O'REILLY®

Real-World Hadoop

Ted Dunning & Ellen Friedman

Real-World Hadoop

Ted Dunning

Ellen Friedman

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

DEDICATION

The authors dedicate this book with gratitude to Yorick Wilks, Fellow of the British Computing Socie and Professor Emeritus in the Natural Language Processing Group at University of Sheffield, Senio Research Fellow at the Oxford Internet Institute, Senior Research Scientist at the Florida Institute fo Human and Machine Cognition, and an extraordinary person.

Yorick mentored Ted Dunning as Department Chair and his graduate advisor during Ted's doctora studies in Computing Science at the University of Sheffield. He also provided guidance as Ted's supervisor while Yorick was Director of the Computing Research Laboratory, New Mexico State University, where Ted did research on statistical methods for natural language processing (NLP). Yorick's strong leadership showed that critical and open examination of a wide range of ideas is th foundation of real progress. Ted can only hope to try to live up to that ideal.

We both are grateful to Yorick for his outstanding and continuing contributions to computing scienc especially in the fields of artificial intelligence and NLP, through a career that spans five decades. His brilliance in research is matched by a sparkling wit, and it is both a pleasure and an inspiration know him.

These links provide more details about Yorick's work:

http://staffwww.dcs.shef.ac.uk/people/Y.Wilks/

http://en.wikipedia.org/wiki/Yorick_Wilks

Preface

This book is for you if you are interested in how Apache Hadoop and related technologies can addres problems involving large-scale data in cost-effective ways. Whether you are new to Hadoop or a seasoned user, you should find the content in this book both accessible and helpful.

Here we speak to business team leaders, CIOs, business analysts, and technical developers to explain in basic terms how NoSQL Apache Hadoop and Apache HBase–related technologies work to meet bi data challenges and the ways in which people are using them, including using Hadoop in production. Detailed knowledge of Hadoop is not a prerequisite for this book. We do assume you are rougly familiar with what Hadoop and HBase are, and we focus mainly on how best to use them to advantag The book includes some suggestions for best practice, but it is intended neither as a technical reference nor a comprehensive guide to how to use these technologies, and people can easily read it whether or not they have a deeply technical background. That said, we think that technical adepts wil also benefit, not so much from a review of tools, but from a sharing of experience.

Based on real-world situations and experience, in this book we aim to describe how Hadoop-based systems and new NoSQL database technologies such as Apache HBase have been used to solve a wid variety of business and research problems. These tools have grown to be very effective and production-ready. Hadoop and associated tools are being used successfully in a variety of use cases and sectors. To choose to move into these new approaches is a big decision, and the first step is to recognize how these solutions can be an advantage to achieve your own specific goals. For those just getting started, we describe some of the pre-planning and early decisions that can make the process easier and more productive. People who are already using Hadoop and NoSQL-based technologies wi find suggestions for new ways to gain the full range of benefits possible from employing Hadoop wel

In order to help inform the choices people make as they consider these new solutions, we've put together:

- An overview of the reasons people are turning to these technologies
- A brief review of what the Hadoop ecosystem tools can do for you
- A collection of tips for success
- A description of some widely applicable prototypical use cases
- Stories from the real world to show how people are already using Hadoop and NoSQL successfull for experimentation, development, and in production

This book is a selection of various examples that should help guide decisions and spark your ideas fo how best to employ these technologies. The examples we describe are based on how customers use th Hadoop distribution from MapR Technologies to solve their big data needs in many situations across

range of different sectors. The uses for Hadoop we describe are not, however, limited to MapR. When a particular capability is MapR-specific, we call that to your attention and explain how this would be handled by other Hadoop distributions. Regardless of the Hadoop distribution you choose, you should be able to see yourself in these examples and gain insights into how to make the best use of Hadoop for your own purposes.

How to Use This Book

If you are inexperienced with Apache Hadoop and NoSQL non-relational databases, you will find basic advice to get you started, as well as suggestions for planning your use of Hadoop going forward.

If you are a seasoned Hadoop user and have familiarity with Hadoop-based tools, you may want to mostly skim or even skip Chapter 2 except as a quick review of the ecosystem.

For all readers, when you reach Chapter 5 and Chapter 6, consider them together. The former explain some of the most rewarding of prototypical Hadoop use cases so that you see what your options are. Chapter 6 then shows you how Hadoop users are putting those options together in real-world settings to address many different problems.

We hope you find this approach helpful.

—Ted Dunning and Ellen Friedman, January 2015

Chapter 1. Turning to Apache Hadoop and NoSQL Solutions

Some questions are easier to answer than others. In response to the question, *"Is Hadoop ready for production?,"* the answer is, simply, "yes."

This answer may surprise you, given how young the Apache Hadoop technology actually is. You may wonder on what basis we offer this definitive response to the question of Hadoop's readiness for production. The key reason we say that it is ready is simply because so many organizations are already using Hadoop in production and doing so successfully. Of course, being ready for production is not the same thing as being a mature technology.

Will Hadoop-based technologies change over the next few years? Of course they will. This is a rapidly expanding new arena, with continual improvements in the underlying technology and the appearance of innovative new tools that run in this ecosystem. The level of experience and understanding among Hadoop users is also rapidly increasing. As Hadoop and its related technologies continue progress toward maturity, there will be a high rate of change. Not only will new features and capabilities be added, these technologies will generally become easier to use as they become more refined.

Are these technologies a good choice for you? The answer to that question is more complicated, as it depends on your own project goals, your resources, and your willingness to adopt new approaches. Even with a mature technology, there would be a learning curve to account for in planning the use of something different; with a maturing technology you also have to account for a cost of novelty and stay adaptable to rapid change in the technology. Hadoop and NoSQL solutions are still young, so not only are the tools themselves still somewhat short of maturity, there is also a more limited pool of experienced users from which to select when building out your own team than with some older approaches.

