min-healthydisks parameter, 727–729
- yarn.nodemanager.keytab parameter, 498
- yarn.nodemanager.linux-container-executor.group parameter, 498
- yarn.nodemanager.local-dirs parameter, 112, 498, 681
- yarn.nodemanager.local-dirs property, 588
- yarn.nodemanager.log.deletion-threads-count parameter, 594
- yarn.nodemanager.log-dirs parameter, 112, 498
- yarn.nodemanager.log.retain-seconds parameter, 594
- yarn.nodemanager.principal parameter, 498
- yarn.nodemanager.remote-app-log-dir parameter, 593
- yarn.nodemanager.resource.cpu-vcores parameter, 110, 620–621
- yarn.nodemanager.resource.cpu-vcores property, 662
- yarn.nodemanager.resource.memory-mb parameter, 109–110, 614, 661
- yarn.nodemanager.vmem-pmem-ratio parameter, 617

- yarn.resourcemanager.keytab parameter, 498
 - yarn.resourcemanager.nodes.exclude-path parameter, 536
 - yarn.resourcemanager.nodes.include-path parameter, 536
 - yarn.resourcemanager.principal parameter, 498
 - yarn.scheduler.maximum-allocation-vcores parameter, 621
 - yarn.scheduler.minimum-allocation-mb property, 662
 - yarn.scheduler.minimum-allocation-vcores parameter, 621
 - yarn-site.xml file
 - allocating memory for containers, 614
 - configuring pseudo-distributed Hadoop clusters, 74
 - configuring ratio of physical memory to virtual, 617
 - configuring the Fair Scheduler, 428–430
 - configuring virtual cores, 620–621
 - configuring YARN, 83–86
 - decommissioning a NodeManager service, 536
 - mapreduce.jobhistory.bind-host parameter, 124
 - mapreduce.map.cpu.vcores, 621
 - mapreduce.reduce.cpu.vcores, 621
 - memory related parameters, 109
 - YARN in a multihomed network, 124
 - yarn .scheduler.maximum-allocation-vcores parameter, 621
 - yarn .scheduler.minimum-allocation-vcores parameter, 621
 - yarn.nodemanager.resource.cpu-vcores parameter, 620–621
 - yarn.nodemanager.resource.memory-mb parameter, 614
 - yarn.nodemanager.vmem-pmem-ratio parameter, 617
 - yarn-site.xml file, configuration parameters, 498
 - yarn.xml file, 109
 - Yet Another Resource Negotiator (YARN). *See* YARN (Yet Another Resource Negotiator).
 - Young Generation garbage collection, 687
 - Young generations, JVM garbage collection, 732–733
 - yum, 63
- Z**
- ZKFC (ZooKeeper Failover controller), 347–348
 - ZooKeeper
 - configuring, 560
 - description, 17, 47
 - as a high availability coordinator, 335
 - large cluster guidelines, 102
 - setting up for Kafka, 402