1 Hadoop in a heartbeat

This chapter covers

- Examining how the core Hadoop system works
- Understanding the Hadoop ecosystem
- Running a MapReduce job

We live in the age of big data, where the data volumes we need to work with on a day-to-day basis have outgrown the storage and processing capabilities of a single host. Big data brings with it two fundamental challenges: how to store and work with voluminous data sizes, and more important, how to understand data and turn it into a competitive advantage.

Hadoop fills a gap in the market by effectively storing and providing computational capabilities for substantial amounts of data. It's a distributed system made up of a distributed filesystem, and it offers a way to parallelize and execute programs on a cluster of machines (see figure 1.1). You've most likely come across Hadoop because it's been adopted by technology giants like Yahoo!, Facebook, and Twitter to address their big data needs, and it's making inroads across all industrial sectors.

Because you've come to this book to get some practical experience with Hadoop and Java,[1] I'll start with a brief overview and then show you how to install

[1] To benefit from this book, you should have some practical experience with Hadoop and understand the basic concepts of MapReduce and HDFS (covered in Manning's *Hadoop in Action* by Chuck Lam, 2010). Further, you should have an intermediate-level knowledge of Java—*Effective Java,* 2nd Edition by Joshua Bloch (Addison-Wesley, 2008) is an excellent resource on this topic.

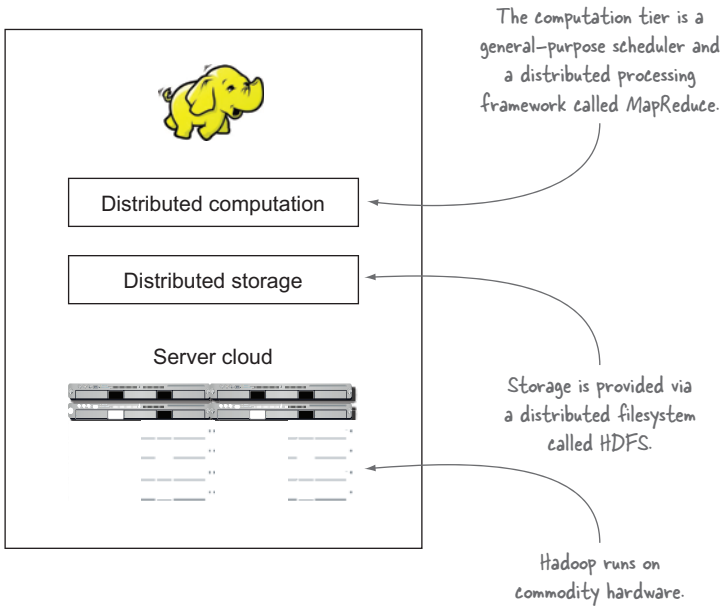


Figure 1.1 The Hadoop environment is a distributed system that runs on commodity hardware.

Hadoop and run a MapReduce job. By the end of this chapter, you'll have had a basic refresher on the nuts and bolts of Hadoop, which will allow you to move on to the more challenging aspects of working with it.

Let's get started with a detailed overview.

1.1 What is Hadoop?

Hadoop is a platform that provides both distributed storage and computational capabilities. Hadoop was first conceived to fix a scalability issue that existed in Nutch,[2] an open source crawler and search engine. At the time, Google had published papers that described its novel distributed filesystem, the Google File System (GFS), and MapReduce, a computational framework for parallel processing. The successful implementation of these papers' concepts in Nutch resulted in it being split into two separate projects, the second of which became Hadoop, a first-class Apache project.

In this section we'll look at Hadoop from an architectural perspective, examine how industry uses it, and consider some of its weaknesses. Once we've covered this background, we'll look at how to install Hadoop and run a MapReduce job.

Hadoop proper, as shown in figure 1.2, is a distributed master-slave architecture[3] that consists of the following primary components:

---

[2] The Nutch project, and by extension Hadoop, was led by Doug Cutting and Mike Cafarella.

[3] A model of communication where one process, called the *master*, has control over one or more other processes, called *slaves*.

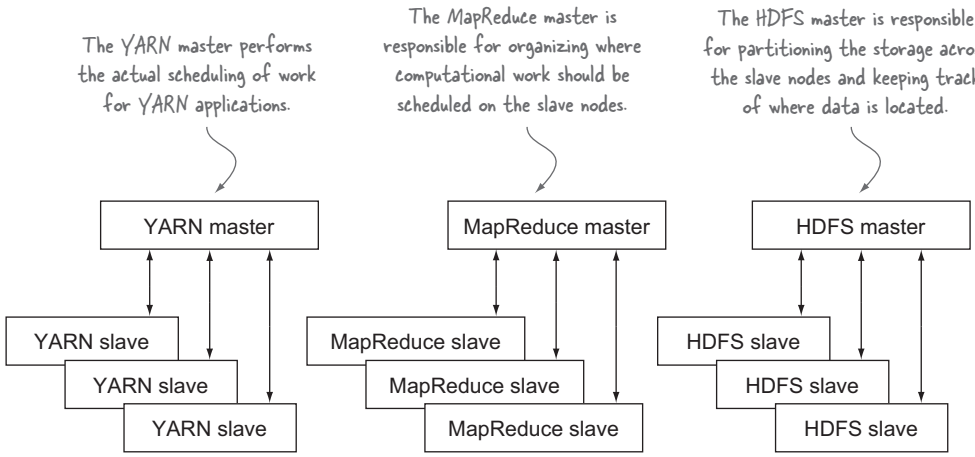


Figure 1.2 High-level Hadoop 2 master-slave architecture

- Hadoop Distributed File System (HDFS) for data storage.
- Yet Another Resource Negotiator (YARN), introduced in Hadoop 2, a general-purpose scheduler and resource manager. Any YARN application can run on a Hadoop cluster.
- MapReduce, a batch-based computational engine. In Hadoop 2, MapReduce is implemented as a YARN application.

Traits intrinsic to Hadoop are data partitioning and parallel computation of large datasets. Its storage and computational capabilities scale with the addition of hosts to a Hadoop cluster; clusters with hundreds of hosts can easily reach data volumes in the petabytes.

In the first step in this section, we'll examine the HDFS, YARN, and MapReduce architectures.

1.1.1 Core Hadoop components

To understand Hadoop's architecture we'll start by looking at the basics of HDFS.

HDFS

HDFS is the storage component of Hadoop. It's a distributed filesystem that's modeled after the Google File System (GFS) paper.[4] HDFS is optimized for high throughput and works best when reading and writing large files (gigabytes and larger). To support this throughput, HDFS uses unusually large (for a filesystem) block sizes and data locality optimizations to reduce network input/output (I/O).

