# UNIT-I

# INDEX

**Defining Big Data Analytics**

- Using Big Data to Get Results
- Basic analytics
- Advanced analytics
- Operationalized analytics
- Monetizing analytics

**Modifying Business Intelligence Products to Handle Big Data**

- Data
- Analytical algorithms
- Infrastructure support

**Big Data Analytics Examples, Big Data Analytics Solutions.**

## Meet Hadoop

- Data Storage and Analysis
- Comparison with Other Systems
- A Brief History of Hadoop
- Apache Hadoop
- Hadoop Ecosystem

**Defining Big Data Analytics**

- Using big data to get results
- Finding what's different with big data
- Exploring the challenges of analyzing big data
- Examining analytics tools for big data

Spending a lot of time describing the infrastructure you need to support your big data initiatives. However, because big data is most useful if you can do something with it, the question becomes, how do you analyze it? Companies like Amazon and Google are masters at analyzing big data. And they use the resulting knowledge to gain a competitive advantage. Just think about Amazon's recommendation engine. The company takes all your buying history together with what it knows about you, your buying patterns, and the buying patterns of people like you to come up with some pretty good suggestions. It is a marketing machine, and its big data analytics capabilities have made it extremely successful. The capability to analyze big data provides unique opportunities for your organization as well. You'll be able to expand the kind of analysis you can do. Instead of being limited to sampling large data sets, you can now utilize much more detailed and complete data to do your analysis. However, analyzing big data can also be challenging. Changing algorithms and technology, even for basic data analysis, often has to be addressed with big data. We introduce big data analytics. We focus on the kinds of analysis you can do with big data. We also discuss some of the differences you need to think about between big data analytics and traditional analytics. In this chapter, we focus primarily on structured data analysis, although unstructured data is a very important part of the big data picture.

**Using Big Data to Get Results**

The first question that you need to ask yourself before you dive into big data analysis is

**What problem are you trying to solve?**

You may not even be sure of what you are looking for. You know you have lots of data that you think you can get valuable insight from. And certainly, patterns can emerge from that data before you understand why they are there.

**If you think about it though, you're sure to have an idea of what you're interested in.** For instance, are you interested in predicting customer behavior to prevent churn? Do you want to analyze the driving patterns of your customers for insurance premium purposes? Are you interested in looking at your system log data to ultimately predict when problems might occur? The kind of high-level problem is going to drive the analytics you decide to use. Alternately, if you're not exactly sure of the business problem you're trying to solve, maybe you need to look at areas in your business that needs improvement. Even an analytics-driven strategy — targeted at the right area — can provide useful results with big data.

When it comes to analytics, you might consider a range of possible kinds, which are outlined in Table 12-1.

| Table 12-1 | Big Data Analysis |
|---|---|
| ***Analysis Type*** | ***Description*** |
| Basic analytics for insight | Slicing and dicing of data, reporting, simple visualizations, basic monitoring. |
| Advanced analytics for insight | More complex analysis such as predictive modeling and other pattern-matching techniques. |
| Operationalized analytics | Analytics become part of the business process. |
| Monetized analytics | Analytics are utilized to directly drive revenue. |

# Basic analytics

Basic analytics can be used to explore your data, if you're not sure what you have, but you think something is of value. This might include simple visualizations or simple statistics. Basic analysis is often used when you have large amounts of disparate data. Here are some examples:

✓ **Slicing and dicing:** *Slicing and dicing* refers to breaking down your data into smaller sets of data that are easier to explore. For example, you might have a scientific data set of water column data from many different locations that contains numerous variables captured from multiple sensors. Attributes might include temperature, pressure, transparency,
dissolved oxygen, pH, salinity, and so on, collected over time. You might want some simple graphs or plots that let you explore your data across different dimensions, such as temperature versus pH or transparency versus salinity. You might want some basic statistics such as average or range for each attribute, from each height, for the time period. The point
is that you might use this basic type of exploration of the variables to ask specific questions in your problem space. The difference between this kind of analysis and what happens in a basic business intelligence system is that you're dealing with huge volumes of data where you might not know how much query space you'll need to examine it and you're probably going to want to run computations in real time.

