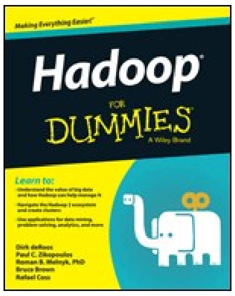

Hadoop for Dummies

by Dirk deRoos et al.

Chapter 11: Hadoop and the Data Warehouse—Friends or Foes?

In This Chapter

- Contrasting the architectural differences between Hadoop and relational databases
- Landing enterprise data in Hadoop
- Archiving data in Hadoop
- Preprocessing data in Hadoop
- Discovery and exploration in Hadoop

IT types like us tend to love the latest and greatest new technologies, and when compelling platforms like Hadoop emerge, they're often accompanied by a significant amount of hype. You might even say that this *For Dummies* book is part of that hype! When it comes to Hadoop, though, there's real substance behind the hype. Not convinced? Just look at the increasing numbers of code contributions in the Apache Hadoop projects as well as the adoption rates of Hadoop in medium-to large-size businesses. The consensus is overwhelming: Hadoop is here to stay.

It's important to understand how any new technology relates to existing technologies and business practices. In the case of Hadoop, you should know how it will impact the field of enterprise data management. In our experience, the IT market reacts in two distinct ways.

On one hand, the Hadoop hype machine is in full gear and bent on world domination. This camp sees Hadoop replacing the relational database products that now power the world's data warehouses. The argument here is compelling: Hadoop is cheap and scalable, and it has queryable interfaces that are becoming increasingly faster and more closely compliant with ANSI SQL — *the* standard for programming applications used with database systems.

On the other hand, many relational warehouse vendors have gone out of their way to resist the appeal of all the Hadoop hype. Understandably, they won't roll over and make way for Hadoop to replace their relational database offerings. They've adopted what we consider to be a protectionist stance, drawing a line between structured data, which they consider to be the exclusive domain of relational databases, and unstructured data, which is where they feel Hadoop can operate. In this model, they're positioning Hadoop as solely a tool to transform unstructured data into a structured form for relational databases to store.

We feel that the truth lies in the middle of these opposing views: there are many workloads and business applications where data warehouses powered by relational databases are still the most practical choice. At the same time, there are classes of data (both structured and unstructured) and workloads where Hadoop is the most practical option. The key consideration here is using tools that are best suited for the task at hand.

The focus of this chapter is on comparing and contrasting the relative strengths of Hadoop technologies and relational databases and then on exploring a family of use cases for how Hadoop's strengths can expand the capabilities of today's data warehouses.

Comparing and Contrasting Hadoop with Relational Databases

Database models and database systems have been around as long as computer systems have roamed the earth, and most of us IT people have at least been exposed to (or perhaps even used) some type of database technology for a very long time. The most prevalent database technology is the relational database management system (RDBMS), which can be traced back to Edgar F. Codd's groundbreaking work at IBM in the 1970s. Several well-known companies (IBM, Informix, Oracle, and Sybase, for example) capitalized on Codd's work and sold, or continue to sell, products based on his relational model. At roughly the same time, Donald D. Chamberlin and Raymond F. Boyce created the structured query language (SQL) as a way to provide a common programming language for managing data stored in an RDBMS.

The 1980s and 1990s saw the birth of the object database, which provided a better fit for a particular class of problems than the relational database, and now *another* new class of technologies, commonly referred to as NoSQL databases, is emerging. Because NoSQL databases play a significant role in the Hadoop story, they deserve a closer look, so be sure to read the next section.