Even so, Hadoop adoption is widespread and growing rapidly. For many, the question is no longer whether or not to turn to Hadoop and NoSQL solutions for their big data challenges but rather, *"What are the best ways to use Hadoop and NoSQL to carry our projects successfully?"* and *"When should we start?"*

This book aims to help answer these questions by providing a conversation around the choices that drive success in big data projects, by sharing tips that others have found useful, and by examining a selection of use cases and stories from successful Hadoop-based projects. What makes this collection of use cases different is that we include examples of how people are already using Hadoop in production and in near-production settings. We base our stories and recommendations on the experiences of real teams running real workloads. Of course, we do not focus only on Hadoop in production—we also provide advice to help you get started and to use Hadoop successfully in development.

When is the time right to give Hadoop a try? There is no "right" answer to that question, as each situation is different, but now may be a good time to give Hadoop a try even if you're not yet ready t consider it in a production setting for your own projects. If you start now, you will not be a Hadoop pioneer—the true pioneers are the ones already using it in production. But there is still an early-mov advantage to be had for those starting now. For one thing, you will find out if this technology holds promise for your situation. For another, you will begin building Hadoop expertise within your organization, which may prove very valuable. Even if you do not at present have an urgent need for a Hadoop-based solution, it's very likely you will need it or a solution similar to it soon. Having teams who are savvy about using Hadoop is an investment in the future.

A Day in the Life of a Big Data Project

Before we look in detail at what Hadoop is and how you might use it, let's start with a look at an unusual Hadoop-based project that is changing society in fundamental ways. The story begins with this challenge: suppose you need to be able to identify every person in India, uniquely and reliably—all 1.2 billion of them. And suppose you need to be able to authenticate this identification for any individual who requests it, at any time, from any place in India, in less than a second. Does this soun sufficiently challenging?

That description is the central mission for India's Aadhaar project, the Unique Identification Authority of India (UIDAI). The project provides a unique 12-digit, government-issued identificatio number that is tied to biometric data to verify the identity and address for each person in India. The biometric data includes an iris scan of both eyes plus multipoint data from the fingerprint pattern of all 10 fingers, as suggested by the illustration in Figure 1-1. The unique Aadhaar ID number is a random number, and it is assigned without classification based on caste, religion, or creed, assuring a openness and equality to the project.

Figure 1-1. Unique Identification Authority of India (UIDAI) is running the Aadhaar project, whose goal is to provide a unique 12-digit identification number plus biometric data to authenticate to every one of the roughly 1.2 billion people in India. This is the largest scale ever reached by a biometric system. (Figure based on image by Christian Als/Panos Pictures.)

The need for such an identification program and its potential impact on society is enormous. In India there is no social security card, and much of the population lacks a passport. Literacy rates are relatively low, and the population is scattered across hundreds of thousands of villages. Without adequately verifiable identification, it has been difficult for many citizens to set up a bank account or otherwise participate in a modern economy.

For India's poorer citizens, this problem has even more dire consequences. The government has extensive programs to provide widespread relief for the poor—for example, through grain subsidies t those who are underfed and through government-sponsored work programs for the unemployed. Yet many who need help do not have access to benefit programs, in part because of the inability to verify who they are and whether they qualify for the programs. In addition, there is a huge level of so-called "leakage" of government aid that disappears to apparent fraud. For example, it has been estimated th over 50% of funds intended to provide grain to the poor goes missing, and that fraudulent claims for "ghost workers" siphon off much of the aid intended to create work for the poor.

The Aadhaar program is poised to change this. It is in the process of creating the largest biometric database in the world, one that can be leveraged to authenticate identities for each citizen, even on sit in rural villages. A wide range of mobile devices from cell phones to micro scanners can be used to enroll people and to authenticate their identities when a transaction is requested. People will be able t make payments at remote sites via micro-ATMs. Aadhaar ID authentication will be used to verify qualification for relief food deliveries and to provide pension payments for the elderly.

Implementation of this massive digital identification system is expected to save the equivalent of millions and perhaps billions of dollars each year by thwarting efforts at fraud. While the UIDAI project will have broad benefits for the Indian society as a whole, the greatest impact will be for the poorest people.

The UIDAI project is a Hadoop-based program that is well into production. At the time of this writin over 700 million people have been enrolled and their identity information has been verified. The target is to reach a total of at least 100 crore (1 billion) enrollments during 2015. Currently the enrollment rate is about 10 million people every 10 days, so the project is well positioned to meet tha target.

From a technical point of view, what are the requirements for such an impressive big data project? Scalability and reliability are among the most significant requirements, along with capability for ver high performance. This challenge starts with the enrollment process itself. Once address and biometr data are collected for a particular individual, the enrollment must undergo deduplication. Deduplication for each new enrollment requires the processing of comparisons against billions of records. As the system grows, deduplication becomes an even greater challenge.

Meanwhile, the Aadhaar digital platform is also busy handling authentication for each transaction conducted by the millions of people already enrolled. Authentication involves a profile lookup, and i is required to support thousands of concurrent transactions with response times on the order of 100m The authentication system was designed to run on Hadoop and Apache HBase. It currently uses the MapR distribution for Hadoop. Rather than employ HBase, the authentication system uses MapR-DB a NoSQL database that supports the HBase API and is part of MapR. We'll delve more into how MapR-DB and other technologies interrelate later in this chapter and in Chapter 2. In addition to beir able to handle the authentication workload, the Aadhaar authentication system also has to meet strict availability requirements, provide robustness in the face of machine failure, and operate across multiple datacenters.

Chief Architect of the Aadhaar project, Pramod Varma, has pointed out that the project is "built on sound strategy and a strong technology backbone." The most essential characteristics of the technology involved in Aadhaar are to be highly available and to be able to deliver sub-second performance.

From Aadhaar to Your Own Big Data Project

The Aadhaar project is not only an impressive example of vision and planning, it also highlights the ability of Hadoop and NoSQL solutions to meet the goals of an ambitious program. As unusual as thi project seems—not everyone is trying to set up an identification program for a country the size of India—there are commonalities between this unusual project and more ordinary projects both in term of the nature of the problems being addressed and the design of successful solutions. In other words, in this use case, as in the others we discuss, you should be able to see your own challenges even if yo work in a very different sector.