✓ **Basic monitoring:** You might also want to monitor large volumes of data in real time. For example, you might want to monitor the water column attributes in the preceding example every second for an extended period of time from hundreds of locations and at varying heights in the water column. This would produce a huge data set. Or, you might be interested
in monitoring the buzz associated with your product every minute when you launch an ad campaign. Whereas the water column data set might produce a large amount of relatively structured time-sensitive data, the social media campaign is going to produce large amounts of disparate kinds of data from multiple sources across the Internet.

✓ **Anomaly identification:** You might want to identify anomalies, such as an event where the actual observation differs from what you expected, in your data because that may clue you in that something is going wrong with your organization, manufacturing process, and so on. For example, you might want to analyze the records for your manufacturing operation to determine whether one kind of

machine, or one operator, has a higher incidence of a certain kind of problem. This might involve some simple statistics like moving averages triggered by an alert from the problematic machine.

# Advanced analytics

Advanced analytics provides algorithms for complex analysis of either structured or unstructured data. It includes sophisticated statistical models, machine learning, neural networks, text analytics, and other advanced data-mining techniques. Among its many use cases, advanced analytics can be deployed to find patterns in data, prediction, forecasting, and complex event processing. While advanced analytics has been used by statisticians and mathematicians for decades, it was not as big a part of the analytics landscape as it is today. Consider that 20 years ago, statisticians at companies were able to predict who might drop a service using advanced survival analysis or machine learning techniques.

However, it was difficult to persuade other people in the organization to understand exactly what this meant and how it could be used to provide a competitive advantage. For one thing, it was difficult to obtain the computational power needed to interpret data that kept changing through time. Today, advanced analytics is becoming more mainstream. With increases in computational power, improved data infrastructure, new algorithm development, and the need to obtain better insight from increasingly vast amounts of data, companies are pushing toward utilizing advanced analytics as part of their decision-making process. Businesses realize that better insights can provide a superior competitive position.

Here are a few examples of advanced analytics for big data:

✓ **Predictive modeling:** Predictive modeling is one of the most popular big data advanced analytics use cases. A predictive model is a statistical or data-mining solution consisting of algorithms and techniques that can be used on both structured and unstructured data (together or individually) to determine future outcomes. For example, Telecommunications Company might use a predictive model to predict customers who might drop its service. In the big data world, you might have large numbers of predictive attributes across huge amounts of observations. Whereas in the past, it might have taken hours (or longer) to run a predictive model, with a large amount of data on your desktop, you might be able to now run it iteratively hundreds of times if you have a big data infrastructure in place.

✓ **Text analytics:** Unstructured data is such a big part of big data, so text analytics — the process of analyzing unstructured text, extracting relevant information, and transforming it into structured information that can then be leveraged in various ways — has become an important component of the big data ecosystem. The analysis and extraction processes used in text analytics take advantage of techniques that originated in computational linguistics, statistics, and other computer science disciplines. Text analytics is being used in all sorts of analysis, from predicting churn, to fraud, and to social media analytics. It is so important that we devote a considerable part of Chapter 13 to this issue of text analytics.

✓ **Other statistical and data-mining algorithms:** This may include advanced forecasting, optimization, cluster analysis for segmentation or even micro segmentation, or affinity analysis.

Advanced analytics doesn't require big data. However, being able to apply advanced analytics with big data can provide some important results.

**Operationalized analytics**

When you operationalize analytics, you make them part of a business process. For example, statisticians at an insurance company might build a model that predicts the likelihood of a claim being fraudulent. The model, along with some decision rules, could be included in the company's claims-processing system to flag claims with a high probability of fraud. These claims would be sent to an investigation unit for further review. In other cases, the model itself might not be as apparent to the end user. For example, a model could be built to predict customers who are good targets for upselling when they call into a call center. The call center agent, while on the phone with the customer, would receive a message on specific additional products to sell to this customer. The agent might not even know that a predictive model was working behind the scenes to make this recommendation.

**Monetizing analytics**

Analytics can be used to optimize your business to create better decisions and drive bottom- and top-line revenue. However, big data analytics can also be used to derive revenue above and beyond the insights it provides just for your own department or company. You might be able to assemble a unique data set that is valuable to other companies, as well. For example, credit card providers take the data they assemble to offer value-added analytics products. Likewise, with financial institutions. Telecommunications companies are beginning to sell location-based insights to retailers. The idea is that various sources of data, such as billing data, location data, text-messaging data, or web-browsing data can be used together or separately to make inferences about customer behavior patterns that retailers would find useful. As a regulated industry, they must do so in compliance with legislation and privacy policies.

