Part

I

What Is Big Data?

In This Part

CHAPTER

1

Industry Needs and Solutions

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Finding Out What Constitutes "Big Data"
- Appreciating the History and Origins of Hadoop
- Defining Hadoop
- Understanding the Core Components of Hadoop
- Looking to the Future with Hadoop 2.0

This first chapter introduces you to the open source world of Apache and to Hadoop, one of the most exciting and innovative platforms ever created for the data professional. In this chapter we're going to go on a bit of a journey. You're going to find out what inspired Hadoop, where it came from, and its future direction. You'll see how from humble beginnings two gentlemen have inspired a generation of data professionals to think completely differently about data processing and data architecture.

Before we look into the world of Hadoop, though, we must first ask ourselves an important question. Why does big data exist? Is this name just a fad, or is there substance to all the hype? Is big data here to stay? If you want to know the answers to these questions and a little more, read on. You have quite a journey in front of you…

What's So *Big* About Big Data?

The world has witnessed explosive, exponential growth in recent times. So, did we suddenly have a need for big data? Not exactly. Businesses have been tackling the capacity challenge for many years (much to the delight of storage hardware vendors). Therefore the *big* in big data isn't purely a statement on size.

Likewise, on the processing front, scale-out solutions such as high-performance computing and distributed database technology have been in place since the last millennium. There is nothing intrinsically new there either.

People also often talk about unstructured data, but, really, this just refers to the format of the data. Could this be a reason we "suddenly" need big data? We know that web data, especially web log data, is born in an unstructured format and can be generated in significant quantities and volume. However, is this really enough to be considered big data?

In my mind, the answer is no. No one property on its own is sufficient for a project or a solution to be considered a big data solution. It's only when you have a cunning blend of these ingredients that you get to bake a big data cake.

This is in line with the Gartner definition of big data, which they updated in Doug Laney's publication, *The Importance of Big Data: A Definition* (Gartner, 2012): "High volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization."

What we do know is that every CIO on the planet seems to want to start a big data project right now. In a world of shrinking budgets, there is this sudden desire to jump in with both feet into this world of big data and start prospecting for golden nuggets. It's the gold rush all over again, and clearly companies feel like they might miss out if they hesitate.

However, this is a picture that has been sharpening its focus for several years. In the buildup to this ubiquitous acceptance of big data, we've been blessed with plenty of industry terms and trends, web scale, new programming paradigms of "code first," and of course, to the total disgust of data modelers everywhere, NoSQL. Technologies such as Cassandra and MongoDB are certainly part of the broader ecosystem, but none have resonated as strongly with the market as Hadoop and big data. Why? In essence, unless you were Facebook, Google, Yahoo!, or Bing, issues like web scale really didn't apply.

It seems as though everyone is now building analytics platforms, and that, to be the king of geek chic, requires a degree in advanced statistics. The reason? Big data projects aren't defined by having big data sets. They are shaped by big ideas, by big questions, and by big opportunities. Big data is not about one technology or even one platform. It's so much more than that: It's a mindset and a movement.

Big data, therefore, is a term that underpins a raft of technologies (including the various Hadoop projects, NoSQL offerings, and even MPP Database Systems, for example) that have been created in the drive to better analyze and derive meaning from data at a dramatically lower cost and while delivering new insights and products for organizations all over the world. In times of recession, businesses look to derive greater value from the assets they have rather than invest in new assets. Big data, and in particular Hadoop, is the perfect vehicle for doing exactly that.

A Brief History of Hadoop

Necessity is the mother of invention, and Hadoop is no exception. Hadoop was created to meet the need of web companies to index and process the data tsunami courtesy of the newfangled Internetz. Hadoop's origins owe everything to both Google and the Apache Nutch project. Without one influencing the other, Hadoop might have ended up a very different animal (joke intended). In this next section, we are going to see how their work contributed to making Hadoop what it is today.

Google

As with many pioneering efforts, Google provided significant inspiration for the development that became known as Hadoop. Google published two landmark papers. The first paper, published in October 2003, was titled "The Google File System," and the second paper, "MapReduce: Simplified Data Processing on Large Clusters," published just over a year later in December 2004, provided the inspiration to Doug Cutting and his team of part-time developers for their project, Nutch.

