Addison Wesley Data & Analytics Series

EXPERT HADOOP® ADMINISTRATION

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

SAM R. ALAPATI

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

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

ISBN-13: 978-0-13-459719-5
ISBN-10: 0-13-459719-2

1 16

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.

Contents

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 garage 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.

III

Managing and Protecting Hadoop Data and High Availability

This page intentionally left blank

9

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.

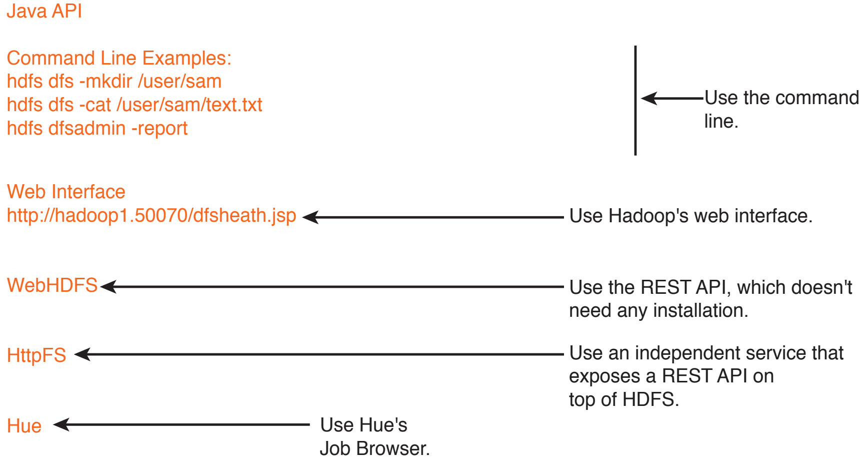


Figure 9.1 The many ways in which you can access HDFS

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 -[defsz] <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
```

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 1432500251198
$
```

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
$
```

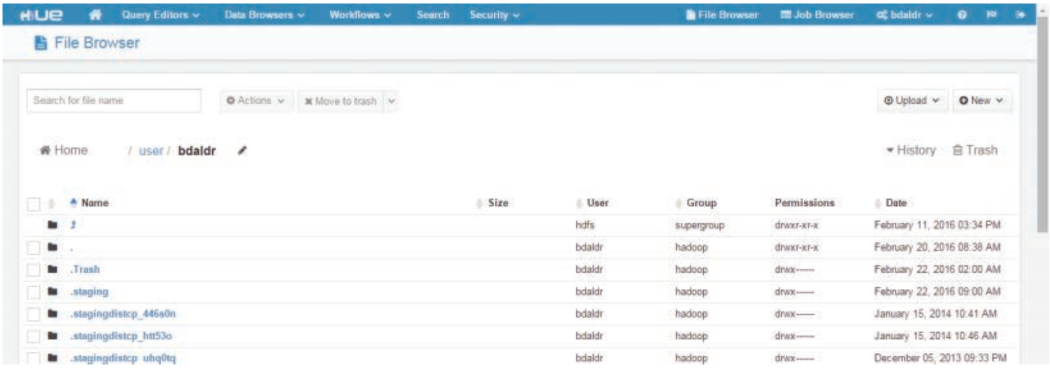


Figure 9.4 Hue's File Browser, showing how you can access HDFS from Hue

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:produsers  /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.

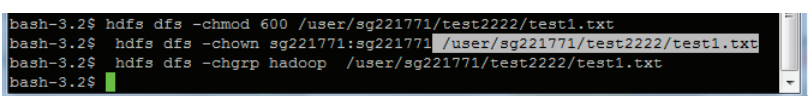


Figure 9.5 Changing file mode, ownership and group with HDFS commands

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.

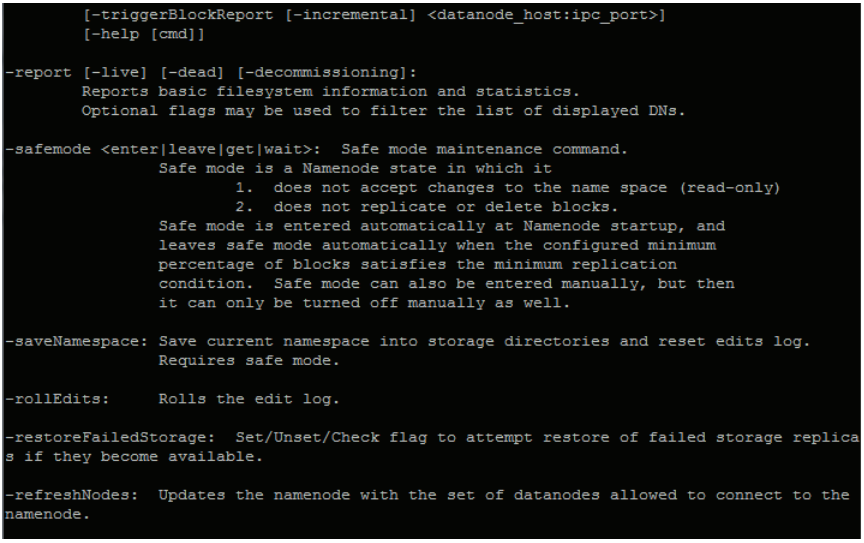
In this section, I'm using HDFS commands from the command line to view and manipulate HDFS files and directories. However, there's an even easier way to access HDFS, and that's through Hue, the web-based interface, which is extremely easy to use and which lets you perform HDFS operations through a GUI. 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 replica
s 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/test2222/
put: Permission denied: user=root, access=WRITE, inode="/user/alapati/
test2222":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.supergroup` configuration parameter in the hdfs-site.xml file, as shown here.

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

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

```
# 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 sudoing 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>df.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 X 3= 30GB 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*12=1556MB 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/bdaldr
        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 with 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

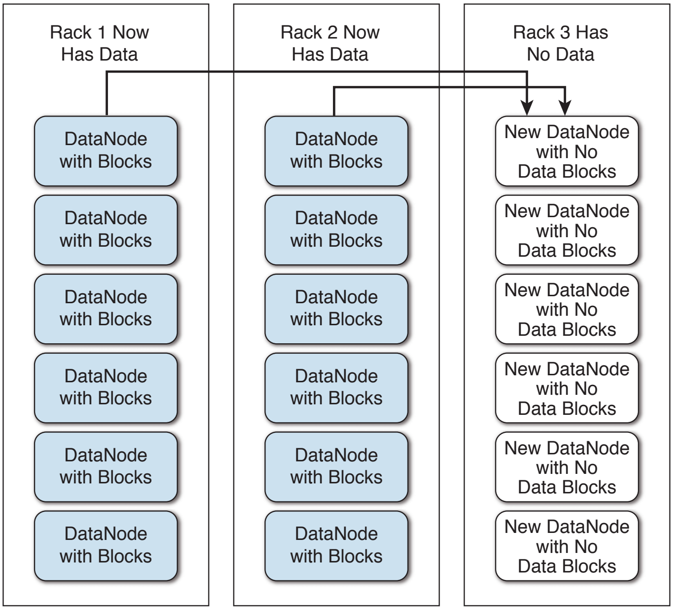


Figure 9.8 How the balancer moves data blocks to the under-utilized nodes from the over-utilized nodes

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 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:

  ```
  Average DFS Used = DFS Used * 100/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 14.56153333333334 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 under-utilized 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.
- RThe 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

M

N

Q

R