**Modifying Business Intelligence Products to Handle Big Data**

Traditional business intelligence products weren't really designed to handle big data. They were designed to work with highly structured, well-understood data, often stored in a relational data repository and displayed on your desktop or laptop computer. This traditional business intelligence analysis is typically applied to snapshots of data rather than the entire amount of data available. So what's different when you start to analyze big data?

**Data**

As we discuss in Chapter 2, big data consists of structured, semi-structured, and unstructured data. You often have a lot of it, and it can be quite complex. When you think about analyzing it, you need to be aware of the potential characteristics of your data:

✓ **It can come from untrusted sources.** Big data analysis often involves aggregating data from various sources. These may include both internal and external data sources. How trustworthy are these external sources of information? For example, how trustworthy is social media data like a tweet? The information

may be coming from an unverified source. The integrity of this data needs to be considered in the analysis. We talk more about big data security and governance in Chapter 19.

✓ **It can be dirty.** Dirty data refers to inaccurate, incomplete, or erroneous data. This may include the misspelling of words; a sensor that is broken, not properly calibrated, or corrupted in some way; or even duplicated data. Data scientists debate about where to clean the data — either close to the source or in real time. Of course, one school of thought says that the dirty data should not be cleaned at all because it may contain interesting outliers. The cleansing strategy will probably depend on the source and type of data and the goal of your analysis. For example, if you're developing a spam filter, the goal is to detect the bad elements in the data, so you would not want to clean it.

✓ **The signal-to-noise ratio can be low.** In other words, the signal (usable information) may only be a tiny percent of the data; the noise is the rest. Being able to extract a tiny signal from noisy data is part of the benefit of big data analytics, but you need to be aware that the signal may indeed be small.

✓ **It can be real-time.** In many cases, you'll be trying to analyze real-time data streams.

Big data governance is going to be an important part of the analytics equation. Underneath business analytics, enhancements will need to be made to governance solutions to ensure the veracity coming from the new data sources, especially as it is being combined with existing trusted data stored in a warehouse. Data security and privacy solutions also need to be enhanced to support managing/governing big data stored within new technologies.

**Analytical algorithms**

When you're considering big data analytics, you need to be aware that when you expand beyond the desktop, the algorithms you use often need to be *refactored,* changing the internal code without affecting its external functioning. The beauty of a big data infrastructure is that you can run a model that used to take hours or days in minutes. This lets you iterate on the model hundreds of times over. However, if you're running a regression on a billion rows of data across a distributed environment, you need to consider the resource requirements relating to the volume of data and its location in the cluster. Your algorithms need to be data aware. Additionally, vendors are starting to offer a new breed of analytics designed to be placed close to the big data sources to analyze data in place rather than first having to store it and then analyze it. This approach of running analytics closer to the data sources minimizes the amount of stored data by retaining only the high-value data. It is also enables you to analyze the data sooner, looking for key events, which is critical for real-time decision making. Of course, analytics will continue to evolve. For example, you may need realtime visualization capabilities to display real-time data that is continuously changing. How do you practically plot a billion points on a graph plot? Or, how do you work with the predictive algorithms so that they perform fast enough and deep enough analysis to utilize an ever-expanding, complex data set? This is an area of active research.

**Infrastructure support**

We spend a good deal of this book talking about the infrastructure needed to support big data, so we don't go into detail about that here. You might want to turn to Chapter 4 for more details on infrastructure issues. Suffice it to say that if you're looking for a platform, it needs to achieve the following:

✓ **Integrate technologies:** The infrastructure needs to integrate new big data technologies with traditional technologies to be able to process all kinds of big data and make it consumable by traditional analytics.

✓ **Store large amounts of disparate data:** An enterprise-hardened Hadoop system may be needed that can process/store/manage large amounts of data at rest, whether it is structured, semi-structured, or unstructured.

✓ **Process data in motion:** A stream-computing capability may be needed to process data in motion that is continuously generated by sensors, smart devices, video, audio, and logs to support real-time decision making.