MapReduce was first designed to enable Google developers to focus on the large-scale computations that they were trying to perform while abstracting away all the scaffolding code required to make the computation possible. Given the size of the data set they were working on and the duration of tasks, the developers knew that they had to have a model that was highly parallelized, was fault tolerant, and was able to balance the workload across a distributed set of machines. Of course, the Google implementation of MapReduce worked over Google File System (GFS); Hadoop Distributed File System (HDFS) was still waiting to be invented.

Google has since continued to release thought-provoking, illuminating, and inspirational publications. One publication worthy of note is "BigTable: A Distributed Storage System for Structured Data." Of course, they aren't the only ones. LinkedIn, Facebook, and of course Yahoo! have all contributed to the big data mind share.

There are similarities here to the SIGMOD papers published by various parties in the relational database world, but ultimately it isn't the same. Let's look at an example. Twitter has open-sourced Storm—their complex event processing engine—which has recently been accepted into the Apache incubator program. For relational database vendors, this level of open sharing is really quite unheard of. For more details about storm head over to Apache: `http://incubator.apache.org/projects/storm.html`.

Nutch

Nutch was an open source crawler-based search engine built by a handful of part-time developers, including Doug Cutting. As previously mentioned Cutting was inspired by the Google publications and changed Nutch to take advantage of the enhanced scalability of the architecture promoted by Google. However, it wasn't too long after this that Cutting joined Yahoo! and Hadoop was born.

Nutch joined the Apache foundation in January 2005, and its first release (0.7) was in August 2005. However, it was not until 0.8 was released in July 2006 that Nutch began the transition to Hadoop-based architecture.

Nutch is still very much alive and is an actively contributed-to project. However, Nutch has now been split into two codebases. Version 1 is the legacy and provides the origins of Hadoop. Version 2 represents something of a re-architecture of the original implementation while still holding true to the original goals of the project.

What Is Hadoop?

Apache Hadoop is a top-level open source project and is governed by the Apache Software Foundation (ASF). Hadoop is not any one entity or thing. It is best thought of as a platform or an ecosystem that describes a method of distributed data processing at scale using commodity hardware configured to run as a cluster of computing power. This architecture enables Hadoop to address and analyze vast quantities of data at significantly lower cost than traditional methods commonly found in data warehousing, for example, with relational database systems.

At its core, Hadoop has two primary functions:

- Processing data (MapReduce)
- Storing data (HDFS)

With the advent of Hadoop 2.0, the next major release of Hadoop, we will see the decoupling of resource management from data processing. This adds a third primary function to this list. However, at the time of this writing, Yarn, the Apache project responsible for the resource management, is in alpha technology preview modes.

That said, a number of additional subprojects have been developed and added to the ecosystem that have been built on top of these two primary functions. When bundled together, these subprojects plus the core projects of MapReduce and HDFS become known as a *distribution*.

Derivative Works and Distributions

To fully understand a distribution, you must first understand the role, naming, and branding of Apache Hadoop. The basic rule here is that only official releases by the Apache Hadoop project may be called *Apache Hadoop* or *Hadoop*. So, what about companies that build products/solutions on top of Hadoop? This is where the term *derivative works* comes in.

What Are Derivative Works?

Any product that uses Apache Hadoop code, known as *artifacts*, as part of its construction is said to be a *derivative work*. A derivative work is *not* an Apache Hadoop release. It may be true that a derivative work can be described as "powered by Apache Hadoop." However, there is strict guidance on product naming to avoid confusion in the marketplace. Consequently, companies that provide distributions of Hadoop should also be considered to be derivative works.

NOTE **I liken the relationship between Hadoop and derivative works to the world of Xbox games development. Many Xbox games use graphics engines provided by a third party. The Unreal Engine is just such an example.**

What Is a Distribution?