Scalability and availability are also key traits of HDFS, achieved in part due to data replication and fault tolerance. HDFS replicates files for a configured number of times, is tolerant of both software and hardware failure, and automatically re-replicates data blocks on nodes that have failed.

[4] See "The Google File System," http://research.google.com/archive/gfs.html.

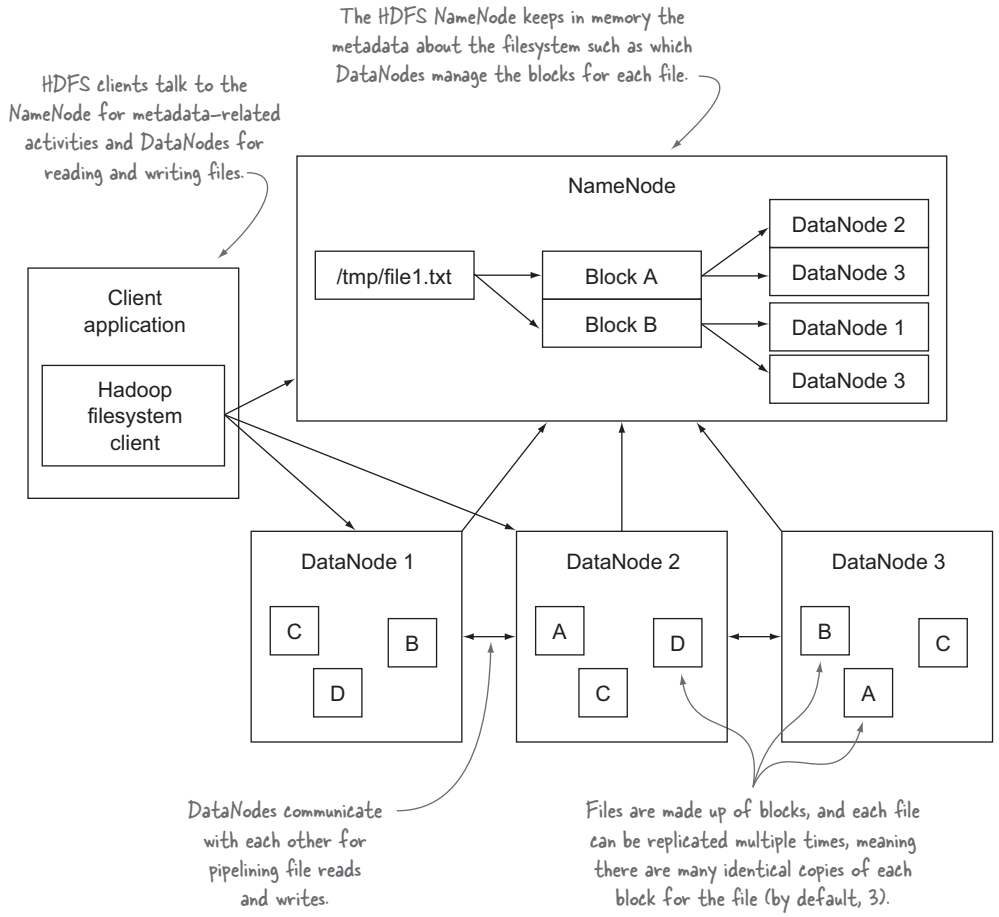


Figure 1.3 An HDFS client communicating with the master NameNode and slave DataNodes

Figure 1.3 shows a logical representation of the components in HDFS: the NameNode and the DataNode. It also shows an application that's using the Hadoop filesystem library to access HDFS.

Hadoop 2 introduced two significant new features for HDFS—Federation and High Availability (HA):

- Federation allows HDFS metadata to be shared across multiple NameNode hosts, which aides with HDFS scalability and also provides data isolation, allowing different applications or teams to run their own NameNodes without fear of impacting other NameNodes on the same cluster.
- High Availability in HDFS removes the single point of failure that existed in Hadoop 1, wherein a NameNode disaster would result in a cluster outage. HDFS HA also offers the ability for failover (the process by which a standby NameNode takes over work from a failed primary NameNode) to be automated.

Now that you have a bit of HDFS knowledge, it's time to look at YARN, Hadoop's scheduler.

YARN

YARN is Hadoop's distributed resource scheduler. YARN is new to Hadoop version 2 and was created to address challenges with the Hadoop 1 architecture:

- Deployments larger than 4,000 nodes encountered scalability issues, and adding additional nodes didn't yield the expected linear scalability improvements.
- Only MapReduce workloads were supported, which meant it wasn't suited to run execution models such as machine learning algorithms that often require iterative computations.

For Hadoop 2 these problems were solved by extracting the scheduling function from MapReduce and reworking it into a generic application scheduler, called YARN. With this change, Hadoop clusters are no longer limited to running MapReduce workloads; YARN enables a new set of workloads to be natively supported on Hadoop, and it allows alternative processing models, such as graph processing and stream processing, to coexist with MapReduce. Chapters 2 and 10 cover YARN and how to write YARN applications.

YARN's architecture is simple because its primary role is to schedule and manage resources in a Hadoop cluster. Figure 1.4 shows a logical representation of the core components in YARN: the ResourceManager and the NodeManager. Also shown are the components specific to YARN applications, namely, the YARN application client, the ApplicationMaster, and the container.

To fully realize the dream of a generalized distributed platform, Hadoop 2 introduced another change—the ability to allocate containers in various configurations.

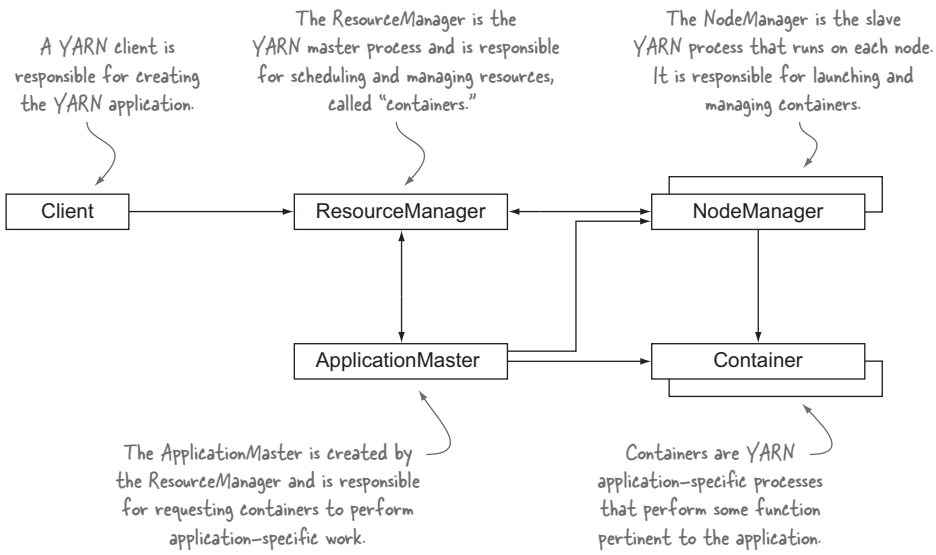


Figure 1.4 The logical YARN architecture showing typical communication between the core YARN components and YARN application components

Hadoop 1 had the notion of "slots," which were a fixed number of map and reduce processes that were allowed to run on a single node. This was wasteful in terms of cluster utilization and resulted in underutilized resources during MapReduce operations, and it also imposed memory limits for map and reduce tasks. With YARN, each container requested by an ApplicationMaster can have disparate memory and CPU traits, and this gives YARN applications full control over the resources they need to fulfill their work.

