Scaling Big Data with Hadoop and Solr
Second Edition

Together, Apache Hadoop and Apache Solr help organizations resolve the problem of information extraction from big data by providing excellent distributed faceted search capabilities.

This book will help you learn everything you need to know to build a distributed enterprise search platform as well as optimize this search to a greater extent, resulting in the maximum utilization of available resources. Starting with the basics of Apache Hadoop and Solr, the book covers advanced topics of optimizing search with some interesting real-world use cases and sample Java code.

This is a step-by-step guide that will teach you how to build a high performance enterprise search while scaling data with Hadoop and Solr in an effortless manner.

Who this book is written for
This book is aimed at developers, designers, and architects who would like to build big data enterprise search solutions for their customers or organizations. No prior knowledge of Apache Hadoop and Apache Solr/Lucene technologies is required.

What you will learn from this book

- Understand Apache Hadoop, its ecosystem, and Apache Solr
- Explore industry-based architectures by designing a big data enterprise search with their applicability and benefits
- Integrate Apache Solr with big data technologies such as Cassandra to enable better scalability and high availability for big data
- Optimize the performance of your big data search platform with scaling data
- Write MapReduce tasks to index your data
- Configure your Hadoop instance to handle real-world big data problems
- Work with Hadoop and Solr using real-world examples to benefit from their practical usage
- Use Apache Solr as a NoSQL database

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Processing Big Data Using Hadoop and MapReduce'
- A synopsis of the book’s content
- More information on Scaling Big Data with Hadoop and Solr Second Edition

About the Author

Hrishikesh Vijay Karambelkar is an enterprise architect who has been developing a blend of technical and entrepreneurial experience for more than 14 years. His core expertise lies in working on multiple subjects, which include big data, enterprise search, semantic web, link data analysis, analytics, and he also enjoys architecting solutions for the next generation of product development for IT organizations. He spends most of his time at work, solving challenging problems faced by the software industry. Currently, he is working as the Director of Data Capabilities at The Digital Group.

In the past, Hrishikesh has worked in the domain of graph databases; some of his work has been published at international conferences, such as VLDB, ICDE, and others. He has also written Scaling Apache Solr, published by Packt Publishing. He enjoys travelling, trekking, and taking pictures of birds living in the dense forests of India. He can be reached at http://hrishikesh.karambelkar.co.in/.
Scaling Big Data with Hadoop and Solr
Second Edition

With the growth of information assets in enterprises, the need to build a rich, scalable search application that can handle a lot of data has becomes critical. Today, Apache Solr is one of the most widely adapted, scalable, feature-rich, and best performing open source search application servers. Similarly, Apache Hadoop is one of the most popular Big Data platforms and is widely preferred by many organizations to store and process large datasets.

Scaling Big Data with Hadoop and Solr, Second Edition is intended to help its readers build a high performance Big Data enterprise search engine with the help of Hadoop and Solr. This starts with a basic understanding of Hadoop and Solr, and gradually develops into building an efficient, scalable enterprise search repository for Big Data, using various techniques throughout the practical chapters.

What This Book Covers

Chapter 1, Processing Big Data Using Hadoop and MapReduce, introduces you to Apache Hadoop and its ecosystem, HDFS and MapReduce. You will also learn how to write MapReduce programs, configure Hadoop clusters, configuration files, and administrate your cluster.

Chapter 2, Understanding Apache Solr, introduces you to Apache Solr. It explains how you can configure the Solr instance, how to create indexes and load your data in the Solr repository, and how you can use Solr effectively to search. It also discusses interesting features of Apache Solr.

Chapter 3, Enabling Distributed Search using Apache Solr, takes you through various aspects of enabling Solr for a distributed search, including with the use of SolrCloud. It also explains how Apache Solr and Big Data can come together to perform a scalable search.