Now that you know what a derivative work is, we can look at distributions. A *distribution* is the packaging of Apache Hadoop projects and subprojects plus any other additional proprietary components into a single managed package. For example, Hortonworks provides a distribution of Hadoop called "Hortonworks Data Platform," or HDP for short. This is the distribution used by Microsoft for its product, HDInsight.

You may be asking yourself what is so special about that? You could certainly do this yourself. However, this would be a significant undertaking. First, you'd need to download the projects you want, resolve any dependencies, and then compile all the source code. However, when you decide to go down this route, all the testing and integration of the various components is on you to manage and maintain. Bear in mind that the creators of distributions also employ the committers of the actual source and therefore can also offer support.

As you might expect, distributions may lag slightly behind the Apache projects in terms of releases. This is one of the deciding factors you might want to

consider when picking a distribution. Frequency of updates is a key factor, given how quickly the Hadoop ecosystem evolves.

If you look at the Hortonworks distribution, known as Hortonworks Data Platform (HDP), you can see that there are a number of projects at different stages of development. The distribution brings these projects together and tests them for interoperability and stability. Once satisfied that the projects all hang together, the distributor (in this case, Hortonworks) creates the versioned release of the integrated software (the distribution as an installable package).

The 1.3 version made a number of choices as to which versions to support. Today, though, just a few months later, the top-line Hadoop project has a 1.2.0.5 release available, which is not part of HDP 1.3. This and other ecosystem changes will be consumed in the next release of the HDP distribution.

To see a nice graphic of the Hortonworks distribution history, I will refer you to `http://hortonworks.com/products/hdp-2/`. Hadoop is a rapidly changing and evolving ecosystem and doesn't rest on its laurels so including version history is largely futile.

Hadoop Distributions

Note that there are several Hadoop distributions on the market for you to choose from. Some include proprietary components; others do not. The following sections briefly cover some of the main Hadoop distributions.

Hortonworks HDP

Hortonworks provides a distribution of Apache Hadoop known as Hortonworks Data Platform (HDP). HDP is a 100% open source distribution. Therefore, it does not contain any proprietary code or licensing. The developers employed by Hortonworks contribute directly to the Apache projects. Hortonworks is also building a good track record for regular releases of their distribution, educational content, and community engagement. In addition, Hortonworks has established a number of strategic partnerships, which will stand them in good stead. HDP is available in three forms. The first is for Hadoop 1.x, and the second is for Hadoop 2.0, which is currently in development. Hortonworks also offers HDP for Windows, which is a third distribution. HDP for Windows is the only version that runs on the Windows platform.

MapR

MapR is an interesting distribution for Hadoop. They have taken some radical steps to alter the core architecture of Hadoop to mitigate some of its single points of failure, such as the removal of the single master name node for an alternative

architecture that provides them with a multimaster system. As a result, MapR has also implemented its own JobTracker to improve availability.

MapR also takes a different approach to storage. Instead of using direct attached storage in the data nodes, MapR uses mounted network file storage, which they call Direct Access NFS. The storage provided uses MapR's file system, which is fully POSIX compliant.

MapR is available both within Amazon's Elastic MapReduce Service and within Google's Cloud Platform. MapR also offers a free distribution called M3. However, it is not available in Azure or on Windows and is missing some of the high-availability (HA) features. For those goodies, you have to pay to get either the M5 or M7 versions.

Cloudera CDH

Cloudera, whose chief architect is Doug Cutting, offers an open source distribution called Cloudera Distribution Including Apache Hadoop (CDH). Like MapR, Cloudera has invested heavily in some proprietary extensions to Hadoop for their Enterprise distribution. Cloudera, however, also has an additional release, Cloudera Standard, which combines CDH with their own cluster management tool: Cloudera Manager. Cloudera Manager is proprietary, but it is a free download. As far as competition goes, this puts Cloudera Standard firmly up against Hortonworks's HDP distribution, which includes Ambari for its cluster management.

Cloudera's big-ticket item is Impala. Impala is a real-time, massively parallel processing (MPP) query engine that runs natively on Hadoop. This enables users to issue SQL queries against data stored in HDFS and Apache HBase without having to first move the data into another platform.

IS HDINSIGHT A DISTRIBUTION?