You'll work with YARN in more detail in chapters 2 and 10, where you'll learn how YARN works and how to write a YARN application. Next up is an examination of MapReduce, Hadoop's computation engine.

MAPREDUCE

MapReduce is a batch-based, distributed computing framework modeled after Google's paper on MapReduce.[5] It allows you to parallelize work over a large amount of raw data, such as combining web logs with relational data from an OLTP database to model how users interact with your website. This type of work, which could take days or longer using conventional serial programming techniques, can be reduced to minutes using MapReduce on a Hadoop cluster.

The MapReduce model simplifies parallel processing by abstracting away the complexities involved in working with distributed systems, such as computational parallelization, work distribution, and dealing with unreliable hardware and software. With this abstraction, MapReduce allows the programmer to focus on addressing business needs rather than getting tangled up in distributed system complications.

MapReduce decomposes work submitted by a client into small parallelized map and reduce tasks, as shown in figure 1.5. The map and reduce constructs used in

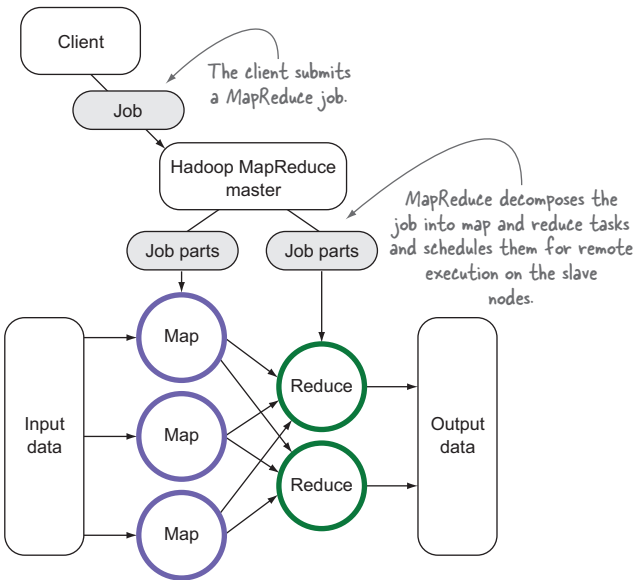


Figure 1.5 A client submitting a job to MapReduce, breaking the work into small map and reduce tasks

[5] See "MapReduce: Simplified Data Processing on Large Clusters," http://research.google.com/archive/mapreduce.html.

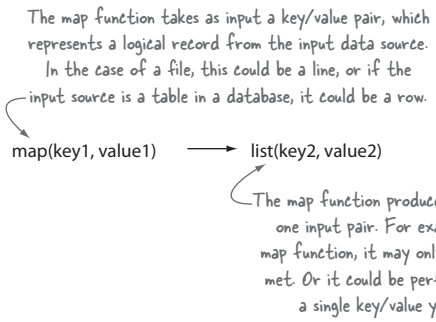


Figure 1.6 A logical view of the map function that takes a key/value pair as input

MapReduce are borrowed from those found in the Lisp functional programming language, and they use a shared-nothing model to remove any parallel execution interdependencies that could add unwanted synchronization points or state sharing.[6]

The role of the programmer is to define map and reduce functions where the map function outputs key/value tuples, which are processed by reduce functions to produce the final output. Figure 1.6 shows a pseudocode definition of a map function with regard to its input and output.

The power of MapReduce occurs between the map output and the reduce input in the shuffle and sort phases, as shown in figure 1.7.

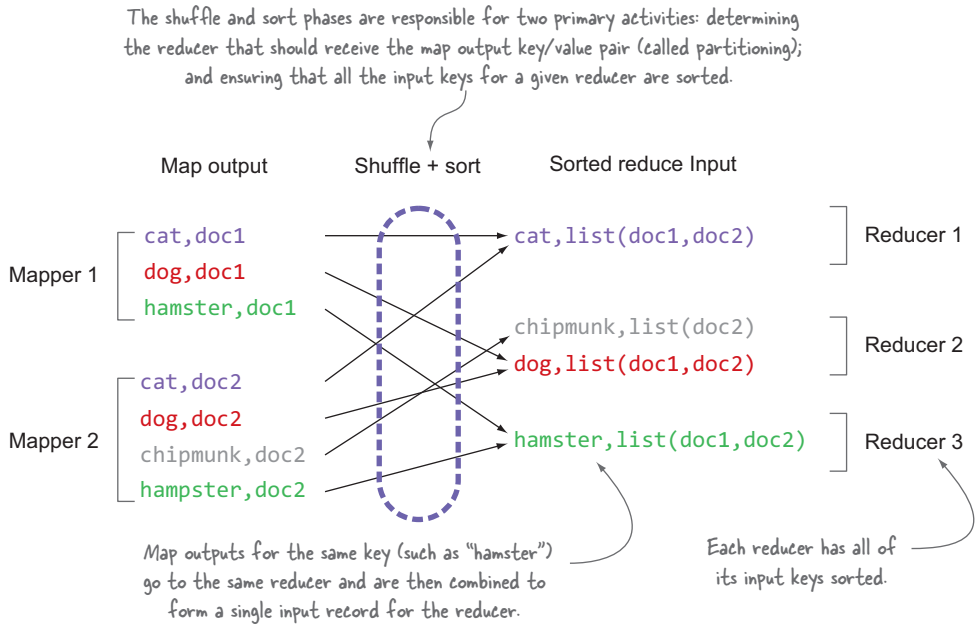


Figure 1.7 MapReduce's shuffle and sort phases

[6] A shared-nothing architecture is a distributed computing concept that represents the notion that each node is independent and self-sufficient.

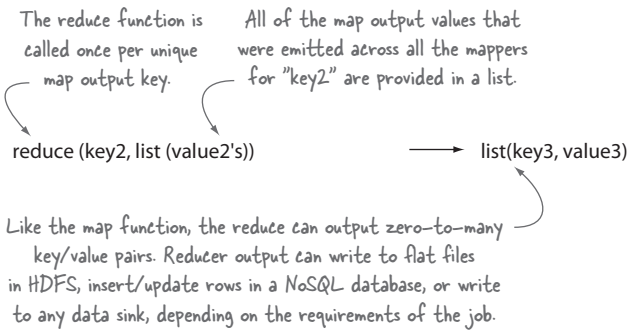


Figure 1.8 A logical view of the reduce function that produces output for flat files, NoSQL rows, or any data sink

Figure 1.8 shows a pseudocode definition of a reduce function.