✓ **Warehouse data:** You may need a solution optimized for operational or deep analytical workloads to store and manage the growing amounts of trusted data.

And of course, you need the capability to integrate the data you already have in place along with the results of the big data analysis.

**Big Data Analytics Examples**

Big data analytics has many different use cases. We mention examples throughout this book, but we now look at a few others from Internet companies and others.

**Orbitz**

If you've ever looked for deals on travel, you've probably been to sites like Orbitz (www.orbitz.com). The company was established in 1999, and its website went live in 2001. Users of Orbitz perform over a million searches a day, and the company collects hundreds of gigabytes of raw data each day from these searches. Orbitz realized that it might have useful information in the web log files that it was collecting from its web analytics software that contained information about consumer interaction with its site. In particular, it was interested to see whether it could identify consumer preferences to determine the best-performing hotels to display to users so that it could increase conversions (bookings). It had not been utilizing this data in the past because it was too expensive to store all of it. It implemented Hadoop and Hive running on comodity hardware to help. Hadoop provided the distributed file system and Hive provided an SQL-type interface. It took a series of steps to put the data into Hive. After the data was in Hive, the company used *machine learning* — a data-driven (and data-mining; see the sidebar earlier in this chapter) approach to unearthing patterns in data and helping to analyze the data.

**Nokia**

Nokia provides wireless communication devices and services. The company believes that its data is a strategic asset. Its big data analytics service includes a multi-peta byte platform that executes over tens of thousands of jobs each day. This includes utilizing advanced analytics over terabytes of streaming data. For example, the company wants to understand how people interact with its different applications on its phones. Nokia wants to understand what features customers use, how they use a feature, and how they move from feature to feature and whether they get lost in the application as they are using it. This level of detail helps the company lay out new features for its applications and improve customer retention.

**NASA**

NASA is using predictive models to analyze safety data on aircrafts. It wants to understand whether the introduction of a new technology into an aircraft will make a dramatic impact in safety. Needless to say, NASA is dealing with a massive amount of data. Each airplane each day is recording a *thousand* parameters every second for every flight. Some of this data is streaming. The company also receives text data from reports written by pilots and other crew members. NASA also throws weather data (that changes in time and space) into the mix. The data scientists there are looking to predict

outcomes — for example, what pattern indicates a possible accident or incident.

**Big Data Analytics Solutions**

A number of vendors on the market today support big data solutions. Here is a listing of a few solutions that you may find interesting:

✓ **IBM** (www.ibm.com) is taking an enterprise approach to big data and integrating across the platform including embedding/bundling its analytics. Its products include a warehouse (InfoSphere warehouse) that has its own built-in data-mining and cubing capability. Its new PureData Systems (a packaging of advanced analytics technology into an integrated systems platform) includes many packaged analytical integrations. Its InfoSphere Streams product is tightly integrated with its Statistical Package for the Social Sciences (SPSS) statistical software to support real-time predictive analytics, including the capability to dynamically update models based on real-time data. It is bundling a limited-use license of Cognos Business Intelligence with its key big dataplatform capabilities (enterprise-class Hadoop, stream computing, and warehouse solutions).

✓ **SAS** (www.sas.com) provides multiple approaches to analyze big data via its high-performance analytics infrastructure and its statistical software. SAS provides several distributed processing options. These include in-database analytics, in-memory analytics, and grid computing. Deployments can be on-site or in the cloud.

✓ **Tableau** (www.tableausoftware.com), a business analytics and data visualization software company, offers its visualization capabilities to run on top appliances and other infrastructure offered by a range of big data partners, including Cirro, EMC Greenplum, Karmasphere, Teradata/ Aster, HP Vertica, Hortonworks, ParAccel, IBM Netezza, and a host of

others.

✓ **Oracle** (www.oracle.com) offers a range of tools to complement its big data platform called Oracle Exadata. These include advanced analytics via the R programming language, as well as an in-memory database option with Oracle's Exalytics in-memory machine and Oracle's data warehouse. Exadata is integrated with its hardware platform.

✓ **Pentaho** (www.pentaho.com) provides open source business analytics via a community and enterprise edition. Pentaho supports the leading Hadoop-based distributions and supports native capabilities, such as MapR's NFS high-performance mountable file system.