In a word, no. HDInsight is a product that has been built on top of the Hortonworks HDP distribution (specifically the HDP distribution for Windows). At the time of this writing, HDP 1.3 is the currently available version.

Core Hadoop Ecosystem

Some projects in the world of Hadoop are simply more important than others. Projects like HDFS, the Hadoop Distributed File System, are fundamental to the operation of Hadoop. Similarly, MapReduce currently provides both the scheduling and the execution and programming engines to the whole of Hadoop. Without these two projects there simply is no Hadoop.

In this next section, we are going to delve a little deeper into these core Hadoop projects to build up our knowledge of the main building blocks. Once we've done that, we'll be well placed to move forward with the next section, which will touch on some of the other projects in the Hadoop ecosystem.

HDFS

HDFS, one of the core components of Apache Hadoop, stands for Hadoop Distributed File System. There's no exotic branding to be found here. HDFS is a Java-based, distributed, fault-tolerant file storage system designed for distribution across a number of commodity servers. These servers have been configured to operate together as an HDFS *cluster*. By leveraging a scale-out model, HDFS ensures that it can support truly massive data volumes at a low and linear cost point.

Before diving into the details of HDFS, it is worth taking a moment to discuss the files themselves. Files created in HDFS are made up of a number of HDFS *data blocks* or simply HDFS *blocks*. These blocks are not small. They are 64MB or more in size, which allows for larger I/O sizes and in turn greater throughput. Each block is replicated and then distributed across the machines of the HDFS cluster.

HDFS is built on three core subcomponents:

- NameNode
- DataNode
- Secondary NameNode

Simply put, the NameNode is the "brain." It is responsible for managing the file system, and therefore is responsible for allocating directories and files. The NameNode also manages the *blocks*, which are present on the DataNode. There is only one NameNode per HDFS cluster.

The DataNodes are the workers, sometimes known as *slaves*. The DataNodes perform the bidding of the NameNode. DataNodes exist on every machine in the cluster, and they are responsible for offering up the machine's storage to HDFS. In summary, the job of the DataNode is to manage all the I/O (that is, read and write requests).

HDFS is also the point of integration for a new Microsoft technology called Polybase, which you will learn more about in Chapter 10, "Data Warehouses and Hadoop Integration."

MapReduce

MapReduce is both an engine and a programming model. Users develop MapReduce programs and submit them to the MapReduce engine for processing. The programs created by the developers are known as *jobs*. Each *job* is a

combination of Java ARchive (JAR) files and classes required to execute the MapReduce program. These files are themselves collated into a single JAR file known as a *job file*.

Each MapReduce job can be broken down into a few key components. The first phase of the job is the *map*. The *map* breaks the input up into many tiny pieces so that it can then process each piece independently and in parallel. Once complete, the results from this initial process can be collected, aggregated, and processed. This is the *reduce* part of the job.

The MapReduce engine is used to distribute the workload across the HDFS cluster and is responsible for the execution of MapReduce jobs. The MapReduce engine accepts jobs via the JobTracker. There is one JobTracker per Hadoop cluster (the impact of which we discuss shortly). The JobTracker provides the scheduling and orchestration of the MapReduce engine; it does not actually process data itself.

To execute a job, the JobTracker communicates with the HDFS NameNode to determine the location of the data to be analyzed. Once the location is known, the JobTracker then speaks to another component of the MapReduce engine called the *TaskTracker.* There are actually many TaskTracker nodes in the Hadoop cluster. Each node of the cluster has its own TaskTracker. Clearly then, the MapReduce engine is another master/slave architecture.

TaskTrackers provide the execution engine for the MapReduce engine by spawning a separate process for every task request. Therefore, the JobTracker must identify the appropriate TaskTrackers to use by assessing which are available to accept task requests and, ideally, which trackers are closest to the data. After the decision has been made, the JobTracker can submit the workload to the targeted TaskTrackers.

TaskTrackers are monitored by the JobTracker. This is a bottom-up monitoring process. Each TaskTracker must "report in" via a heartbeat signal. If it fails to do so for any reason, the JobTracker assumes it has failed and reassigns the tasks accordingly. Similarly, if an error occurs during the processing of an assigned task, the TaskTracker is responsible for calling that in to the JobTracker. The decision on what to do next then lies with the JobTracker.