One commonality is data size. As it turns out, while Aadhaar is a large-scale project with huge social impact and fairly extreme requirements for high availability and performance, as far as data volume

goes, it is not unusually large among big data projects. Data volumes in the financial sector, for instance, are often this size or even larger because so many transactions are involved. Similarly, machine-produced data in the area of the industrial Internet can easily exceed the volumes of Aadhaa The need for reliable scalability in India's project applies to projects in these cases as well.

Other comparisons besides data volume can be drawn between Aadhaar and more conventional use cases. If you are involved with a large retail business, for instance, the idea of identity or profile lookup is quite familiar. You may be trying to optimize an advertising campaign, and as part of the project you need to look up the profiles of customers to verify their location, tastes, or buying behaviors, possibly at even higher rates than needed by Aadhaar. Such projects often involve more than simple verification, possibly relying on complex analytics or machine learning such as predictiv filtering, but the need to get the individual profile data for a large number of customers is still an essential part of implementing such a solution. The challenging performance requirements of Aadhaa are also found in a wide range of projects. In these situations, a Hadoop-based NoSQL solution such HBase or MapR-DB provides the ability to scale horizontally to meet the needs of large volume data and to avoid traffic problems that can reduce performance.

What Hadoop and NoSQL Do

The most fundamental reason to turn to Hadoop is for the ability to handle very large-scale data at reasonable cost in a reasonable time. The same applies to Hadoop-based NoSQL database management solutions such as HBase and MapR-DB. There are other choices for large-scale distributed computing, including Cassandra, Riak, and more, but Hadoop-based systems are widely used and are the focus of our book. Figure 1-2 shows how interest in Hadoop has grown, based on search terms used in Google Trends.

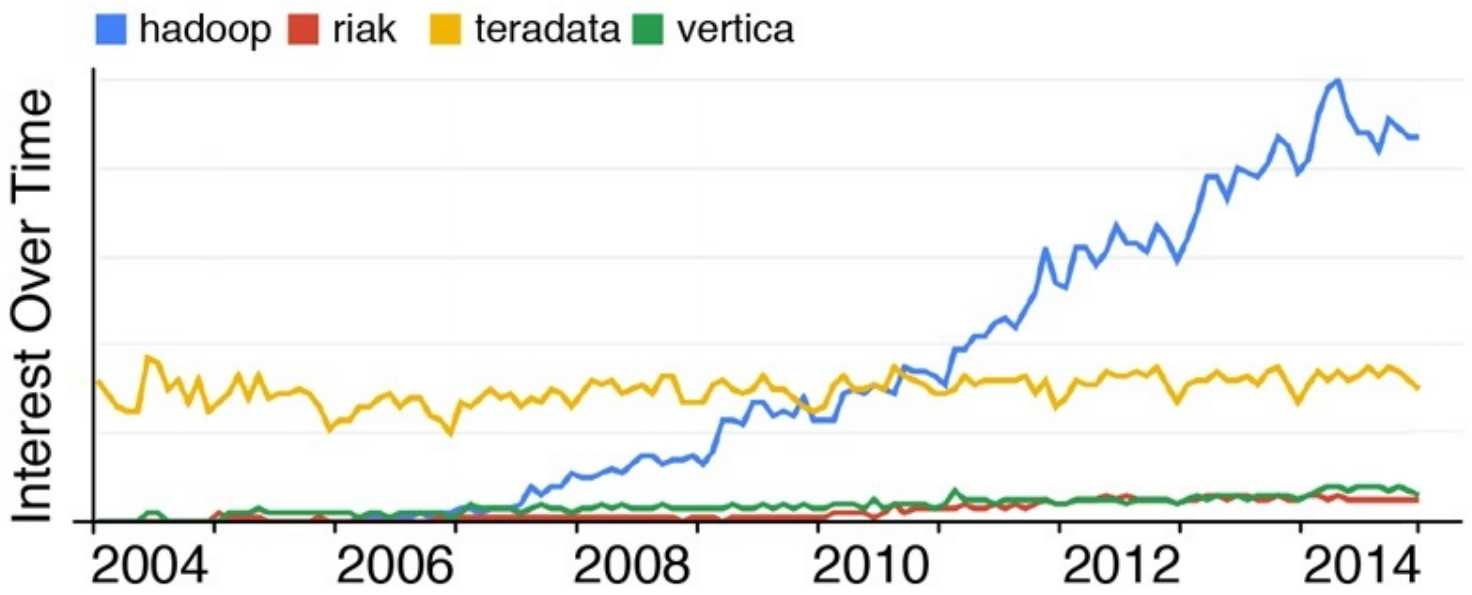

Figure 1-2. Google Trends shows a sharp rise in popularity of the term "hadoop" in searches through recent years, suggesting an increased interest in Hadoop as a technology. We did not include "cassandra" as a search term because its popularity as a personal name means there is no easy way to disambiguate results for the database from results for human names.

In addition to the ability to store large amounts of data in a cost-effective way, Hadoop also provides mechanisms to greatly improve computation performance at scale. Hadoop involves a distributed storage layer plus a framework to coordinate that storage. In addition, Hadoop provides a computational framework to support parallel processing. In its original form, Hadoop was developed as an open source Apache Foundation project based on Google's MapReduce paradigm. Today there are a variety of different distributions for Hadoop.

One of the key aspects of this distributed approach to computation involves dividing large jobs into a collection of smaller tasks that can be run in parallel, completely independently of each other. The outputs of these tasks are shuffled and then processed by other tasks. By running tasks in parallel, job can be completed more quickly, which allows the system to have very high throughput. The original Hadoop MapReduce provided a framework that allowed programs to be built in a relatively straightforward way that could run in this style and thus provided highly scalable computation. MapReduce programs run in batch, and they are useful for aggregation and counting at large scale.

Another key factor for performance is the ability to move the computation to where the data is stored rather than having to move data to the computation. In traditional computing systems, data storage is segregated from computational systems. With Hadoop, there is no such segregation, and programs ca run on the same machines that store the data. The result is that you move only megabytes of program instead of terabytes of data in order to do a very large-scale computation, which results in greatly improved performance.