Chapter 4, Big Data Search Using Hadoop and Its Ecosystem, explains the NoSQL and concepts of distributed search. It then explains how to use different algorithms for Big Data search, and includes covering shards and indexing. It also talks about integration with Cassandra, Apache Blur, Storm, and search analytics.
Chapter 5, *Scaling Search Performance*, will guide you in improving the performance of searches with Scaling Big Data. It covers different levels of optimization that you can perform on your Big Data search instance as the data keeps growing. It discusses different performance improvement techniques that can be implemented by users for the purposes of deployment.

Appendix, *Use Cases for Big Data Search*, discusses some of the most important business cases for high-level enterprise search architecture with Big Data and Solr.
Continuous evolution in computer sciences has enabled the world to work in a faster, more reliable, and more efficient manner. Many businesses have been transformed to utilize electronic media. They use information technologies to innovate the communication with their customers, partners, and suppliers. It has also given birth to new industries such as social media and e-commerce. This rapid increase in the amount of data has led to an "information explosion." To handle the problems of managing huge information, the computational capabilities have evolved too, with a focus on optimizing the hardware cost, giving rise to distributed systems. In today's world, this problem has multiplied; information is generated from disparate sources such as social media, sensors/embedded systems, and machine logs, in either a structured or an unstructured form. Processing of these large and complex data using traditional systems and methods is a challenging task. Big Data is an umbrella term that encompasses the management and processing of such data.

Big data is usually associated with high-volume and heavily growing data with unpredictable content. The IT advisory firm Gartner defines big data using 3Vs (high volume of data, high velocity of processing speed, and high variety of information). IBM has added a fourth V (high veracity) to this definition to make sure that the data is accurate and helps you make your business decisions. While the potential benefits of big data are real and significant, there remain many challenges. So, organizations that deal with such a high volumes of data, must work on the following areas:

- Data capture/acquisition from various sources
- Data massaging or curating
- Organization and storage
Big data processing such as search, analysis, and querying
• Information sharing or consumption
• Information security and privacy

Big data poses a lot of challenges to the technologies in use today. Many organizations have started investing in these big data areas. As per Gartner, through 2015, 85% of the Fortune 500 organizations will be unable to exploit big data for a competitive advantage.

To handle the problem of storing and processing complex and large data, many software frameworks have been created to work on the big data problem. Among them, Apache Hadoop is one of the most widely used open source software frameworks for the storage and processing of big data. In this chapter, we are going to understand Apache Hadoop. We will be covering the following topics:

• Apache Hadoop's ecosystem
• Configuring Apache Hadoop
• Running Apache Hadoop
• Setting up a Hadoop cluster

**Apache Hadoop's ecosystem**

Apache Hadoop enables the distributed processing of large datasets across a commodity of clustered servers. It is designed to scale up from a single server to thousands of commodity hardware machines, each offering partial computational units and data storage.

The Apache Hadoop system comes with the following primary components:

• **Hadoop Distributed File System (HDFS)**
• MapReduce framework

The Apache Hadoop distributed file system or HDFS provides a file system that can be used to store data in a replicated and distributed manner across various nodes, which are part of the Hadoop cluster. Apache Hadoop provides a distributed data processing framework for large datasets by using a simple programming model called MapReduce.
A programming task that takes a set of data (key-value pair) and converts it into another set of data, is called **Map Task**. The results of map tasks are combined into one or many **Reduce Tasks**. Overall, this approach towards computing tasks is called the **MapReduce approach**.

The MapReduce programming paradigm forms the heart of the Apache Hadoop framework, and any application that is deployed on this framework must comply with MapReduce programming. The following figure demonstrates how MapReduce can be used to sort input documents with the MapReduce approach:

MapReduce can also be used to transform data from a domain into the corresponding range. We are going to look at these in more detail in the following chapters.
Hadoop has been used in environments where data from various sources needs to be processed using large server farms. Hadoop is capable of running its cluster of nodes on commodity hardware, and does not demand any high-end server configuration. With this, Hadoop also brings scalability that enables administrators to add and remove nodes dynamically. Some of the most notable users of Hadoop are companies like Google (in the past), Facebook, and Yahoo, who process petabytes of data every day, and produce rich analytics to the consumer in the shortest possible time. All this is supported by a large community of users who consistently develop and enhance Hadoop every day. Apache Hadoop 2.0 onwards uses YARN (which stands for Yet Another Resource Negotiator).