The JobTracker keeps a record of the tasks as they complete. It maintains the status of the job, and a client application can poll it to get the latest state of the job.

NOTE **The JobTracker is a single point of failure for the MapReduce engine. If it goes down, all running jobs are halted, and new jobs cannot be scheduled.**

Important Apache Projects for Hadoop

Now that we have a conceptual grasp of the core projects for Hadoop (the brain and heart if you will), we can start to flesh out our understanding of the broader ecosystem. There are a number of projects that fall under the Hadoop umbrella.

Some will succeed, while others will wither and die. That is the very nature of open source software. The good ideas get developed, evolve, and become great—at least, that's the theory.

Some of the projects we are about to discuss are driving lots of innovation—especially for Hadoop 2.0. Hive is the most notable project in this regard. Almost all the work around the Hortonworks Stinger initiative is to empower SQL in Hadoop. Many of these changes will be driven through the Hive project. Therefore, it is important to know what Hive is and why it is getting so much attention.

Hive

Apache Hive is another key subproject of Hadoop. It provides data warehouse software that enables a SQL-like querying experience for the end user. The Hive query language is called Hive Query Language (HQL). (Clearly, the creators of Hive had no time for any kind of creative branding.) HQL is similar to ANSI SQL, making the crossover from one to the other relatively simple. HQL provides an abstraction over MapReduce; HQL queries are translated by Hive into MapReduce jobs. Hive is therefore quite a popular starting point for end users because there is no need to learn how to program a MapReduce job to access and process data held in Hadoop.

It is important to understand that Hive does not turn Hadoop into a relational database management system (RDBMS). Hive is still a batch-processing system that generates MapReduce jobs. It does not offer transactional support, a full type system, security, high concurrency, or predictable response times. Queries tend to be measured in minutes rather in than milliseconds or seconds. This is because there is a high spin-up cost for each query and, at the end of the day, no cost-based optimizer underpins the query plan like traditional SQL developers are used to. Therefore, it is important not to overstate Hive's capabilities.

Hive does offer certain features that an RDBMS might not, though. For example, Hive supports the following complex types: structs, maps (key/value pairs), and arrays. Likewise, Hive offers native operator support for regular expressions, which is an interesting addition. HQL also offers additional extensibility by allowing MapReduce developers to plug in their own custom mappers and reducers, allowing for more advanced analysis.

The most recent and exciting developments for Hive have been the new *Stinger* initiatives. Stinger has the goal of delivering 100X performance improvement to Hive plus SQL compatibility. These two features will have a profound impact on Hadoop adoption; keep them on your radar. We'll talk more about Stinger in Chapter 2, "Microsoft's Approach to Big Data."

Pig

Apache Pig is an openly extensible programmable platform for loading, manipulating, and transforming data in Hadoop using a scripting language called Pig

Latin. Pig is another abstraction on top of the Hadoop core. It converts the Pig Latin script into MapReduce jobs, which can then be executed against Hadoop.

Pig Latin scripts define the flow of data through transformations and, although simple to write, can result in complex and sophisticated manipulation of data. So, even though Pig Latin is SQL-like syntactically, it is more like a SQL Server Integration Services (SSIS) Data Flow task in spirit. Pig Latin scripts can have multiple inputs, transformations, and outputs. Pig has a large number of its own built-in functions, but you can always either create your own or just "raid the piggybank" (`https://cwiki.apache.org/confluence/display/PIG/PiggyBank`) for community-provided functions.

As previously mentioned, Pig provides its scalability by operating in a distributed mode on a Hadoop cluster. However, Pig Latin programs can also be run in a local mode. This does not use a Hadoop cluster; instead, the processing takes place in a single local Java Virtual Machine (JVM). This is certainly advantageous for iterative development and initial prototyping.

SQOOP

SQOOP is a top-level Apache project. However, I like to think of Apache SQOOP as a glue project. It provides the vehicle to transfer data from the relational, tabular world of structured data stores to Apache Hadoop (and vice versa).