The original Hadoop MapReduce implementation was innovative but also fairly limited and inflexible. MapReduce provided a start and was good enough to set in motion a revolution in scalable computing. With recent additions to the Hadoop ecosystem, more advanced and more flexible system are also becoming available. MapReduce is, however, still an important method for aggregation and counting, particularly in certain situations where the batch nature of MapReduce is not a problem. More importantly, the basic ideas on which MapReduce is based—parallel processing, data locality, and the shuffle and re-assembly of results—can also be seen underlying new computational tools such as Apache Spark, Apache Tez, and Apache Drill. Most likely, more tools that take advantage of these basic innovations will also be coming along.

All of these computational frameworks run on the Hadoop storage layer, as shown in Figure 1-3. An important difference in these new systems is that they avoid storing every intermediate result to disk which in turn provides improved speed for computation. Another difference is that the new systems allow computations to be chained more flexibly. The effect of these differences is better overall performance in some situations and applicability to a wider range of problems.

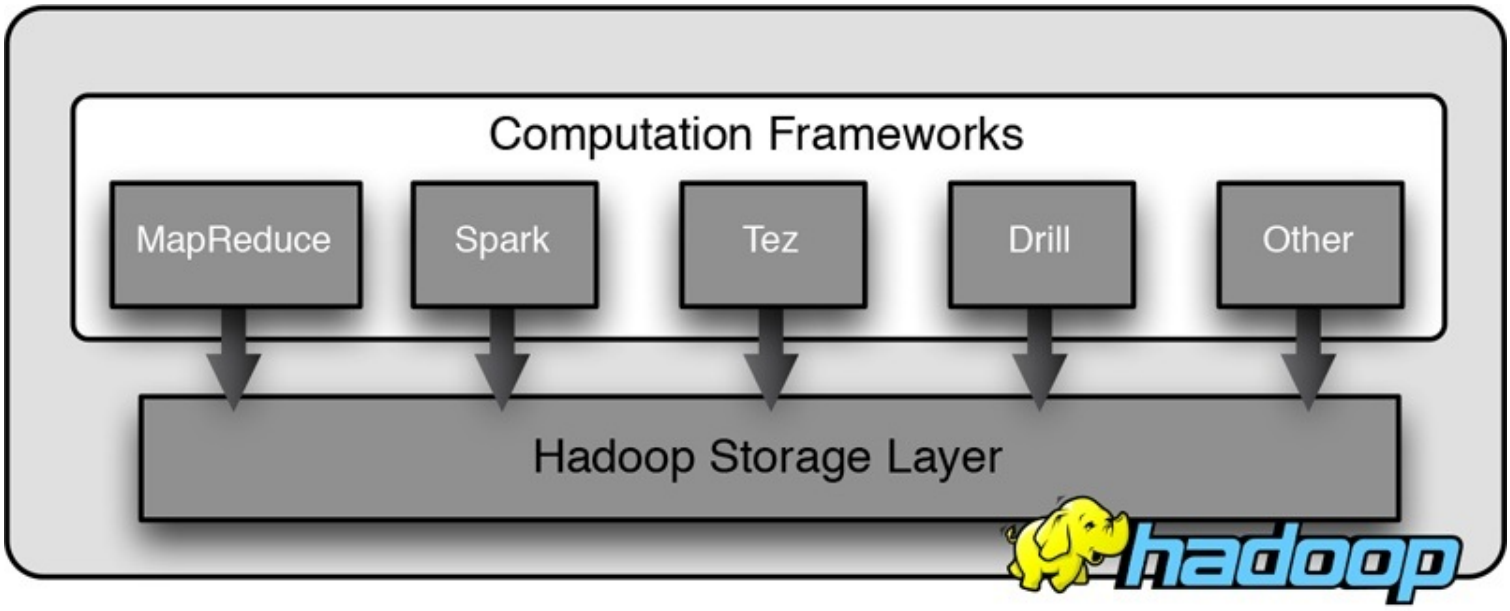

Figure 1-3. A variety of different computational frameworks for parallel processing are available to run on the Apache Hadoop storage layer for large data systems.

In addition to the ability to scale horizontally at low cost and to perform large-scale computations very efficiently and rapidly, Hadoop-based technologies are also changing the game by encouraging the use of new types of data formats. Both files and NoSQL databases allow you to use a wide range data formats, including unstructured or semistructured data. Concurrent with the development of Hadoop's computational capabilities, there have been dramatic improvements in our understanding c how to store data in flat files and NoSQL databases. These new ways for structuring data greatly expand the options and provide a greater degree of flexibility than you may be used to. We say more about new data formats among the tips in Chapter 4.

NoSQL nonrelational databases can augment the capabilities of the flat files with the ability to acces and update a subset of records, each identified by a unique key, rather than having to scan an entire file as you would with flat files.

With their ability to scale in a cost-effective way and to handle unstructured data, NoSQL databases provide a powerful solution for a variety of use cases, but they should not be thought of as a replacement for the function of traditional databases. This distinction is described in more detail in Chapter 2, but the essential point to note is that each of these technologies—NoSQL databases and traditional relational databases (RDBMS)—should be used for the things they do best. Some NoSQL databases have a better ability to scale and to do so while keeping costs down. They can handle raw, unstructured data that also affects use cases for which they are well suited. In contrast, there is a pric to be paid with RDBMS (literally and in terms of the effort required for processing and structuring data for loading). The reward for this cost, however, can be extremely good performance with RDBM for specific, highly defined tasks such as critical path billing and standardized reporting. However, to get these benefits, the data must already be prepared for the tasks.

When Are Hadoop and NoSQL the Right Choice?

The need to handle large amounts of data cost effectively has led to the development of scalable distributed computing systems such as those discussed in this book, based on Hadoop and on NoSQL databases. But these new technologies are proving so effective that they go beyond providing solutio for existing projects; they are inviting exploration into new problems that previously would have seemed impractical to consider.