The Apache Hadoop 1.X MapReduce framework used concepts of job tracker and task tracker. If you are using the older Hadoop versions, it is recommended to move to Hadoop 2.x, which uses advanced MapReduce (also called 2.0). This was released in 2013.

Core components
The following diagram demonstrates how the core components of Apache Hadoop work together to ensure distributed execution of user jobs:
Chapter 1

The Resource Manager (RM) in a Hadoop system is responsible for globally managing the resources of a cluster. Besides managing resources, it coordinates the allocation of resources on the cluster. RM consists of Scheduler and ApplicationsManager. As the names suggest, Scheduler provides resource allocation, whereas ApplicationsManager is responsible for client interactions (accepting jobs and identifying and assigning them to Application Masters).

The Application Master (AM) works for a complete application lifecycle, that is, the life of each MapReduce job. It interacts with RM to negotiate for resources.

The Node Manager (NM) is responsible for the management of all containers that run on a given node. It keeps a watch on resource usage (CPU, memory, and so on), and reports the resource health consistently to the resource manager.

All the metadata related to HDFS is stored on NameNode. The NameNode is the master node that performs coordination activities among data nodes, such as data replication across data nodes, naming system such as filenames, and the disk locations. NameNode stores the mapping of blocks on the Data Nodes. In a Hadoop cluster, there can only be one single active NameNode. NameNode regulates access to its file system with the use of HDFS-based APIs to create, open, edit, and delete HDFS files.

Earlier, NameNode, due to its functioning, was identified as the single point of failure in a Hadoop system. To compensate for this, the Hadoop framework introduced SecondaryNameNode, which constantly syncs with NameNode and can take over whenever NameNode is unavailable.

DataNodes are nothing but slaves that are deployed on all the nodes in a Hadoop cluster. DataNode is responsible for storing the application's data. Each uploaded data file in HDFS is split into multiple blocks, and these data blocks are stored on different data nodes. The default file block size in HDFS is 64 MB. Each Hadoop file block is mapped to two files in the data node; one file is the file block data, while the other is checksum.

When Hadoop is started, each DataNode connects to NameNode informing it of its availability to serve the requests. When the system is started, the namespace ID and software versions are verified by NameNode and DataNode sends the block report describing all the data blocks it holds for NameNode on startup. During runtime, each DataNode periodically sends a heartbeat signal to NameNode, confirming its availability. The default duration between two heartbeats is 3 seconds. NameNode assumes the unavailability of DataNode if it does not receive a heartbeat in 10 minutes by default; in which case, NameNode replicates the data blocks of that DataNode to other DataNodes.
When a client submits a job to Hadoop, the following activities take place:

1. Application manager launches AM to a given client job/application after negotiating with a specific node.
2. The AM, once booted, registers itself with the RM. All the client communication with AM happens through RM.
3. AM launches the container with help of NodeManager.
4. A container that is responsible for executing a MapReduce task reports the progress status to the AM through an application-specific protocol.
5. On receiving any request for data access on HDFS, NameNode takes the responsibility of returning to the nearest location of DataNode from its repository.

Understanding Hadoop's ecosystem

Although Hadoop provides excellent storage capabilities along with the MapReduce programming framework, it is still a challenging task to transform conventional programming into a MapReduce type of paradigm, as MapReduce is a completely different programming paradigm. The Hadoop ecosystem is designed to provide a set of rich applications and development framework. The following block diagram shows Apache Hadoop's ecosystem:
We have already seen MapReduce, HDFS, and YARN. Let us look at each of the blocks.

HDFS is an append-only file system; it does not allow data modification. **Apache HBase** is a distributed, random-access, and column-oriented database. HBase directly runs on top of HDFS and allows application developers to read-write the HDFS data directly. HBase does not support SQL; hence, it is also called a **NoSQL** database. However, it provides a command line-based interface, as well as a rich set of APIs to update the data. The data in HBase gets stored as key-value pairs in HDFS.