SQOOP is extensible to allow developers to create new connectors using the SQOOP application programming interface (API). This is a core part of SQOOP's architecture, enabling a plug-and-play framework for new connectors.

SQOOP is currently going through something of a re-imagining process. As a result, there are now two versions of SQOOP. SQOOP 1 is a client application architecture that interacts directly with the Hadoop configurations and databases. SQOOP 1 also experienced a number of challenges in its development. SQOOP 2 aims to address the original design issues and starts from a server-based architecture. These are discussed in more detail later in this book.

Historically, SQL Server had SQOOP connectors that were separate downloads available from Microsoft. These have now been rolled into SQOOP 1.4 and are also included into the HDInsight Service. SQL Server Parallel Data Warehouse (PDW) has an alternative technology, Polybase, which we discuss in more detail in Chapter 10, "Data Warehouses and Hadoop Integration."

HCatalog

So, what is HCatalog? Simply put, HCatalog provides a tabular abstraction of the HDFS files stored in Hadoop. A number of tools then leverage this abstraction when working with the data. Pig, Hive, and MapReduce all use this abstraction to reduce the complexity and overhead of reading and writing data to Hadoop.

HDFS files can, in theory, be in any format, and the data blocks can be placed anywhere on the cluster. HCatalog provides the mechanism for mapping both the file formats and data locations to the tabular view of the data. Again, HCatalog is open and extensible to allow for the fact that some file formats may be proprietary. Additional coding would be required, but the fact that a file format in HDFS was previously unknown would not be a blocker to using HCatalog.

Apache HCatalog is technically no longer a Hadoop project. It is still an important feature, but its codebase was merged with the Hive Project early in 2013. HCatalog is built on top of the Hive and leverages its command-line interface for issuing commands against the HCatalog.

One way to think about HCatalog is as the master database for Hive. In that sense, HCatalog provides the catalog views and interfaces for your Hadoop "database."

HBase

HBase is an interesting project because it provides NoSQL database functionality on top of HDFS. It is also a column store, providing fast access to large quantities of data, which is often sparsely populated. HBase also offers transactional support to Hadoop, enabling a level of Data Modification Language (DML) (that is, inserts, updates, and deletes). However, HBase does not offer a SQL interface; remember, it is part of the NoSQL family. It also does not offer a number of other RDBMS features, such as typed columns, security, enhanced data programmability features, and querying languages.

HBase is designed to work with large tables, but you are unlikely to ever see a table like this in an RDBMS (not even in a SharePoint database). HBase tables can have billions of rows, which is not uncommon these days; but in conjunction with that, those rows can have an almost limitless number of columns. In that sense, there could be millions of columns. In contrast, SQL Server is limited to 1,024 columns.

Architecturally, HBase belongs to the master/slave collection of distributed Hadoop implementations. It is also heavily reliant on Zookeeper (an Apache project we discuss shortly).

Flume

Flume is the StreamInsight of the Hadoop ecosystem. As you would expect, it is a distributed system that collects, aggregates, and shifts large volumes of event streaming data into HDFS. Flume is also fault tolerant and can be tuned for failover and recovery. However, in general terms, faster recovery tends to mean trading some performance; so, as with most things, a balance needs to be found.

The Flume architecture consists of the following components:

- Client
- Source
- Channel
- Sink
- Destination

Events flow from the client to the source. The *source* is the first Flume component. The source inspects the event and then farms it out to one or more *channels* for processing. Each channel is consumed by a *sink*. In Hadoop parlance, the event is "drained" by the sink. The channel provides the separation between source and sink and is also responsible for managing recovery by persisting events to the file system if required.

Once an event is drained, it is the sink's responsibility to then deliver the event to the destination. There are a number of different sinks available, including an HDFS sink. For the Integration Services users out there familiar with the term backpressure, you can think of the channel as the component that handles backpressure. If the source is receiving events faster than they can be drained, it is the channel's responsibility to grow and manage that accumulation of events.