A key distinction of these systems from previous ones is flexibility. This flexibility is manifested in the platform itself through the capability to handle new data types and multiple computational model as well as to scale to very large data sizes. Hadoop adds flexibility in your choices by being able to store essentially "raw" data and make decisions about how to process and use it later. Flexibility is also manifested in the ways that developers are combining diverse data streams and new analysis techniques in a single platform. In the MapR distribution for Hadoop, there is further flexibility to us existing non-Hadoop applications side by side on the same systems and operate on the same data as Hadoop applications.

Chapter 2 provides you with an overview of the functions supported by Hadoop ecosystem tools, whi Chapter 3 explains some of the extra capabilities of MapR's distribution so that you will be better ab to extrapolate from our examples to your own situation, whichever Hadoop distribution you choose t try. In Chapter 4, we provide a list of tips for success when working with Hadoop that offer help for newcomers and for experienced Hadoop users.

In order to help you better understand how these technologies may bring value to your own projects, this book also describes a selection of Hadoop use cases based on how MapR customers are using it. MapR currently has over 700 paying customers across a wide range of sectors including financial services, web-based businesses, manufacturing, media, telecommunications, and retail. Based on those, we have identified a variety of key usage patterns that we describe in Chapter 5 as well as telling example stories in Chapter 6 that show how customers combine these solutions to meet their own big data challenges.

Are the examples described in this book MapR specific? For the most part, no. There are some aspect of what people are doing that rely on specific MapR features or behaviors, but we've tried to call tho to your attention and point out what the alternatives would be. The use cases and customer stories have been chosen to show the power of Hadoop and NoSQL solutions to provide new ways to solve problems at scale. Regardless of the Hadoop distribution you choose, the material in this book should help guide you in the decisions you'll need to make in order to plan well and execute successful big data projects.

Chapter 2. What the Hadoop Ecosystem Offers

Apache Hadoop and related technologies are rapidly evolving, and as such they are spawning a large array of new tools. As people see growing value and expanding use cases in this area, the number of tools to address significant needs also grows. This trend is good news in that it provides a wide range of functions to support the activities you may want to carry out in this new arena. However, the wealt of new and unfamiliar tools can feel a bit overwhelming.

In order to help guide you through the choices being offered in the Hadoop ecosystem in a meaningfu way, we take a look here at some of the key actions that are commonly desired in Hadoop and NoSQ use cases and provide you with a description of some of the tools widely used to carry out those operations. This is by no means a full catalog of what's available, nor is it a how-to manual for using the tools. Instead, our focus is on the issues associated with functions common to many Hadoop-base projects. This high-level view of what various Hadoop ecosystem tools are used for is intended to hel you to assess tools of interest to you, whether or not they're included in our list, in terms of how they may be helpful for your projects.

To get started, we've put together a chart of some major needs that are common to many use cases ar that shows you a few of the tools associated with each one. In Table 2-1, a selection of the most prominent tools in the Hadoop ecosystem are broken down roughly by time-scale and by their typica purposes.

Table 2-1. Hadoop ecosystem tools broken down by time-scale and general purpose. Note that all tools generally work on all platforms, with a few exceptions. For instance, NFS and direct file access allow realtime updates and POSIX semantics on MapR but not on other platforms. Also, some interface methods such as ODBC and NFS (depending on platform, as noted) open up a huge range of tools. Note that Impala and ElasticSearch are open source with a closed community. MapR-DB, Teradata, and Tableau are closed source. The rest listed here are fully open source projects.

	Ingest	Process	Persist	Extract
Batch	Flume, sqoop, Kafka, NFS	Hive, Pig	Files	NFS, Teradata, and other connectors
Ad hoc	Flume, NFS	Spark, Impala, Drill, Hive (soon), Solr, ElasticSearch	File, HBase, MapR-DB	ODBC tools such as Tableau, direct web access
Streaming and realtime	Flume, Kafka, NFS	Spark streaming, Storm, Samza	HBase, MapR-DB, file (on some platforms)	HBase, MapR-DB, NFS

Typical Functions

Table 2-1 divides the functions to be done in big data applications into *ingestion*, *data processing or transformation*, *persistence*, and *extraction*. Ingestion is the process of getting data into a system wit minimal processing or transformation applied during ingestion. Processing is where all significant computing and transformation is done. The most common operation in processing raw data is that it i aggregated into summaries or arranged into profiles. Data is commonly persisted after processing, bu in Hadoop systems, data is also commonly persisted in nearly raw form as it is ingested but before it is processed. The retention of relatively raw data makes it possible for errors in stream processing to be corrected. It is also advantageous in that it widens the opportunities for data exploration and avoid discarding data that may later be of great value, as we discuss further in Chapter 4, Chapter 5, and Chapter 6.

Different forms of persistence lend themselves to different kinds of processing. For example, files ca be used to achieve very high scan rates that are particularly useful in batch programming, while HBa or MapR-DB are very useful in real time or streaming processing where updates may have to be mad each time a record is processed. Finally, data must somehow be transmitted to other systems via som form of extraction.

For each kind of function, it is typical that the tool of choice depends strongly on the time scale in which the processing must be done.

The time scale dealt with in Table 2-1 ranges from batch, through ad hoc, to streaming and realtime. Batch processing is typically done on all the data that accumulates over a period of minutes to days o even a month or longer. The emphasis in batch processing is total throughput, and it usually doesn't matter how long it takes to process any single input record as long many records are processed quickly.

In ad hoc processing, the emphasis is on a quick (in human terms) response, but the data being processed is very much like the input for a batch process in that it typically consists of all of the data available or all of the data for a recent time period. In fact, ad hoc processing can be thought of as batch processing that is initiated by some user action instead of being based on a schedule. Ad hoc processing is sometimes mislabeled as realtime because there is a user-visible response time, but this is a serious misnomer that could lead to some confusion about which tool is appropriate to use.

With streaming or realtime processing, records are processed as they arrive or in very small batches known as micro batches. Realtime processing adds the additional constraint that records must not onl be processed as they arrive, but that processing must complete before a pre-specified deadline passes Requiring that records be processed one at a time is more expensive than processing large batches of records since it does not allow certain economies of scale that are possible in batch processing.