**Apache Pig** provides another abstraction layer on top of MapReduce. It's a platform for the analysis of very large datasets that runs on HDFS. It also provides an infrastructure layer, consisting of a compiler that produces sequences of MapReduce programs, along with a language layer consisting of the query language Pig Latin. Pig was initially developed at Yahoo! Research to enable developers to create ad-hoc MapReduce jobs for Hadoop. Since then, many big organizations such as eBay, LinkedIn, and Twitter have started using Apache Pig.

**Apache Hive** provides data warehouse capabilities using big data. Hive runs on top of Apache Hadoop and uses HDFS for storing its data. The Apache Hadoop framework is difficult to understand, and requires a different approach from traditional programming to write MapReduce-based programs. With Hive, developers do not write MapReduce at all. Hive provides an SQL-like query language called **HiveQL** to application developers, enabling them to quickly write ad-hoc queries similar to RDBMS SQL queries.

Apache Hadoop nodes communicate with each other through **Apache ZooKeeper**. It forms a mandatory part of the Apache Hadoop ecosystem. Apache ZooKeeper is responsible for maintaining co-ordination among various nodes. Besides coordinating among nodes, it also maintains configuration information and the group services to the distributed system. Apache ZooKeeper can be used independent of Hadoop, unlike other components of the ecosystem. Due to its in-memory management of information, it offers distributed co-ordination at a high speed.

**Apache Mahout** is an open source machine learning software library that can effectively empower Hadoop users with analytical capabilities, such as clustering and data mining, over a distributed Hadoop cluster. Mahout is highly effective over large datasets; the algorithms provided by Mahout are highly optimized to run the MapReduce framework over HDFS.
Apache HCatalog provides metadata management services on top of Apache Hadoop. It means that all the software that runs on Hadoop can effectively use HCatalog to store the corresponding schemas in HDFS. HCatalog helps any third-party software to create, edit, and expose (using REST APIs) the generated metadata or table definitions. So, any users or scripts can run on Hadoop effectively without actually knowing where the data is physically stored on HDFS. HCatalog provides DDL (which stands for Data Definition Language) commands with which the requested MapReduce, Pig, and Hive jobs can be queued for execution, and later monitored for progress as and when required.

Apache Ambari provides a set of tools to monitor the Apache Hadoop cluster, hiding the complexities of the Hadoop framework. It offers features such as installation wizard, system alerts and metrics, provisioning and management of the Hadoop cluster, and job performances. Ambari exposes RESTful APIs to administrators to allow integration with any other software. Apache Oozie is a workflow scheduler used for Hadoop jobs. It can be used with MapReduce as well as Pig scripts to run the jobs. Apache Chukwa is another monitoring application for distributed large systems. It runs on top of HDFS and MapReduce.

Apache Sqoop is a tool designed to load large datasets into Hadoop efficiently. Apache Sqoop allows application developers to import/export easily from specific data sources, such as relational databases, enterprise data warehouses, and custom applications. Apache Sqoop internally uses a map task to perform data import/export effectively on a Hadoop cluster. Each mapper loads/unloads a slice of data across HDFS and a data source. Apache Sqoop establishes connectivity between non-Hadoop data sources and HDFS.

Apache Flume provides a framework to populate Hadoop with data from non-conventional data sources. Typical usage of Apache Fume could be for log aggregation. Apache Flume is a distributed data collection service that extracts data from the heterogeneous sources, aggregates the data, and stores it into the HDFS. Most of the time, Apache Flume is used as an ETL (which stands for Extract-Transform-Load) utility at various implementations of the Hadoop cluster.