A single pass through a source, channel, and sink is known as a *hop*. The components for a hop exist in a single JVM called an *agent*. However, Flume does not restrict the developer to a single hop. Complex multihop flows are perfectly possible with Flume. This includes creating fan-out and fan-in flows; failover routes for failed hops; and conditional, contextual routing of events. Consequently, events can be passed from agent to agent before reaching their ultimate destination.

Mahout

Mahout is all about machine learning. The goal of the project is to build scalable machine-learning libraries. The core of Apache Mahout is implemented on top of Hadoop using MapReduce. However, the project does not limit itself to that paradigm. At present, Mahout is focused on four use cases:

- **Recommendation mining**: Recommendation mining is the driving force behind several recommendation engines. How many of you have seen something like this appear in your inbox: "Because you bought this New England Patriots shirt, you might also like this NFL football."
- **Clustering**: Clustering is the grouping of text documents to create topically related groupings or categories.

- **Classification**: Classification algorithms sit on top of classified documents and subsequently learn how to classify new documents. You could imagine how recruitment agents would love clustering and classification for their buzzword bingo analysis. If Apache Mahout is able to reduce the number of calls received for the wrong job, that's a win for everyone in my book.
- **Frequent item set mining**: Frequent item set mining is a way to understand which items are often bucketed together (for example, in shopping basket analysis).

Ambari

Ambari is the system center of the Hadoop ecosystem. It provides all the provisioning, operational insight, and management for Hadoop clusters. Remember that Hadoop clusters can contain many hundreds or thousands of machines. Keeping them configured correctly is a significant undertaking, and so having some tooling in this space is absolutely essential.

Ambari provides a web interface for ease of management where you can check on all the Hadoop services and core components. The same web interface can also be used to monitor the cluster, configuring notification alerts for health and performance conditions. Job diagnostic information is also surfaced in the web UI, helping users better understand job interdependencies, historic performance, and system trends.

Finally, Ambari can integrate with other third-party monitoring applications via its RESTful API. So when I say it is the system center of Hadoop, it literally is!

Oozie

Oozie is a Java web scheduling application for Hadoop. Often, a single job on its own does not define a business process. More often than not, there is a chain of events, processing, or processes that must be initiated and completed for the result to have meaning. It is Oozie's lot in life to provide this functionality. Simply put, Oozie can be used to compose a single container/unit of work from a collection of jobs, scripts, and programs. For those familiar with enterprise schedulers, this will be familiar territory. Oozie takes these units of work and can schedule them accordingly.

It is important to understand that Oozie is a trigger mechanism. It submits jobs and such, but MapReduce is the executor. Consequently, Oozie must also solicit status information for actions that it has requested. Therefore, Oozie has callback and polling mechanisms built in to provide it with job status/completion information.

Zookeeper

Distributed applications use Zookeeper to help manage and store configuration information. Zookeeper is interesting because it steps away from the master/slave model seen in other areas of Hadoop and is itself a highly distributed architecture and consequently highly available. What is interesting is that it achieves this while providing a "single view of the truth" for the configuration information data that it holds. Zookeeper is responsible for managing and mediating potentially conflicting updates to this information to ensure synchronized consistency across the cluster. For those of you who are familiar with managing complex merge replication topologies, you know that this is no trivial task!

The Future for Hadoop

You don't have to look too far into the future to discern the future direction of Hadoop. Alpha code and community previews are already available for Hadoop 2.0, which is fantastic to see. Aside from this, the projects we've talked about in the previous section continue to add new features, and so we should also expect to see new V1 distributions from the likes of Hortonworks for the foreseeable future.

Of course, one of the most exciting things to happen to Hadoop is the support for Hadoop on Windows and Azure. The opportunity this presents for the market cannot be overstated. Hadoop is now an option for all data professionals on all major platforms, and that is very exciting indeed.

So, what can we expect in Hadoop 2.0? Two projects that are worth highlighting here (at least in summary): YARN and Tez.

Summary

In this first chapter, you learned all about what big data is, about the core components of the Hadoop ecosystem, and a little bit about its history and inspiration. The stage is set now for you to immerse yourself in this new and exciting world of big data using Hadoop.