## MEET HADOOP

### Data Storage and Analysis

The problem is simple: while the storage capacities of hard drives have increased massively over the years, access speeds—the rate at which data can be read from drives— have not kept up. One typical drive from 1990 could store 1,370 MB of data and had a transfer speed of 4.4 MB/s,§ so you could read all the data from a full drive in around five minutes. Over 20 years later, one terabyte drives are the norm, but the transfer speed is around 100 MB/s, so it takes more than two and a half hours to read all the data off the disk. This is a long time to read all data on a single drive—and writing is even slower. The obvious way to reduce the time is to read from multiple disks at once. Imagine if we had 100 drives, each holding one hundredth of the data. Working in parallel, we could read the data in under two minutes. Only using one hundredth of a disk may seem wasteful. But we can store one hundred datasets, each of which is one terabyte, and provide shared access to them. We can imagine that the users of such a system would be happy to share access in return for shorter analysis times, and, statistically, that their analysis jobs would be likely to be spread over time, so they wouldn't interfere with each other too much. There's more to being able to read and write data in parallel to or from multiple disks, though. The first problem to solve is hardware failure: as soon as you start using many pieces of hardware, the chance that one will fail is fairly high. A common way of avoiding data loss is through replication: redundant copies of the data are kept by the system so that in the event of failure, there is another copy available. This is how RAID works, for instance, although Hadoop's filesystem, the Hadoop Distributed Filesystem (HDFS), takes a slightly different approach, as you shall see later. The second problem is that most analysis tasks need to be able to combine the data in some way; data read from one disk may need to be combined with the data from any of the other 99 disks. Various distributed systems allow data to be combined from multiple sources, but doing this correctly is notoriously challenging. MapReduce provides

a programming model that abstracts the problem from disk reads and writes, transforming it into a computation over sets of keys and values. We will look at the details of this model in later chapters, but the important point for the present discussion is that there are two parts to the computation, the map and the reduce, and it's the interface between the two where the "mixing" occurs. Like HDFS, MapReduce has built-in reliability. This, in a nutshell, is what Hadoop provides: a reliable shared storage and analysis system. The storage is provided by HDFS and analysis by MapReduce. There are other parts to Hadoop, but these capabilities are its kernel.

### Comparison with Other Systems

The approach taken by MapReduce may seem like a brute-force approach. The premise is that the entire dataset—or at least a good portion of it—is processed for each query. But this is its power. MapReduce is a *batch* query processor, and the ability to run an ad hoc query against your whole dataset and get the results in a reasonable time is transformative. It changes the way you think about data, and unlocks data that was previously archived on tape or disk. It gives people the opportunity to innovate with data. Questions that took too long to get answered before can now be answered, which in turn leads to new questions and new insights. For example, Mailtrust, Rackspace's mail division, used Hadoop for processing email logs. One ad hoc query they wrote was to find the geographic

distribution of their users. In their words: This data was so useful that we've scheduled the MapReduce job to run monthly and we will be using this data to help us decide which Rackspace data centers to place new mail servers in as we grow.
By bringing several hundred gigabytes of data together and having the tools to analyze it, the Rackspace engineers were able to gain an understanding of the data that they otherwise would never have had, and, furthermore, they were able to use what they had learned to improve the service for their customers. You can read more about how Rackspace uses Hadoop in Chapter 16.

**RDBMS**

Why can't we use databases with lots of disks to do large-scale batch analysis? Why is MapReduce needed?
The answer to these questions comes from another trend in disk drives: seek time is improving more slowly than transfer rate. Seeking is the process of moving the disk's head to a particular place on the disk to read or write data. It characterizes the latency of a disk operation, whereas the transfer rate corresponds to a disk's bandwidth.
If the data access pattern is dominated by seeks, it will take longer to read or write large portions of the dataset than streaming through it, which operates at the transfer rate. On the other hand, for updating a small proportion of records in a database, a traditional B-Tree (the data structure used in relational databases, which is limited by the
rate it can perform seeks) works well. For updating the majority of a database, a B-Tree is less efficient than MapReduce, which uses Sort/Merge to rebuild the database. In many ways, MapReduce can be seen as a complement to an RDBMS. (The differences between the two systems are shown in Table 1-1.) MapReduce is a good fit for problems that need to analyze the whole dataset, in a batch fashion, particularly for ad hoc analysis.
An RDBMS is good for point queries or updates, where the dataset has been indexed to deliver low-latency retrieval and update times of a relatively small amount of data. MapReduce suits applications where the data is written once, and read many times, whereas a relational database is good for datasets that are continually updated.