Configuring Apache Hadoop

Setting up a Hadoop cluster is a step-by-step process. It is recommended to start with a single node setup and then extend it to the cluster mode. Apache Hadoop can be installed with three different types of setup:

- **Single node setup**: In this mode, Hadoop can be set up on a single standalone machine. This mode is used by developers for evaluation, testing, basic development, and so on.
Chapter 1

- **Pseudo distributed setup**: Apache Hadoop can be set up on a single machine with a distributed configuration. In this setup, Apache Hadoop can run with multiple Hadoop processes (daemons) on the same machine. Using this mode, developers can do the testing for a distributed setup on a single machine.

- **Fully distributed setup**: In this mode, Apache Hadoop is set up on a cluster of nodes, in a fully distributed manner. Typically, production-level setups use this mode for actively using the Hadoop computing capabilities.

In Linux, Apache Hadoop can be set up through the root user, which makes it globally available, or as a separate user, which makes it available to only that user (Hadoop user), and the access can later be extended for other users. It is better to use a separate user with limited privileges to ensure that the Hadoop runtime does not have any impact on the running system.

### Prerequisites

Before setting up a Hadoop cluster, it is important to ensure that all prerequisites are addressed. Hadoop runs on the following operating systems:

- All Linux Flavors are supported for development as well as production.
- In the case of Windows, Microsoft Windows 2008 onwards are supported. Apache Hadoop version 2.2 onwards support Windows. The older versions of Hadoop have limited support through Cygwin.

Apache Hadoop requires the following software:

- Java 1.6 onwards are all supported; however, there are compatibility issues, so it is best to look at Hadoop's Java compatibility wiki page at [http://wiki.apache.org/Hadoop/HadoopJavaVersions](http://wiki.apache.org/Hadoop/HadoopJavaVersions).

- **Secure shell (ssh)** is needed to run start, stop, status, or other such scripts across a cluster. You may also consider using parallel-ssh (more information is available at [https://code.google.com/p/parallel-ssh/](https://code.google.com/p/parallel-ssh/)) for connectivity.

Apache Hadoop can be downloaded from [http://www.apache.org/dyn/closer.cgi/Hadoop/common/](http://www.apache.org/dyn/closer.cgi/Hadoop/common/). Make sure that you download and choose the correct release from different releases, that is, one that is a stable release, the latest beta/alpha release, or a legacy stable version. You can choose to download the package or download the source, compile it on your OS, and then install it. Using operating system package installer, install the Hadoop package. This software can be installed directly by using `apt-get/dpkg` for Ubuntu/Debian or `rpm` for Red Hat/Oracle Linux from the respective sites. In the case of a cluster setup, this software should be installed on all the machines.
Setting up ssh without passphrase

Apache Hadoop uses ssh to run its scripts on different nodes, it is important to make this ssh login happen without any prompt for password. If you already have a key generated, then you can skip this step. To make ssh work without a password, run the following commands:

```
$ ssh-keygen -t dsa
```