With the advent of YARN in Hadoop 2, MapReduce has been rewritten as a YARN application and is now referred to as MapReduce 2 (or MRv2). From a developer's perspective, MapReduce in Hadoop 2 works in much the same way it did in Hadoop 1, and code written for Hadoop 1 will execute without code changes on version 2.[7] There are changes to the physical architecture and internal plumbing in MRv2 that are examined in more detail in chapter 2.

With some Hadoop basics under your belt, it's time to take a look at the Hadoop ecosystem and the projects that are covered in this book.

1.1.2 The Hadoop ecosystem

The Hadoop ecosystem is diverse and grows by the day. It's impossible to keep track of all of the various projects that interact with Hadoop in some form. In this book the focus is on the tools that are currently receiving the greatest adoption by users, as shown in figure 1.9.

MapReduce and YARN are not for the faint of heart, which means the goal for many of these Hadoop-related projects is to increase the accessibility of Hadoop to programmers and nonprogrammers. I'll cover many of the technologies listed in figure 1.9 in this book and describe them in detail within their respective chapters. In addition, the appendix includes descriptions and installation instructions for technologies that are covered in this book.

> **Coverage of the Hadoop ecosystem in this book** The Hadoop ecosystem grows by the day, and there are often multiple tools with overlapping features and benefits. The goal of this book is to provide practical techniques that cover the core Hadoop technologies, as well as select ecosystem technologies that are ubiquitous and essential to Hadoop.

Let's look at the hardware requirements for your cluster.

---

[7] Some code may require recompilation against Hadoop 2 binaries to work with MRv2; see chapter 2 for more details.

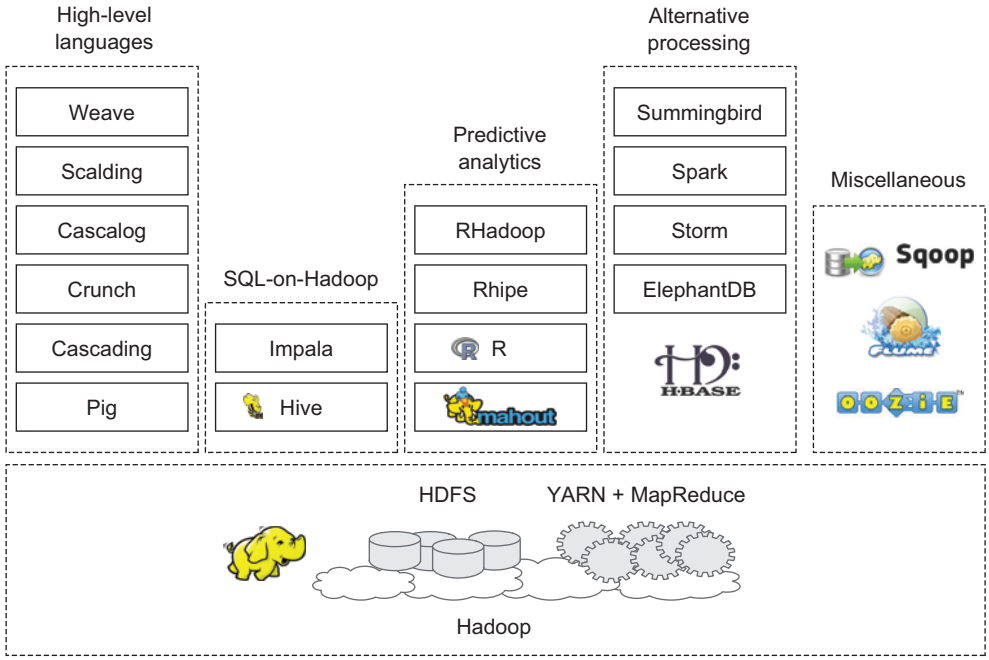


Figure 1.9 **Hadoop and related technologies that are covered in this book**

1.1.3 Hardware requirements

The term *commodity hardware* is often used to describe Hadoop hardware requirements. It's true that Hadoop can run on any old servers you can dig up, but you'll still want your cluster to perform well, and you don't want to swamp your operations department with diagnosing and fixing hardware issues. Therefore, *commodity* refers to mid-level rack servers with dual sockets, as much error-correcting RAM as is affordable, and SATA drives optimized for RAID storage. Using RAID on the DataNode filesystems used to store HDFS content is strongly discouraged because HDFS already has replication and error-checking built in; on the NameNode, RAID is strongly recommended for additional security.[8]

From a network topology perspective with regard to switches and firewalls, all of the master and slave nodes must be able to open connections to each other. For small clusters, all the hosts would run 1 GB network cards connected to a single, good-quality switch. For larger clusters, look at 10 GB top-of-rack switches that have at least multiple 1 GB uplinks to dual-central switches. Client nodes also need to be able to talk to all of the master and slave nodes, but if necessary, that access can be from behind a firewall that permits connection establishment only from the client side.

[8] HDFS uses disks to durably store metadata about the filesystem.

After reviewing Hadoop from a software and hardware perspective, you've likely developed a good idea of who might benefit from using it. Once you start working with Hadoop, you'll need to pick a distribution to use, which is the next topic.

1.1.4 Hadoop distributions

Hadoop is an Apache open source project, and regular releases of the software are available for download directly from the Apache project's website (http://hadoop.apache.org/releases.html#Download). You can either download and install Hadoop from the website or use a quickstart virtual machine from a commercial distribution, which is usually a great starting point if you're new to Hadoop and want to quickly get it up and running.

After you've whet your appetite with Hadoop and have committed to using it in production, the next question that you'll need to answer is which distribution to use. You can continue to use the vanilla Hadoop distribution, but you'll have to build the in-house expertise to manage your clusters. This is not a trivial task and is usually only successful in organizations that are comfortable with having dedicated Hadoop DevOps engineers running and managing their clusters.

Alternatively, you can turn to a commercial distribution of Hadoop, which will give you the added benefits of enterprise administration software, a support team to consult when planning your clusters or to help you out when things go bump in the night, and the possibility of a rapid fix for software issues that you encounter. Of course, none of this comes for free (or for cheap!), but if you're running mission-critical services on Hadoop and don't have a dedicated team to support your infrastructure and services, then going with a commercial Hadoop distribution is prudent.