*Table 1-1. RDBMS compared to MapReduce*

| | **Traditional RDBMS** | **MapReduce** |
|---|---|---|
| **Data size** | Gigabytes | Petabytes |
| **Access** | Interactive and batch | Batch |
| **Updates** | Read and write many times | Write once, read many times |
| **Structure** | Static schema | Dynamic schema |
| **Integrity** | High | Low |
| **Scaling** | Nonlinear | Linear |

Another difference between MapReduce and an RDBMS is the amount of structure in the datasets that they operate on. *Structured data* is data that is organized into entities that have a defined format, such as XML documents or database tables that conform to a particular predefined schema. This is the realm of the RDBMS. *Semi-structured data*, on the other hand, is looser, and though there may be a schema, it is often ignored, so it may be used only as a guide to the structure of the data: for example, a spreadsheet, in which the structure is the grid of cells, although the

cells themselves may hold any form of data. *Unstructured data* does not have any particular internal structure: for example, plain text or image data. MapReduce works well on unstructured or semistructured data, since it is designed to interpret the data at processing time. In other words, the input keys and values for MapReduce are not an intrinsic property of the data, but they are chosen by the person analyzing the data.

Relational data is often *normalized* to retain its integrity and remove redundancy. Normalization poses problems for MapReduce, since it makes reading a record a nonlocal operation, and one of the central assumptions that MapReduce makes is that it is possible to perform (high-speed) streaming reads and writes. A web server log is a good example of a set of records that is *not* normalized (for example, the client hostnames are specified in full each time, even though the same client may appear many times), and this is one reason that logfiles of all kinds are particularly well-suited to analysis with MapReduce.

MapReduce is a linearly scalable programming model. The programmer writes two functions—a map function and a reduce function—each of which defines a mapping from one set of key-value pairs to another. These functions are oblivious to the size of the data or the cluster that they are operating on, so they can be used unchanged for a small dataset and for a massive one. More important, if you double the size of the input data, a job will run twice as slow. But if you also double the size of the cluster, a job will run as fast as the original one. This is not generally true of SQL queries. Over time, however, the differences between relational databases and MapReduce systems are likely to blur—both as relational databases start incorporating some of the ideas from MapReduce (such as Aster Data's and Greenplum's databases) and, from the other direction, as higher-level query languages built on MapReduce (such as Pig and Hive) make MapReduce systems more approachable to traditional database programmers.

**Grid Computing**

The High Performance Computing (HPC) and Grid Computing communities have been doing large-scale data processing for years, using such APIs as Message Passing Interface (MPI). Broadly, the approach in HPC is to distribute the work across a cluster of machines, which access a shared filesystem, hosted by a SAN. This works well for predominantly compute-intensive jobs, but becomes a problem when nodes need to access larger data volumes (hundreds of gigabytes, the point at which MapReduce really starts to shine), since the network bandwidth is the bottleneck and compute nodes become idle. MapReduce tries to collocate the data with the compute node, so data access is fast since it is local.# This feature, known as *data locality*, is at the heart of MapReduce and is the reason for its good performance. Recognizing that network bandwidth is the most precious resource in a data center environment (it is easy to saturate network links by copying data around), MapReduce implementations go to great lengths to conserve it by explicitly modelling network topology. Notice that this arrangement does not preclude high-CPU analyses in MapReduce.

MPI gives great control to the programmer, but requires that he or she explicitly handle the mechanics of the data flow, exposed via low-level C routines and constructs, such as sockets, as well as the higher-level algorithm for the analysis. MapReduce operates only at the higher level: the programmer thinks in terms of functions of key and value pairs, and the data flow is implicit.

Coordinating the processes in a large-scale distributed computation is a challenge. The hardest aspect is gracefully handling partial failure—when you don't know if a remote process has failed or not—and still making progress with the overall computation.