You can also use RSA-based encryption algorithm (link to know about RSA: http://en.wikipedia.org/wiki/RSA_%28cryptosystem%29 instead of DSA (Digital Signature Algorithm) for your ssh authorization key creation. (For more information about differences between these two algorithms, visit http://security.stackexchange.com/questions/5096/rsa-vs-dsa-for-ssh-authentication-keys. Keep the default file for saving the key, and do not enter a passphrase. Once the key generation is successfully complete, the next step is to authorize the key by running the following command:

```
$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

This step will actually create an authorization key with ssh, bypassing the passphrase check as shown in the following screenshot:
Once this step is complete, you can `ssh localhost` to connect to your instance without password. If you already have a key generated, you will get a prompt to overwrite it; in such a case, you can choose to overwrite it or you can use the existing key and put it in the `authorized_keys` file.

## Configuring Hadoop

Most of the Hadoop configuration is specified in the following configuration files, kept in the `$HADOOP_HOME/etc/Hadoop` folder of the installation. `$HADOOP_HOME` is the place where Apache Hadoop has been installed. If you have installed the software by using the pre-build package installer as the root user, the configuration can be found at `/etc/Hadoop`.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>core-site.xml</td>
<td>In this file, you can modify the default properties of Hadoop. This covers setting up different protocols for interaction, working directories, log management, security, buffers and blocks, temporary files, and so on.</td>
</tr>
<tr>
<td>hdfs-site.xml</td>
<td>This file stores the entire configuration related to HDFS. So, properties like DFS site address, data directory, replication factors, and so on are covered in these files.</td>
</tr>
<tr>
<td>mapred-site.xml</td>
<td>This file is responsible for handling the entire configuration related to the MapReduce framework. This covers the configuration for JobTracker and TaskTracker properties for Job.</td>
</tr>
<tr>
<td>yarn-site.xml</td>
<td>This file is required for managing YARN-related configuration. This configuration typically contains security/access information, proxy configuration, resource manager configuration, and so on.</td>
</tr>
<tr>
<td>httpfs-site.xml</td>
<td>Hadoop supports REST-based data transfer between clusters through an HttpFS server. This file is responsible for storing configuration related to the HttpFS server.</td>
</tr>
<tr>
<td>fair-scheduler.xml</td>
<td>This file contains information about user allocations and pooling information for the fair scheduler. It is currently under development.</td>
</tr>
<tr>
<td>capacity-scheduler.xml</td>
<td>This file is mainly used by the RM in Hadoop for setting up the scheduling parameters of job queues.</td>
</tr>
<tr>
<td>Hadoop-env.sh or Hadoop-env.cmd</td>
<td>All the environment variables are defined in this file; you can change any of the environments: namely the Java location, Hadoop configuration directory, and so on.</td>
</tr>
<tr>
<td>mapred-env.sh or mapred-env.cmd</td>
<td>This file contains the environment variables used by Hadoop while running MapReduce.</td>
</tr>
</tbody>
</table>
### File Name Description

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yarn-env.sh or yarn-env.cmd</td>
<td>This file contains the environment variables used by the YARN daemon that starts/stops the node manager and the RM.</td>
</tr>
<tr>
<td>httpfs-env.sh or httpfs-env.cmd</td>
<td>This file contains environment variables required by the HttpFS server.</td>
</tr>
<tr>
<td>Hadoop-policy.xml</td>
<td>This file is used to define various access control lists for Hadoop services. It controls who can use the Hadoop cluster for execution.</td>
</tr>
<tr>
<td>Masters/slaves</td>
<td>In this file, you can define the hostname for the masters and the slaves. The masters file lists all the masters, and the slaves file lists the slave nodes. To run Hadoop in the cluster mode, you need to modify these files to point to the respective master and slaves on all nodes.</td>
</tr>
<tr>
<td>log4j.properties</td>
<td>You can define various log levels for your instance; this is helpful while developing or debugging Hadoop programs. You can define levels for logging.</td>
</tr>
<tr>
<td>common-logging.properties</td>
<td>This file specifies the default logger used by Hadoop; you can override it to use your logger.</td>
</tr>
</tbody>
</table>

The file names marked in *pink italicized* letters will be modified while setting up your basic Hadoop cluster.

Now, let's start with the configuration of these files for the first Hadoop run. Open `core-sites.xml`, and add the following entry in it:

```xml
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```
This snippet tells the Hadoop framework to run inter-process communication on port 9000. Next, edit hdfs-site.xml and add the following entries:

```xml
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
```

This tells HDFS to have the distributed file system's replication factor as 1. Later when you run Hadoop in the cluster configuration, you can change this replication count. The choice of replication factor varies from case to case, but if you are not sure about it, it is better to keep it as 3. This means that each document will have a replication of factor of 3.

Let's start looking at the MapReduce configuration. Some applications such as Apache HBase use only HDFS for storage, and they do not rely on the MapReduce framework. This means that all they require is the HDFS configuration, and the next configuration can be skipped.