> **Picking the distribution that's right for you** It's highly recommended that you engage with the major vendors to gain an understanding of which distribution suits your needs from a feature, support, and cost perspective. Remember that each vendor will highlight their advantages and at the same time expose the disadvantages of their competitors, so talking to two or more vendors will give you a more realistic sense of what the distributions offer. Make sure you download and test the distributions and validate that they integrate and work within your existing software and hardware stacks.

There are a number of distributions to choose from, and in this section I'll briefly summarize each distribution and highlight some of its advantages.

APACHE

Apache is the organization that maintains the core Hadoop code and distribution, and because all the code is open source, you can crack open your favorite IDE and browse the source code to understand how things work under the hood. Historically the challenge with the Apache distributions has been that support is limited to the goodwill of the open source community, and there's no guarantee that your issue will be investigated and fixed. Having said that, the Hadoop community is a very supportive one, and

responses to problems are usually rapid, even if the actual fixes will likely take longer than you may be able to afford.

The Apache Hadoop distribution has become more compelling now that administration has been simplified with the advent of Apache Ambari, which provides a GUI to help with provisioning and managing your cluster. As useful as Ambari is, though, it's worth comparing it against offerings from the commercial vendors, as the commercial tooling is typically more sophisticated.

CLOUDERA

Cloudera is the most tenured Hadoop distribution, and it employs a large number of Hadoop (and Hadoop ecosystem) committers. Doug Cutting, who along with Mike Caferella originally created Hadoop, is the chief architect at Cloudera. In aggregate, this means that bug fixes and feature requests have a better chance of being addressed in Cloudera compared to Hadoop distributions with fewer committers.

Beyond maintaining and supporting Hadoop, Cloudera has been innovating in the Hadoop space by developing projects that address areas where Hadoop has been weak. A prime example of this is Impala, which offers a SQL-on-Hadoop system, similar to Hive but focusing on a near-real-time user experience, as opposed to Hive, which has traditionally been a high-latency system. There are numerous other projects that Cloudera has been working on: highlights include Flume, a log collection and distribution system; Sqoop, for moving relational data in and out of Hadoop; and Cloudera Search, which offers near-real-time search indexing.

HORTONWORKS

Hortonworks is also made up of a large number of Hadoop committers, and it offers the same advantages as Cloudera in terms of the ability to quickly address problems and feature requests in core Hadoop and its ecosystem projects.

From an innovation perspective, Hortonworks has taken a slightly different approach than Cloudera. An example is Hive: Cloudera's approach was to develop a whole new SQL-on-Hadoop system, but Hortonworks has instead looked at innovating inside of Hive to remove its high-latency shackles and add new capabilities such as support for ACID. Hortonworks is also the main driver behind the next-generation YARN platform, which is a key strategic piece keeping Hadoop relevant. Similarly, Hortonworks has used Apache Ambari for its administration tooling rather than developing an in-house proprietary administration tool, which is the path taken by the other distributions. Hortonworks' focus on developing and expanding the Apache ecosystem tooling has a direct benefit to the community, as it makes its tools available to all users without the need for support contracts.

MAPR

MapR has fewer Hadoop committers on its team than the other distributions discussed here, so its ability to fix and shape Hadoop's future is potentially more bounded than its peers.

From an innovation perspective, MapR has taken a decidedly different approach to Hadoop support compared to its peers. From the start it decided that HDFS wasn't an

enterprise-ready filesystem, and instead developed its own proprietary filesystem, which offers compelling features such as POSIX compliance (offering random-write support and atomic operations), High Availability, NFS mounting, data mirroring, and snapshots. Some of these features have been introduced into Hadoop 2, but MapR has offered them from the start, and, as a result, one can expect that these features are robust.

As part of the evaluation criteria, it should be noted that parts of the MapR stack, such as its filesystem and its HBase offering, are closed source and proprietary. This affects the ability of your engineers to browse, fix, and contribute patches back to the community. In contrast, most of Cloudera's and Hortonworks' stacks are open source, especially Hortonworks', which is unique in that the entire stack, including the management platform, is open source.

MapR's notable highlights include being made available in Amazon's cloud as an alternative to Amazon's own Elastic MapReduce and being integrated with Google's Compute Cloud.

I've just scratched the surface of the advantages that the various Hadoop distributions offer; your next steps will likely be to contact the vendors and start playing with the distributions yourself.

Next, let's take a look at companies currently using Hadoop, and in what capacity they're using it.

1.1.5 Who's using Hadoop?

Hadoop has a high level of penetration in high-tech companies, and it's starting to make inroads in a broad range of sectors, including the enterprise (Booz Allen Hamilton, J.P. Morgan), government (NSA), and health care.

Facebook uses Hadoop, Hive, and HBase for data warehousing and real-time application serving.[9] Facebook's data warehousing clusters are petabytes in size with thousands of nodes, and they use separate HBase-driven, real-time clusters for messaging and real-time analytics.

Yahoo! uses Hadoop for data analytics, machine learning, search ranking, email antispam, ad optimization, ETL,[10] and more. Combined, it has over 40,000 servers running Hadoop with 170 PB of storage. Yahoo! is also running the first large-scale YARN deployments with clusters of up to 4,000 nodes.[11]

Twitter is a major big data innovator, and it has made notable contributions to Hadoop with projects such as Scalding, a Scala API for Cascading; Summingbird, a

[9] See Dhruba Borthakur, "Looking at the code behind our three uses of Apache Hadoop" on Facebook at http://mng.bz/4cMc. Facebook has also developed its own SQL-on-Hadoop tool called Presto and is migrating away from Hive (see Martin Traverso, "Presto: Interacting with petabytes of data at Facebook," http://mng.bz/p0Xz).

[10] Extract, transform, and load (ETL) is the process by which data is extracted from outside sources, transformed to fit the project's needs, and loaded into the target data sink. ETL is a common process in data warehousing.

[11] There are more details on YARN and its use at Yahoo! in "Apache Hadoop YARN: Yet Another Resource Negotiator" by Vinod Kumar Vavilapalli et al., www.cs.cmu.edu/~garth/15719/papers/yarn.pdf.

component that can be used to implement parts of Nathan Marz's lambda architecture; and various other gems such as Bijection, Algebird, and Elephant Bird.

eBay, Samsung, Rackspace, J.P. Morgan, Groupon, LinkedIn, AOL, Spotify, and StumbleUpon are some other organizations that are also heavily invested in Hadoop. Microsoft has collaborated with Hortonworks to ensure that Hadoop works on its platform.

Google, in its MapReduce paper, indicated that it uses Caffeine,[12] its version of MapReduce, to create its web index from crawl data. Google also highlights applications of MapReduce to include activities such as a distributed grep, URL access frequency (from log data), and a term-vector algorithm, which determines popular keywords for a host.

The number of organizations that use Hadoop grows by the day, and if you work at a Fortune 500 company you almost certainly use a Hadoop cluster in some capacity. It's clear that as Hadoop continues to mature, its adoption will continue to grow.