The rest of this chapter will provide a quick inventory of the tools used in the Hadoop ecosystem to carry out these functions.

Data Storage and Persistence

The most obvious requirement for data storage in this new arena is for scalability, both in terms of

starting with large amounts of data and in being able to adapt to growing data volumes in future. For large-scale and long-term storage, reliability is, of course, a critical requirement for most projects.

Data storage comes in different forms that have different virtues. Storage as files has the virtue of being fast to scan and is therefore well suited for batch processing. It is difficult, however, to find a particular record in large input files or to find a number of records that pertain to a single individual. Furthermore, updates to files can be difficult to coordinate. These actions are much better supported by some sort of database, and in the Hadoop ecosystem, that means Apache HBase or MapR-DB. Reading or updating single records is easy with these NoSQL databases at the significant cost in scanning performance relative to flat files.

Another aspect of storage that is almost as important as scalability is the shift toward storing a wide variety of data sources, including unstructured or semistructured data. This change in what is being stored also reflects the change in when data is processed: in these systems, the storage layer is often used to store raw data at scale and persist it for long periods of time in a relatively raw form. Choices about how to process, extract, and analyze data come after this "ingestion" step, which is a very different approach than the extract, transform, and load (ETL) process that is usual for traditional relational databases, for instance. The traditional style of ETL processing is not required for storage in Hadoop-based systems, although using Hadoop to do ETL *for a traditional RDBMS resource* is a widely beneficial Hadoop use case (more on that in Chapter 5).

For highly scalable flat file storage, the tools of interest here are Hadoop-based. These are Hadoop Distributed File System (HDFS) or the storage layer of the MapR distribution of Hadoop (MapR-FS). Files in both of these systems can be created and accessed using the HDFS API or, in the case of MapR-FS, files also can be created, accessed, and updated using standard file operations via the network file system (NFS).

For persistence of data in a nonrelational NoSQL database, there are a number of popular non-Hadoop choices, including Cassandra, Riak, or the Hadoop-based NoSQL databases such as HBase or MapR-DB, a NoSQL database integrated into the MapR file system.

As mentioned briefly in Chapter 1, NoSQL databases are not intended to completely replace relational databases. Each has its own strengths and should be used for what it does best. NoSQL databases generally have given up some of the capabilities of relational databases such as advanced indexing and transactions in order to allow them to scale to much higher throughput and data sizes than are possible with relational systems.

The data stored in files or NoSQL databases is often very different from the data stored in a traditional relational database. Somewhat ironically, the term "structured" data is typically used to refer to data in traditional records with fields containing primitive values. Data with substantial free-form components such as text is often called "unstructured." The term "semistructured" refers to data that has records with fields but where different records may have different collections of fields, and the contents of the fields may contain sub-fields or lists of record-like objects.

Semistructured data is more expressive than structured data, but it cannot easily be manipulated by traditional query languages like SQL. The expressivity of semistructured data can be a particular advantage in big data systems because it allows data to be denormalized. Denormalization can involve the inclusion of redundant data. For example, more data may be stored inline, which in turn decreases

the degree to which other data sources must be referenced in order to understand the data. The advantages of denormalization can include improved read performance, plus it can allow storing data in its natural state, thus preserving potentially useful information. This is particularly important with flat files and with NoSQL databases, not only because it makes data easier to understand, but also precisely because flat files and NoSQL databases typically lack the strong ability of *relational* databases for records to have references to other records.

> To succeed in big data, it is very important to come to understand both the advantages and the problems of semistructured data.

The use cases in this book will highlight many of the ways that semistructured data can be used for big-data applications.

Data Ingest

Storing data is useful, but unless the data is manufactured from nothing, you have to get it into your cluster from somewhere else. Exactly how you get it in depends on where the data is coming from an how you need to use it.

As shown in Table 2-1, the best choice of ingest method depends in part on whether you are dealing with batch, ad hoc, or streaming and realtime processes. There are several tools that have been developed specifically for ingesting data into Hadoop, but these Hadoop-specific tools were built wit the assumption of batch programming that can limit their utility if you need realtime ingestion. In addition to specialized tools for data ingestion, some Hadoop distributions offer NFS access to the cluster. This can be a good way to get realtime ingestion using standard file-oriented Linux tools, but only if the underlying distributed file system is a realtime file system. For the Hadoop-in-production stories discussed in this book, we've drawn from the experience of companies who use the MapR distribution for Hadoop and are therefore using a realtime distributed file system in the form of MapR-FS.

A function that is needed in the case of streaming data ingestion (or export from a realtime analytics application) is to provide a queuing layer. The benefits of queuing are described in more detail in the tips presented in Chapter 4. Two of the tools listed here, Apache Kafka and Apache Flume, are useful to provide a queue.

Some of the widely used tools for data ingestion in Hadoop are described in detail in the sections tha follow.

Apache Kafka

Kafka is a robust pub-sub (publish, subscribe) framework that allows highly available, dependable message streaming. It is a paradox that a key feature of Kakfa is its small number of features. It has far fewer features and is much less configurable than Flume, which is described later. All Kafka does is store messages relatively reliably and at high volumes and rates. All computational considerations are outside of Kafka's scope. Such computation can be implemented using a variety of computationa

frameworks such as Spark Streaming, Apache Storm, or Apache Samza. This simplicity and focus of Kafka have helped make it very good at what it does.

There are a variety of programs in the Kafka ecosystem that support copying messages to a Hadoop cluster, but this is also commonly done as a side effect of processing messages.

Apache Sqoop

Sqoop is a batch-oriented program to import data from a database or export data back to a database. Sqoop can create files in a variety of formats in a Hadoop cluster. Sqoop is a very useful tool due to the wide range of databases that it supports, but Sqoop is, by design, entirely batch-oriented.

An interesting alternative to using Sqoop is to use database-specific export commands to export table directly to a cluster or import them from the cluster using NFS. A virtue of this alternative approach that it can be very high performance since databases generally have highly optimized table export/bulk import utilities. Using database export/import utilities will not, however, provide you wi the native ability to store the resulting data in an advanced format such as Parquet.