Now, edit mapred-site.xml and add the following entries:

```xml
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

This entry points to YARN as the MapReduce framework used. Further, modify yarn-site.xml with the following entries:

```xml
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```
This entry enables YARN to use the ShuffleHandler service with nodemanager. Once the configuration is complete, we are good to start the Hadoop. Here are the default ports used by Apache Hadoop:

<table>
<thead>
<tr>
<th>Particular</th>
<th>Default Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFS Port</td>
<td>9000/8020</td>
</tr>
<tr>
<td>NameNode – Web Application</td>
<td>50070</td>
</tr>
<tr>
<td>Data Node</td>
<td>50075</td>
</tr>
<tr>
<td>Secondary NameNode</td>
<td>50090</td>
</tr>
<tr>
<td>Resource Manager Web App.</td>
<td>8088</td>
</tr>
</tbody>
</table>

**Running Hadoop**

Before setting up the HDFS, we must ensure that Hadoop is configured for the pseudo-distributed mode, as per the previous section, that is, Configuring Hadoop. Set up the JAVA_HOME and HADOOP_PREFIX environment variables in your profile before you proceed. To set up a single node configuration, first you will be required to format the underlying HDFS file system; this can be done by running the following command:

```bash
$ $HADOOP_PREFIX/bin/hdfs namenode -format
```

Once the formatting is complete, simply try running HDFS with the following command:

```bash
$ $HADOOP_PREFIX/sbin/start-dfs.sh
```

The `start-dfs.sh` script file will start the name node, data node, and secondary name node on your machine through ssh. The Hadoop daemon log output is written to the `$HADOOP_LOG_DIR` folder, which by default points to `$HADOOP_HOME/logs`. Once the Hadoop daemon starts running, you will find three different processes running when you check the snapshot of the running processes. Now, browse the web interface for the NameNode; by default, it is available at `http://localhost:50070/`. You will see a web page similar to the one shown as follows with the HDFS information:
Once the HDFS is set and started, you can use all Hadoop commands to perform file system operations. The next job is to start the MapReduce framework, which includes the node manager and RM. This can be done by running the following command:

```
$ $HADOOP_PREFIX/bin/start-yarn.sh
```
You can access the RM web page by accessing http://localhost:8088/.
The following screenshot shows a newly set-up Hadoop RM page.

We are good to use this Hadoop setup for development now.

**Safe Mode**

When a cluster is started, NameNode starts its complete functionality only when the configured minimum percentage of blocks satisfies the minimum replication. Otherwise, it goes into safe mode. When NameNode is in the safe mode state, it does not allow any modification to its file systems. This mode can be turned off manually by running the following command:

$ Hadoop dfsadmin – safemode leave

You can test the instance by running the following commands:

This command will create a test folder, so you need to ensure that this folder is not present on a server instance:

$ bin/Hadoop dfs –mkdir /test

This will create a folder. Now, load some files by using the following command:

$ bin/Hadoop dfs -put <file-location> test/input

Now, run the shipped example of wordcount that is packaged with the Hadoop deployment:

$ bin/Hadoop jar share/Hadoop/mapreduce/Hadoop-mapreduce-examples-2.5.1.jar test/input test/output
A successful run will create the output in HDFS's `test/output/part-r-00000` file. You can view the output by downloading this file from HDFS to a local machine.

**Setting up a Hadoop cluster**

In this case, assuming that you already have a single node setup as explained in the previous sections, with ssh being enabled, you just need to change all the slave configurations to point to the master. This can be achieved by first introducing the `slaves` file in the `$HADOOP_PREFIX/etc/Hadoop` folder. Similarly, on all slaves, you require the `master` file in the `$HADOOP_PREFIX/etc/Hadoop` folder to point to your master server hostname.

While adding new entries for the hostname, one must ensure that the firewall is disabled to allow remote nodes access to different ports. Alternatively, specific ports can be opened/modified by modifying the Hadoop configuration files. Similarly, all the names of nodes that are participating in the cluster should be resolvable through DNS (which stands for Domain Name System), or through the `/etc/hosts` entries of Linux.