As with all technologies, a key part to being able to work effectively with Hadoop is to understand its shortcomings and design and architect your solutions to mitigate these as much as possible.

1.1.6 Hadoop limitations

High availability and security often rank among the top concerns cited with Hadoop. Many of these concerns have been addressed in Hadoop 2; let's take a closer look at some of its weaknesses as of release 2.2.0.

Enterprise organizations using Hadoop 1 and earlier had concerns with the lack of high availability and security. In Hadoop 1, all of the master processes are single points of failure, which means that a failure in the master process causes an outage. In Hadoop 2, HDFS now has high availability support, and the re-architecture of MapReduce with YARN has removed the single point of failure. Security is another area that has had its wrinkles, and it's receiving focus.

HIGH AVAILABILITY

High availability is often mandated in enterprise organizations that have high uptime SLA requirements to ensure that systems are always on, even in the event of a node going down due to planned or unplanned circumstances. Prior to Hadoop 2, the master HDFS process could only run on a single node, resulting in single points of failure.[13] Hadoop 2 brings NameNode High Availability (HA) support, which means that multiple NameNodes for the same Hadoop cluster can be running. With the current design, one of the NameNodes is active and the other NameNode is designated as a standby process. In the event that the active NameNode experiences a planned or

[12] In 2010 Google moved to a real-time indexing system called Caffeine; see "Our new search index: Caffeine" on the Google blog (June 8, 2010), http://googleblog.blogspot.com/2010/06/our-new-search-index-caffeine.html.

[13] In reality, the HDFS single point of failure may not be terribly significant; see "NameNode HA" by Suresh Srinivas and Aaron T. Myers, http://goo.gl/1iSab.

unplanned outage, the standby NameNode will take over as the active NameNode, which is a process called *failover*. This failover can be configured to be automatic, negating the need for human intervention. The fact that a NameNode failover occurred is transparent to Hadoop clients.

The MapReduce master process (the JobTracker) doesn't have HA support in Hadoop 2, but now that each MapReduce job has its own JobTracker process (a separate YARN ApplicationMaster), HA support is arguably less important.

HA support in the YARN master process (the ResourceManager) is important, however, and development is currently underway to add this feature to Hadoop.[14]

MULTIPLE DATACENTERS

Multiple datacenter support is another key feature that's increasingly expected in enterprise software, as it offers strong data protection and locality properties due to data being replicated across multiple datacenters. Apache Hadoop, and most of its commercial distributions, has never had support for multiple datacenters, which poses challenges for organizations that have software running in multiple datacenters. WANdisco is currently the only solution available for Hadoop multidatacenter support.

SECURITY

Hadoop does offer a security model, but by default it's disabled. With the security model disabled, the only security feature that exists in Hadoop is HDFS file- and directory-level ownership and permissions. But it's easy for malicious users to subvert and assume other users' identities. By default, all other Hadoop services are wide open, allowing any user to perform any kind of operation, such as killing another user's MapReduce jobs.

Hadoop can be configured to run with Kerberos, a network authentication protocol, which requires Hadoop daemons to authenticate clients, both users and other Hadoop components. Kerberos can be integrated with an organization's existing Active Directory and therefore offers a single-sign-on experience for users. Care needs to be taken when enabling Kerberos, as any Hadoop tool that wishes to interact with your cluster will need to support Kerberos.

Wire-level encryption can be configured in Hadoop 2 and allows data crossing the network (both HDFS transport[15] and MapReduce shuffle data[16]) to be encrypted. Encryption of data at rest (data stored by HDFS on disk) is currently missing in Hadoop.

Let's examine the limitations of some of the individual systems.

---

[14] For additional details on YARN HA support, see the JIRA ticket titled "ResourceManager (RM) High-Availability (HA)," https://issues.apache.org/jira/browse/YARN-149.

[15] See the JIRA ticket titled "Add support for encrypting the DataTransferProtocol" at https://issues.apache.org/jira/browse/HDFS-3637.

[16] See the JIRA ticket titled "Add support for encrypted shuffle" at https://issues.apache.org/jira/browse/MAPREDUCE-4417.

HDFS

The weakness of HDFS is mainly its lack of high availability (in Hadoop 1.x and earlier), its inefficient handling of small files,[17] and its lack of transparent compression. HDFS doesn't support random writes into files (only appends are supported), and it's generally designed to support high-throughput sequential reads and writes over large files.

MAPREDUCE

MapReduce is a batch-based architecture, which means it doesn't lend itself to use cases that need real-time data access. Tasks that require global synchronization or sharing of mutable data aren't a good fit for MapReduce, because it's a shared-nothing architecture, which can pose challenges for some algorithms.

VERSION INCOMPATIBILITIES

The Hadoop 2 release brought with it some headaches with regard to MapReduce API runtime compatibility, especially in the `org.hadoop.mapreduce` package. These problems often result in runtime issues with code that's compiled against Hadoop 1 (and earlier). The solution is usually to recompile against Hadoop 2, or to consider a technique outlined in chapter 2 that introduces a compatibility library to target both Hadoop versions without the need to recompile code.

Other challenges with Hive and Hadoop also exist, where Hive may need to be recompiled to work with versions of Hadoop other than the one it was built against. Pig has had compatibility issues, too. For example, the Pig 0.8 release didn't work with Hadoop 0.20.203, and manual intervention was required to work around this problem. This is one of the advantages of using a Hadoop distribution other than Apache, as these compatibility problems have been fixed. If using the vanilla Apache distributions is desired, it's worth taking a look at Bigtop (http://bigtop.apache.org/), an Apache open source automated build and compliance system. It includes all of the major Hadoop ecosystem components and runs a number of integration tests to ensure they all work in conjunction with each other.

After tackling Hadoop's architecture and its weaknesses, you're probably ready to roll up your sleeves and get hands-on with Hadoop, so let's look at running the first example in this book.

1.2 Getting your hands dirty with MapReduce

This section shows you how to run a MapReduce job on your host.

> **Installing Hadoop and building the examples** To run the code example in this section, you'll need to follow the instructions in the appendix, which explain how to install Hadoop and download and run the examples bundled with this book.

[17] Although HDFS Federation in Hadoop 2 has introduced a way for multiple NameNodes to share file metadata, the fact remains that metadata is stored in memory.

Let's say you want to build an inverted index. MapReduce would be a good choice for this task because it can create indexes in parallel (a common MapReduce use case). Your input is a number of text files, and your output is a list of tuples, where each tuple is a word and a list of files that contain the word. Using standard processing techniques, this would require you to find a mechanism to join all the words together. A naive approach would be to perform this join in memory, but you might run out of memory if you have large numbers of unique keys. You could use an intermediary datastore, such as a database, but that would be inefficient.