Apache Flume

Flume is a complex tool that allows processing units to be strung together to transport data, typically with the aim of doing minimal ETL on the fly and then storing in HDFS files. Flume is nominally stream based, but a de facto batch orientation is often imposed by storing data in HDFS files. If data pushed instead into a pub-sub framework like Kafka, then true streaming operation can be achieved. Flume has limited and complex provisions for high availability and guaranteed delivery. Flume was originally limited to processing textual data arranged one record per line as is normally done in log files, and there are still echoes of this limitation in various parts of the framework. In general, the complexity of Flume makes the use of Kafka plus either Storm or Spark Streaming a preferable option.

Data Extraction from Hadoop

Data can be extracted from a Hadoop cluster using Sqoop to move data into a database. NFS access is another good way to get data out of a Hadoop cluster if you are using MapR. To export file data on other systems, you can use the command line tools that come with Hadoop distributions to copy individual files or directories of files.

Since the data being extracted from a cluster is typically much smaller than the data ingested into a cluster, the actual moving of the data out of the cluster is not usually a major concern. Getting the da into a file format that is accepted by other systems is typically a more important concern, but format conversion is relatively easy using systems like Apache Pig, Apache Drill, or Apache Hive.

Processing, Transforming, Querying

Processing of data in a Hadoop cluster can be done in a variety of ways including streaming, micro-

batched, batch mode, and by issuing interactive queries. The boundaries between these ways of processing data are not always completely sharp, but the basic distinctions are important to keep in mind. We mention a variety of tools to support each of these functions, and we call out several of the SQL-on-Hadoop query engines (Apache Drill, Apache Spark, and Impala) in more detail at the end of this section.

Streaming

In streaming processing, data is processed one record at a time. This is appropriate for simple enrichment, parsing or extraction, and simple aggregation, but many kinds of processing are not suitable for this style. It can also be difficult to make true streaming programs both efficient and restartable without losing data. Streaming processing of data does give the lowest latencies, however so when latencies really need to be less than a second or so, streaming may be the best option. Currently Apache Storm is the de facto standard for true streaming processing, while Spark Streamin is probably the most common micro-batching environment, as described next.

Micro-batching

Micro-batching involves processing all the records from a short time period (typically seconds to minutes) in a batch. Micro-batching usually has real problems providing very low latency, but it does provide a much simpler environment than streaming, especially when high availability is required. If your use case allows for tens of seconds of latency, then micro-batching with a tool such as Spark Streaming can be ideal.

Batch Processing

True batch processing is typically used for computing complex aggregates or when training models. Batch processing is typically initiated every hour, day, week, or month and computes summaries of large amounts of data accumulated over long time periods. Batch processing is often combined with stream processing or micro-batching in what is known as the lambda architecture to allow small erro in the streaming computation to be corrected. These errors are often due to delays in receiving incoming data or failover or upgrade of some component of the cluster supporting the streaming computation. By allowing small errors to be corrected by a batch process, the system doing the streaming processing can often be made much simpler. Batch processing is most commonly implemented using Apache Hive, Apache Spark, or Apache Pig.

Interactive Query

In some cases, users require the ability to perform bespoke aggregation operations on data according to whatever analysis they are working on at the moment. In such cases, neither batch nor streaming models are particularly appropriate. Instead, an interactive compute model is required with quick response. This is a change from streaming and micro-batching models that makes results available shortly after the data arrives, but where the programs are provided long in advance. Instead, interactive computing provides results shortly after the query arrives and assumes that the data is already in place. Interactive queries are commonly custom aggregations and are often generated by

visualization tools. The most common systems for interactive querying are Impala, Apache Spark, an Apache Drill.

Impala

Impala is a system that is designed to scan flat files at a very high rate to compute the result of aggregation queries written in SQL. The preferred format for data to be queried by Impala is Parquet and Impala is able to use the features of Parquet to great advantage to accelerate the scanning of larg files. For the most part, Impala uses the table management capabilities of Hive to define tables and their schema. The data model used by Impala is currently very similar to the model used by relationa systems, but extension beyond the relational model, including nested data, is planned.

Apache Drill

Apache Drill is a newly graduated top-level Apache project that offers the user an unusual level of flexibility. It provides standard SQL (not SQL-like) query capabilities that can access a surprisingly diverse range of data sources and formats, including nested formats such as Parquet and JSON. Furthermore, Drill queries can be schema-less, allowing flexibility in data exploration. The Drill optimizer is a sophisticated cost-based optimizer that can radically restructure queries based on characteristics of the input files, and it is being extended to understand more about distributed query computation as the software grows further into maturity. Drill also offers useful extensibility, so it is useful tool for business analysts as well as for developers. Like Impala, a key focus with Drill is to make the minimum response time very short so that it can be used for interactive purposes.

Apache Spark

Spark is an ambitious project that has defined an entire new computational framework for running programs in parallel. The key technical innovation in Spark is that it allows parallel datasets to be check-pointed implicitly by remembering how they were computed. In most cases, this avoids the need to write intermediate datasets to persistent storage such as disks, thus avoiding one of the traditional bottlenecks of Hadoop's MapReduce execution model. On top of the basic machinery of distributed in-memory datasets (known as RDDs for resilient distributed datasets) and a fast distributed execution engine, Spark has a large number of subprojects. One key subproject is SparkStreaming. SparkStreaming extends the concept of RDDs by defining D-streams as a sequence of RDDs. Each RDD in a D-stream can be acted on by a parallel program that allows computation to proceed as a series of very quick batch programs, or micro-batches.

Together this collection of query tools provides some attractive options. Impala and Drill allow SQL queries, while Spark allows queries to be written in the Scala programming language or in SQL. Spar and Drill can also be tightly integrated to get the best of both. Spark queries can be executed in micro batched or interactive fashion. Together, these tools provide some very interesting possibilities.

Search Abuse—Using Search and Indexing for Interactive Query

Search engines are not normally considered as processing elements, but they can actually be used ver nicely for many forms of interactive queries. Especially recently, both Apache Solr and ElasticSearch have added a variety of aggregation capabilities that can allow either system to be used to do simple

aggregation queries.