MapReduce spares the programmer from having to think about failure, since the implementation detects failed map or reduce tasks and reschedules replacements on machines that are healthy. MapReduce is able to do this since it is a *shared-nothing* architecture, meaning that tasks have no dependence on one other. (This is a slight oversimplification, since the output from mappers is fed to the reducers, but this is under the control of the MapReduce system; in this case, it needs to take more care rerunning a failed reducer than rerunning a failed map, since it has to make sure it can retrieve the necessary map outputs, and if not, regenerate them by running the relevant maps again.) So from the programmer's point of view, the order in which the tasks run doesn't matter. By contrast, MPI programs have to explicitly manage their own checkpointing and recovery, which gives more control to the programmer, but makes them more difficult to write.

MapReduce might sound like quite a restrictive programming model, and in a sense it is: you are limited to key and value types that are related in specified ways, and mappers and reducers run with very limited coordination between one another (the mappers pass keys and values to reducers). A natural question to ask is: can you do anything useful or nontrivial with it? The answer is yes. MapReduce was invented by engineers at Google as a system for building production search indexes because they found themselves solving the same problem over and over again (and MapReduce was inspired by older ideas from the functional programming, distributed computing, and database communities), but it has since been used for many other applications in many other industries. It is pleasantly surprising to see the range of algorithms that can be expressed in MapReduce, from image analysis, to graph-based problems, to machine learning algorithms.* It can't solve every problem, of course, but it is a general data-processing tool.

**Volunteer Computing** When people first hear about Hadoop and MapReduce, they often ask, "How is it different from SETI@home?" SETI, the Search for Extra-Terrestrial Intelligence, runs a project called SETI@home in which volunteers donate CPU time from their otherwise idle computers to analyze radio telescope data for signs of intelligent life outside earth. SETI@home is the most well-known of many *volunteer computing* projects; others include the Great Internet Mersenne Prime Search (to search for large prime numbers) and Folding@home (to understand protein folding and how it relates to disease). Volunteer computing projects work by breaking the problem they are trying to solve into chunks called *work units*, which are sent to computers around the world to be analyzed. For example, a SETI@home work unit is about 0.35 MB of radio telescope data, and takes hours or days to analyze on a typical home computer. When the analysis is completed, the results are sent back to the server, and the client gets another work unit. As a precaution to combat cheating, each work unit is sent to three different machines and needs at least two results to agree to be accepted. Although SETI@home may be superficially similar to MapReduce (breaking a problem into independent pieces to be worked on in parallel), there are some significant differences. The SETI@home problem is very CPU-intensive, which makes it suitable for running on hundreds of thousands of computers across the world,† since the time to transfer the work unit is dwarfed by the time to run the

computation on it. Volunteers are donating CPU cycles, not bandwidth. MapReduce is designed to run jobs that last minutes or hours on trusted, dedicated hardware running in a single data center with very high aggregate bandwidth interconnects.By contrast, SETI@home runs a perpetual computation on untrusted machines on the Internet with highly variable connection speeds and no data locality.

**A Brief History of Hadoop**

Hadoop was created by Doug Cutting, the creator of Apache Lucene, the widely used text search library. Hadoop has its origins in Apache Nutch, an open source web search engine, itself a part of the Lucene project.