A better approach would be to tokenize each line and produce an intermediary file containing a word per line. Each of these intermediary files could then be sorted. The final step would be to open all the sorted intermediary files and call a function for each unique word. This is what MapReduce does, albeit in a distributed fashion.

Figure 1.10 walks you through an example of a simple inverted index in MapReduce. Let's start by defining your mapper. Your reducers need to be able to generate a line for each word in your input, so your map output key should be each word in the input files so that MapReduce can join them all together. The value for each key will be the containing filename, which is your document ID.

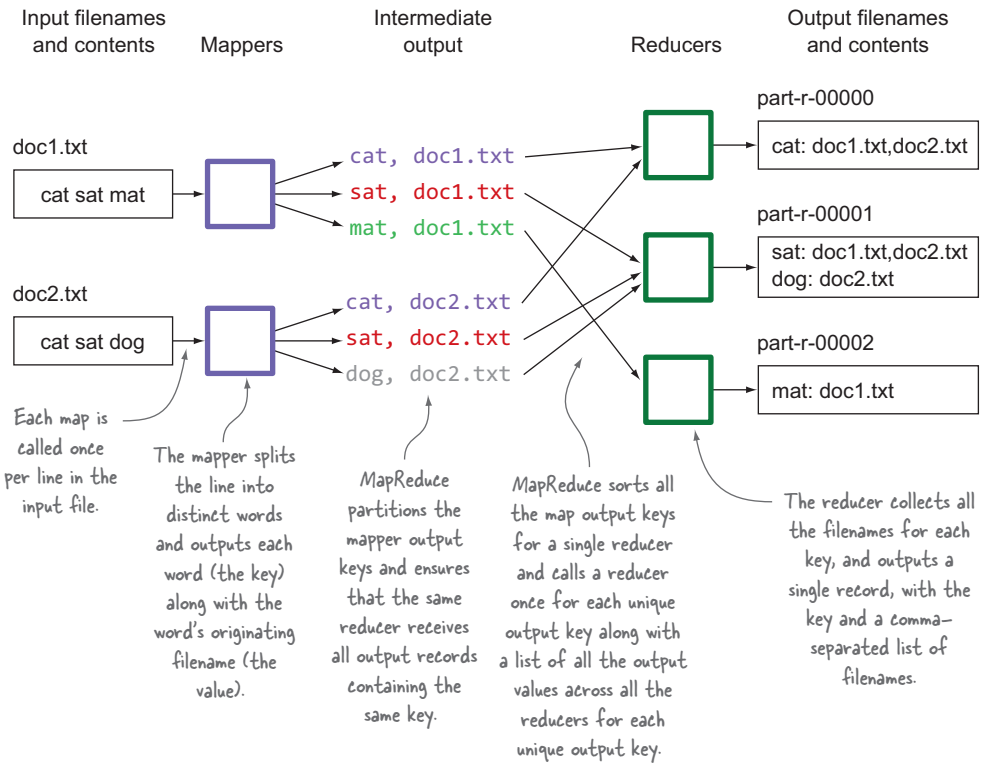
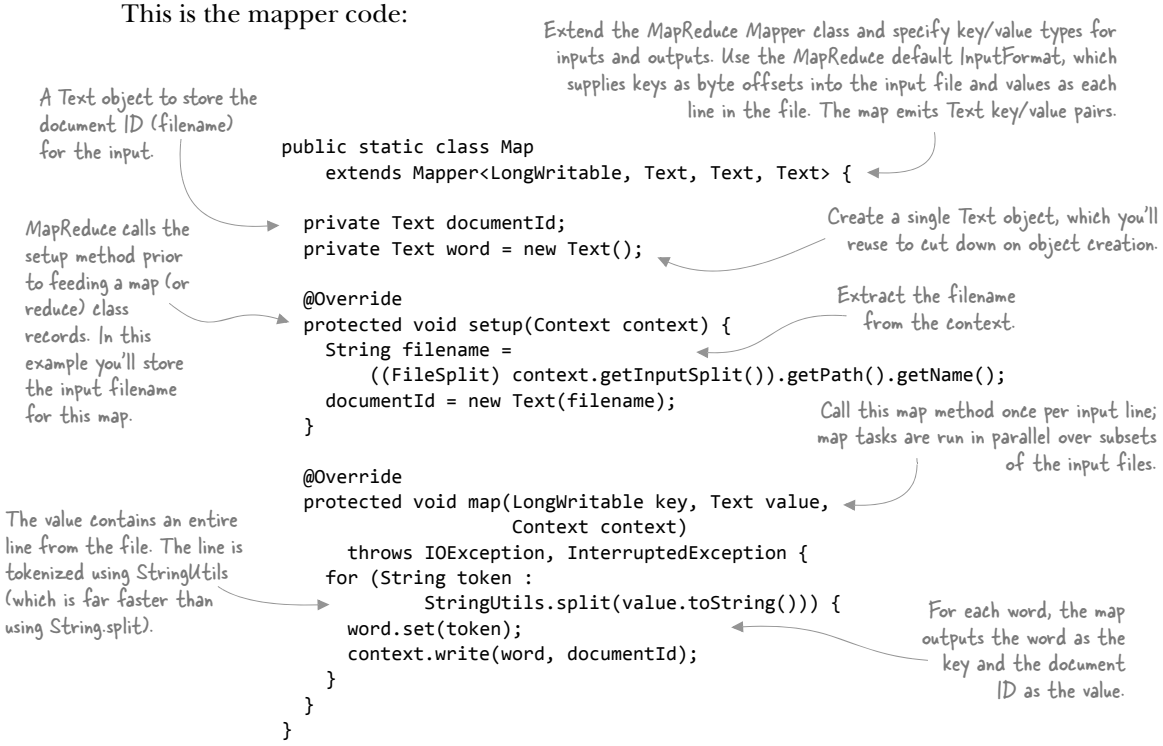


Figure 1.10 An example of an inverted index being created in MapReduce

This is the mapper code:



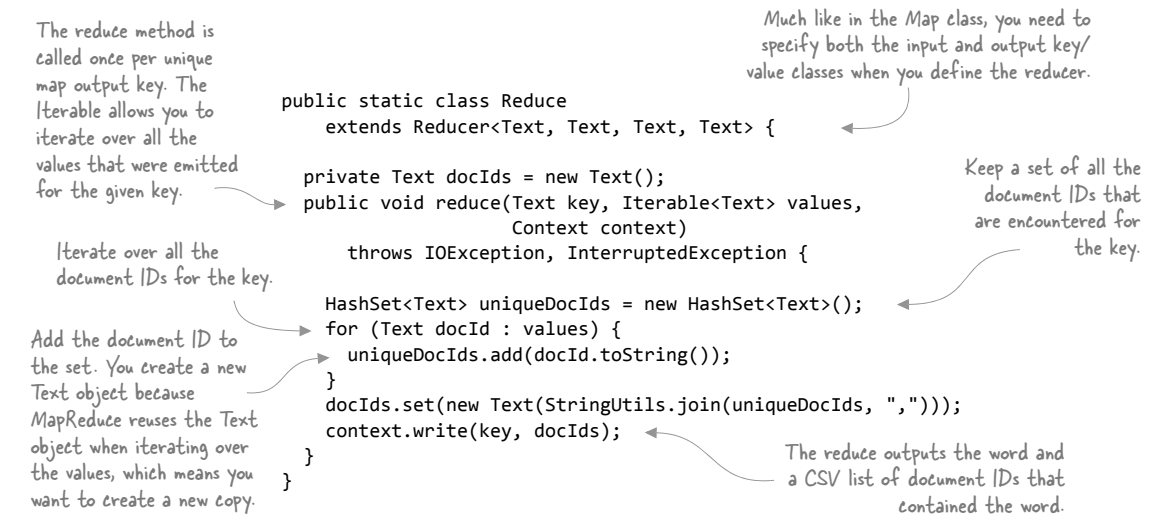

```
public static class Map
    extends Mapper<LongWritable, Text, Text, Text> {

  private Text documentId;
  private Text word = new Text();

  @Override
  protected void setup(Context context) {
    String filename =
        ((FileSplit) context.getInputSplit()).getPath().getName();
    documentId = new Text(filename);
  }

  @Override
  protected void map(LongWritable key, Text value,
                     Context context)
      throws IOException, InterruptedException {
    for (String token :
            StringUtils.split(value.toString())) {
      word.set(token);
      context.write(word, documentId);
    }
  }
}
```

The goal of this reducer is to create an output line for each word and a list of the document IDs in which the word appears. The MapReduce framework will take care of calling the reducer once per unique key outputted by the mappers, along with a list of document IDs. All you need to do in the reducer is combine all the document IDs together and output them once in the reducer, as you can see in the following code:




```
public static class Reduce
    extends Reducer<Text, Text, Text, Text> {

  private Text docIds = new Text();
  public void reduce(Text key, Iterable<Text> values,
                     Context context)
      throws IOException, InterruptedException {

    HashSet<Text> uniqueDocIds = new HashSet<Text>();
    for (Text docId : values) {
      uniqueDocIds.add(docId.toString());
    }
    docIds.set(new Text(StringUtils.join(uniqueDocIds, ",")));
    context.write(key, docIds);
  }
}
```

The last step is to write the driver code that will set all the necessary properties to configure the MapReduce job to run. You need to let the framework know what classes should be used for the map and reduce functions, and also let it know where the input and output data is located. By default, MapReduce assumes you're working with text; if you're working with more complex text structures, or altogether different data-storage technologies, you'll need to tell MapReduce how it should read and write from these data sources and sinks. The following source shows the full driver code:[18]

```
public int run(final String[] args) throws Exception {

  Cli cli = Cli.builder().setArgs(args)
              .addOptions(IOOptions.values()).build();
  cli.runCmd();

  Path input = new Path(cli.getArgValueAsString(IOOptions.INPUT));
  Path output = new Path(cli.getArgValueAsString(IOOptions.OUTPUT));

  Configuration conf = super.getConf();

  Job job = new Job(conf);
  job.setJarByClass(InvertedIndexJob.class);
  job.setMapperClass(Map.class);
  job.setReducerClass(Reduce.class);
  job.setMapOutputKeyClass(Text.class);
  job.setMapOutputValueClass(Text.class);

  FileInputFormat.setInputPaths(job, input);
  FileOutputFormat.setOutputPath(job, output);

  if (job.waitForCompletion(true)) {
    System.out.println("Job completed successfully.");

    return 0;
  }
  return 1;
}
```

Extract the input and output directories from the arguments.

Get a handle for the Configuration instance for the job.

Set the Map class that should be used for the job.

The Job class's setJarByClass informs MapReduce that the supplied class should be used to determine the encapsulating JAR, which in turn is added to the classpath of all your map and reduce tasks.

Set the Reduce class that should be used for the job.

If the map output key/value types differ from the input types, you must tell Hadoop what they are. Here, the map will output each word and file as key/value pairs, and both are Text objects.

Set the map output value class.

Set the HDFS input directory for the job.

Set the HDFS output directory for the job.

Tell the framework to run the job and block until the job has completed.

Let's see how this code works. First, you need to create two simple input files in HDFS:

```
$ hadoop fs -mkdir -p hip1/input
$ echo "cat sat mat" | hadoop fs -put - hip1/input/1.txt
$ echo "dog lay mat" | hadoop fs -put - hip1/input/2.txt
```

Create two files in HDFS to serve as inputs for the job.

Next, run the MapReduce code. You'll use a shell script to run it, supplying the two input files as arguments, along with the job output directory:

```
$ hip hip.ch1.InvertedIndexJob --input hip1/input --output hip1/output
```

[18] GitHub source: https://github.com/alexholmes/hiped2/blob/master/src/main/java/hip/ch1/InvertedIndexJob.java.

> **Executing code examples in the book** The appendix contains instructions for downloading and installing the binaries and code that accompany this book. Most of the examples are launched via the `hip` script, which is located inside the bin directory. For convenience, it's recommended that you add the book's bin directory to your path so that you can copy-paste all the example commands as is. The appendix has instructions on how to set up your environment.

When your job completes, you can examine HDFS for the job output files and view their contents:

```
$ hadoop fs -ls output/
Found 3 items
output/_SUCCESS
output/_logs
output/part-r-00000

$ hadoop fs -cat output/part*
cat    1.txt
dog    2.txt
lay    2.txt
mat    2.txt,1.txt
sat    1.txt
```

This completes your whirlwind tour of how to run Hadoop.

1.3 Chapter summary

Hadoop is a distributed system designed to process, generate, and store large datasets. Its MapReduce implementation provides you with a fault-tolerant mechanism for large-scale data analysis of heterogeneous structured and unstructured data sources, and YARN now supports multi-tenant disparate applications on the same Hadoop cluster.

In this chapter, we examined Hadoop from both functional and physical architectural standpoints. You also installed Hadoop and ran a MapReduce job.

The remainder of this book is dedicated to presenting real-world techniques for solving common problems you'll encounter when working with Hadoop. You'll be introduced to a broad spectrum of subject areas, starting with YARN, HDFS and MapReduce, and Hive. You'll also look at data-analysis techniques and explore technologies such as Mahout and Rhipe.

In chapter 2, the first stop on your journey, you'll discover YARN, which heralds a new era for Hadoop, one that transforms Hadoop into a distributed processing kernel. Without further ado, let's get started.