These queries can be reported in the form of a graphical interface such as that provided by Kibana to produce very nice dashboard systems. These systems can also provide limited kinds of drill-down visualizations that can be very handy in situations where you are trying to zero in on the cause of son sort of anomaly.

Visualization Tools

One tool that blurs the batch/interactive boundary a bit is Datameer. Datameer provides an interactiv approximation of large-scale computations combined with a suite of visualization tools. Designed fo analysts who are most comfortable with advanced GUI tooling rather than programming, the Datameer system allows analysts who would otherwise be uncomfortable to build and run jobs either in batch mode or interactively.

Tableau is a widely used visualization product suite that provides interactive visualization tools that have previously been targeted at the analysis of data in data warehouses. With the emergence of interactive SQL query tools like Impala and Apache Drill that are accessible via ODBC, Tableau's tools can now analyze and visualize data on Hadoop clusters as well.

Integration via ODBC and JDBC

Apache Drill and Impala provide access to data on Hadoop clusters via the standard database access protocols ODBC or JDBC. Hive also provides limited access using these interfaces as well. ODBC w originally developed by Microsoft in the early '90s, but it is now widely used on other platforms as well. JDBC was created by Sun in the late '90s to provide a Java equivalent to ODBC. These protocol allow a wide range of standard tools that generate SQL queries to work with Hadoop. These include Tableau (mentioned above), Excel, Sqirrl, Toad, and many others. There are some speed bumps to keep in mind, however. By the nature of these interfaces, they cannot move very large amounts of da out of the cluster, but instead are more suited to invoking large-scale aggregations that return relatively small summaries of larger datasets. This limitation applies to any tool using ODBC or JDBC, of course, but the issue becomes more apparent when you are querying very large datasets, as common with Hadoop clusters.

Accessing Hive (in particular via ODBC or JDBC) can be very frustrating since the number of simultaneous queries is severely limited by the way that queries are funneled through the HiveServer Impala and Drill are much less subject to these limitations than Hive.

In general, the advantages of using ODBC and JDBC to connect a wide range of tools are substantial enough that these issues are likely to be addressed by vendors and the open source community before long.

Chapter 3. Understanding the MapR Distribution for Apache Hadoop

The Hadoop distribution provided by MapR Technologies contains Apache Hadoop and more. We're not just talking about the Hadoop ecosystem tools that ship with MapR—there are many, including almost all of those described in Chapter 2—but rather some special capabilities of MapR itself. Thes MapR-specific characteristics are the topic of this chapter because the real-world stories in this book are based on how MapR customers are using Apache Hadoop and the MapR NoSQL database, MapR-DB, to meet their large-scale computing needs in a variety of projects. The goal is to show you the benefits of Hadoop when used for the right jobs.

To make sure that you get the most out of this book, regardless of what kind of Hadoop distribution you use, we alert you to any aspects of the use cases we describe here that are not directly generalizable because of extra features of MapR not included in other distributions. For example, MapR is API-compatible with Hadoop, so applications written to run on Hadoop will run on MapR, but, in addition, non-Hadoop applications will also run on MapR, and that's unusual. We will describ how you might work around these issues if you are not using MapR.

Use of Existing Non-Hadoop Applications

One of the key distinctions with MapR is that it has a realtime, fully read-write filesystem. This means that you not only can interact with data stored on the cluster via Hadoop commands and applications, but you also can access data via traditional routes. Any program in any language that ca access files on a Linux or Windows system can also access files in the MapR cluster using the same traditional mechanisms. This compatibility is made possible primarily because the MapR file system (MapR-FS) allows access to files via NFS. This is very different from HDFS, the file system that oth major Hadoop distributions use for distributed data storage.

What are the implications of MapR-FS being a read/write POSIX file system accessible via NFS or Hadoop commands? One effect is that existing applications can be used directly, without needing to rewrite them as Hadoop applications. In contrast, when using with other Hadoop distributions, the workflow generally includes steps in which data is copied out of the Hadoop file system to local files to be available for traditional tools and applications. The output of these applications is then copied back into the HDFS cluster. This doesn't mean the use cases described in this book are only for Mapl but some of the details of the workflow may be somewhat different with other Hadoop distributions. For instance the workflow would need to include time to write Hadoop-specific code to access data and run applications, and the team doing so would need to be well versed in how HDFS file APIs differ from more traditional file APIs.

In case you are curious about the reason for this difference in MapR's ability to use traditional code,

as well as some of MapR's other specific capabilities, here's a brief technical explanation. One key difference lies in the size of the units used to manipulate and track files in a MapR cluster as compared with HDFS. As illustrated in Figure 3-1, files in HDFS files are broken into blocks of a fixed size. The default value for these blocks is 128 megabytes. The value can be changed, but it still applies across all files in the cluster. The block size is fundamental to how HDFS tracks files in a central system called the name node. Every change in the size or other properties of the file aside fro the content of the file itself must be sent to the name node in order to keep track of all pieces of all th files in the cluster. HDFS blocks are the unit of allocation, and once written, they are never updated. This has several logical consequences, including:

1. Realtime use of HDFS is difficult or impossible. This happens because every write to a file extends the length of the file when the write is committed. That means every such change requires talking to the name node, so programs try to avoid committing writes too often. In fact it is common for multiple blocks to be flushed at once.

2. The consistency model of HDFS is such that readers cannot be allowed to read files while a writer still has them open for writing.

3. Out-of-order writes cannot be allowed because they would constitute updates to the file.

4. A name node has to keep track of all of the blocks of all of the files, and because this information can churn rapidly while programs are running, this information has to be kept in memory. This means the name node's memory size is proportional to the number of blocks that can be tracked. That number times the block size is the scalability limit of an HDFS file system

These consequences made HDFS much easier to implement originally, but they make it much harder for it to work in real time (which implies readers can see data instantly and writers can flush often), t scale to very large sizes (because of the limited number of blocks), or to support first-class access via NFS (because NFS inherently reorders writes and has no concept of an open file).