Once this is ready, let us change the configuration files. Open `core-sites.xml`, and add the following entry in it:

```xml
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://master-server:9000</value>
    </property>
</configuration>
```

All other configuration is optional. Now, run the servers in the following order: First, you need to format your storage for the cluster; use the following command to do so:

```
$ $HADOOP_PREFIX/bin/Hadoop dfs namenode -format <Name of Cluster>
```

This formats the name node for a new cluster. Once the name node is formatted, the next step is to ensure that DFS is up and connected to each node. Start `namenode`, followed by the data nodes:

```
$ $HADOOP_PREFIX/sbin/Hadoop-daemon.sh start namenode

$ $HADOOP_PREFIX/sbin/Hadoop-daemon.sh start datanode
```

Similarly, the datanode can be started from all the slaves.

```
$ $HADOOP_PREFIX/sbin/Hadoop-daemon.sh start datanode
```
Processing Big Data Using Hadoop and MapReduce

Keep track of the log files in the $HADOOP_PREFIX/logs folder in order to see that there are no exceptions. Once the HDFS is available, namenode can be accessed through the web as shown here:

The next step is to start YARN and its associated applications. First, start with the RM:

$ $HADOOP_YARN_HOME/sbin/yarn-daemon.sh start resourcemanager

Each node must run an instance of one node manager. To run the node manager, use the following command:

$ $HADOOP_YARN_HOME/sbin/yarn-daemon.sh start nodemanager

Optionally, you can also run Job History Server on the Hadoop cluster by using the following command:

$ $HADOOP_PREFIX/sbin/mr-jobhistory-daemon.sh start historyserver
Once all instances are up, you can see the status of the cluster on the web through the RM UI as shown in the following screenshot. The complete setup can be tested by running the simple wordcount example.

This way, your cluster is set up and is ready to run with multiple nodes. For advanced setup instructions, do visit the Apache Hadoop website at http://Hadoop.apache.org.

Common problems and their solutions

The following is a list of common problems and their solutions:

- **When I try to format the HDFS node, I get the exception java. io.IOException: Incompatible clusterIDs in namenode and datanode?**
  
  This issue usually appears if you have a different/older cluster and you are trying to format a new namenode; however, the datanodes still point to older cluster ids. This can be handled by one of the following:

  1. By deleting the DFS data folder, you can find the location from hdfs-site.xml and restart the cluster
  2. By modifying the version file of HDFS usually located at `<HDFS-STORAGE-PATH>/hdfs/datanode/current/`
  3. By formatting namenode with the problematic datanode's cluster ID:

     ```bash
     $ hdfs namenode -format -clusterId <cluster-id>
     ```
• My Hadoop instance is not starting up with the ./start-all.sh script? When I try to access the web application, it shows the page not found error?
This could be happening because of a number of issues. To understand the issue, you must look at the Hadoop logs first. Typically, Hadoop logs can be accessed from the /var/log folder if the precompiled binaries are installed as the root user. Otherwise, they are available inside the Hadoop installation folder.

• I have setup N node clusters, and I am running the Hadoop cluster with ./start-all.sh. I am not seeing many nodes in the YARN/NameNode web application?
This again can be happening due to multiple reasons. You need to verify the following:
1. Can you reach (connect to) each of the cluster nodes from namenode by using the IP address/machine name? If not, you need to have an entry in the /etc/hosts file.
2. Is the ssh login working without password? If not, you need to put the authorization keys in place to ensure logins without password.
3. Is datanode/nodemanager running on each of the nodes, and can you connect to namenode/AM? You can validate this by running ssh on the node running namenode/AM.
4. If all these are working fine, you need to check the logs and see if there are any exceptions as explained in the previous question.
5. Based on the log errors/exceptions, specific action has to be taken.

Summary
In this chapter, we discussed the need for Apache Hadoop to address the challenging problems faced by today's world. We looked at Apache Hadoop and its ecosystem, and we focused on how to configure Apache Hadoop, followed by running it. Finally, we created Hadoop clusters by using a simple set of instructions. The next chapter is all about Apache Solr, which has brought a revolution in the search and analytics domain.
Where to buy this book

You can buy Scaling Big Data with Hadoop and Solr Second Edition from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.