**The Origin of the Name "Hadoop"**

The name Hadoop is not an acronym; it's a made-up name. The project's creator, Doug Cutting, explains how the name came about: The name my kid gave a stuffed yellow elephant. Short, relatively easy to spell and pronounce, meaningless, and not used elsewhere: those are my naming criteria. Kids are good at generating such. Googol is a kid's term. Subprojects and "contrib" modules in Hadoop also tend to have names that are unrelated to their function, often with an elephant or other animal theme ("Pig," for example). Smaller components are given more descriptive (and therefore more mundane) names. This is a good principle, as it means you can generally work out what something does from its name. For example, the jobtracker‡ keeps track of MapReduce jobs. Building a web search engine from scratch was an ambitious goal, for not only is the software required to crawl and index websites complex to write, but it is also a challenge to run without a dedicated operations team, since there are so many moving parts. It's expensive, too: Mike Cafarella and Doug Cutting estimated a system supporting a 1-billion-page index would cost around half a million dollars in hardware, with a monthly running cost of $30,000.§ Nevertheless, they believed it was a worthy goal, as it would open up and ultimately democratize search engine algorithms. Nutch was started in 2002, and a working crawler and search system quickly emerged. However, they realized that their architecture wouldn't scale to the billions of pages on the Web. Help was at hand with the publication of a paper in 2003 that described the architecture of Google's distributed filesystem, called GFS, which was being used in production at Google.‖ GFS, or something like it, would solve their storage needs for the very large files generated as a part of the web crawl and indexing process. In particular, GFS would free up time being spent on administrative tasks such as managing storage nodes. In 2004, they set about writing an open source implementation, the Nutch Distributed Filesystem (NDFS). In 2004, Google published the paper that introduced MapReduce to the world.# Early in 2005, the Nutch developers had a working MapReduce implementation in Nutch, and by the middle of that year all the major Nutch algorithms had been ported to run using MapReduce and NDFS. NDFS and the MapReduce implementation in Nutch were applicable beyond the realm of search, and in February 2006 they moved out of Nutch to form an independent subproject of Lucene called Hadoop. At around the same time, Doug Cutting joined Yahoo!, which provided a dedicated team and the resources to turn Hadoop into a system that ran at web scale (see sidebar). This was demonstrated in February 2008 when Yahoo! announced that it production search index was being generated by a 10,000-core Hadoop cluster.

In January 2008, Hadoop was made its own top-level project at Apache, confirming its success and its diverse, active community. By this time, Hadoop was being used by many other companies besides Yahoo!, such as Last.fm, Facebook, and the *New York Times*. Some applications are covered in the case studies in Chapter 16 and on the Hadoop wiki. In one well-publicized feat, the *New York Times* used Amazon's EC2 compute cloud to crunch through four terabytes of scanned archives from the paper converting them to PDFs for the Web.† The processing took less than 24 hours to run using 100 machines, and the project probably wouldn't have been embarked on without the combination of Amazon's pay-by-the-hour model (which allowed the NYT to access a large number of machines for a short period) and Hadoop's easy-to-use parallel programming model. In April 2008, Hadoop broke a world record to become the fastest system to sort a terabyte of data. Running on a 910-node cluster, Hadoop sorted one terabyte in 209 seconds (just under 3½ minutes), beating the previous year's winner of 297 seconds (described in detail in "TeraByte Sort on Apache Hadoop" on page 553). In November of the same year, Google reported that its MapReduce implementation sorted one terabyte in 68 seconds.‡ As the first edition of this book was going to press (May 2009), it was announced that a team at Yahoo! used Hadoop to sort one terabyte in 62 seconds.

**Apache Hadoop and the Hadoop Ecosystem**

Although Hadoop is best known for MapReduce and its distributed filesystem (HDFS, renamed from NDFS), the term is also used for a family of related projects that fall under the umbrella of infrastructure for distributed computing and large-scale data processing. Most of the core projects covered in this book are hosted by the Apache Software Foundation, which provides support for a community of open source software projects, including the original HTTP Server from which it gets its name. As the Hadoop ecosystem grows, more projects are appearing, not necessarily hosted at Apache, which provide complementary services to Hadoop, or build on the core to add higher-level abstractions.

The Hadoop projects that are covered in this book are described briefly here:

***Common***

A set of components and interfaces for distributed filesystems and general I/O (serialization, Java RPC, persistent data structures).

***Avro***

A serialization system for efficient, cross-language RPC, and persistent data storage.

***MapReduce***

A distributed data processing model and execution environment that runs on large clusters of commodity machines.

***HDFS***

A distributed filesystem that runs on large clusters of commodity machines.

***Pig***

A data flow language and execution environment for exploring very large datasets Pig runs on HDFS and MapReduce clusters.

***Hive***

A distributed data warehouse. Hive manages data stored in HDFS and provides a query language based on SQL (and which is translated by the runtime engine to MapReduce jobs) for querying the data.

***HBase***

A distributed, column-oriented database. HBase uses HDFS for its underlying storage, and supports both batch-style computations using MapReduce and point queries (random reads).

***Zookeeper***

A distributed, highly available coordination service. ZooKeeper provides primitives such as distributed locks that can be used for building distributed applications.

***Sqoop***

A tool for efficiently moving data between relational databases and HDFS.