NoSQL Data Stores

NoSQL data stores originally subscribed to the notion "Just Say No to SQL" (to paraphrase from an anti-drug advertising campaign in the 1980s), and they were a reaction to the perceived limitations of (SQL-based) relational databases. It's not that these folks hated SQL, but they were tired of forcing square pegs into round holes by solving problems that relational databases weren't designed for. A relational database is a powerful tool, but for some kinds of data (like key-value pairs, or graphs) and some usage patterns (like extremely large scale storage) a relational database just isn't practical. And when it comes to high-volume storage, relational database can be expensive, both in terms of database license costs and hardware costs. (Relational databases are designed to work with enterprise-grade hardware.) So, with the NoSQL movement, creative programmers developed dozens of solutions for different kinds of thorny data storage and processing problems. These NoSQL databases typically provide massive scalability by way of clustering, and are often designed to enable high throughput and low latency.

REMEMBER The name NoSQL is somewhat misleading because many databases that fit the category *do* have SQL support (rather than "NoSQL" support). Think of its name instead as "Not Only SQL."

The NoSQL offerings available today can be broken down into four distinct categories, based on their design and purpose:

- **Key-value stores**: This offering provides a way to store any kind of data without having to use a schema. This is in contrast to relational databases, where you need to define the schema (the table structure) before any data is inserted. Since key-value stores don't require a schema, you have great flexibility to store data in many formats. In a key-value store, a row simply consists of a key (an identifier) and a value, which can be anything from an integer value to a large binary data string. Many implementations of key-value stores are based on Amazon's Dynamo paper.
- **Column family stores**: Here you have databases in which columns are grouped into column families and stored together on disk.

  TECHNICAL STUFF Strictly speaking, many of these databases aren't column-oriented, because they're based on Google's BigTable paper, which stores data as a multidimensional sorted map. (For more on the role of Google's BigTable paper on database design, see Chapter 12.)
- **Document stores**: This offering relies on collections of similarly encoded and formatted documents to improve efficiencies. Document stores enable individual documents in a collection to include only a subset of fields, so only the data that's needed is stored. For sparse data sets, where many fields are often not populated, this can translate into significant space savings. By contrast, empty columns in relational database tables do take up space. Document stores also enables schema flexibility, because only the fields that are needed are stored, and new fields can be added. Again, in contrast to relational databases, table structures are defined up front before data is stored, and changing columns is a tedious task that impacts the entire data set.
- **Graph databases**: Here you have databases that store *graph structures* — representations that show collections of entities (vertices or nodes) and their relationships (edges) with each other. These structures enable graph databases to be extremely well suited for storing complex structures, like the linking relationships between all known web pages. (For example, individual web pages are nodes, and the edges connecting them are links from one page to another.) Google, of course, is all over graph technology, and invented a graph processing engine called Pregel to power its PageRank algorithm. (And yes, there's a white paper on Pregel.) In the Hadoop community, there's an Apache project called Giraph (based on the Pregel paper), which is a graph processing engine designed to process graphs stored in HDFS.

REMEMBER The data storage and processing options available in Hadoop are in many cases implementations of the NoSQL categories listed here. This will help you better evaluate solutions that are available to you and see how Hadoop can complement traditional data warehouses.

ACID versus BASE Data Stores

One hallmark of relational database systems is something known as *ACID compliance*. As you might have guessed, ACID is an acronym — the individual letters, meant to describe a characteristic of individual database transactions, can be expanded as described in this list:

- **Atomicity**: The database transaction must completely succeed or completely fail. Partial success is not allowed.
- **Consistency**: During the database transaction, the RDBMS progresses from one valid state to another. The state is never invalid.
- **Isolation**: The client's database transaction must occur in isolation from other clients attempting to transact with the RDBMS.
- **Durability**: The data operation that was part of the transaction must be reflected in *nonvolatile storage* (computer memory that can retrieve stored information even when not powered – like a hard disk) and persist after the transaction successfully completes. Transaction failures cannot leave the data in a partially committed state.

Certain use cases for RDBMSs, like online transaction processing, depend on ACID-compliant transactions between the client and the RDBMS for the system to function properly. A great example of an ACID-compliant transaction is a transfer of funds from one bank account to another. This breaks down into two database transactions, where the originating account shows a withdrawal, and the destination account shows a deposit. Obviously, these two transactions have to be tied together in order to be valid so that if either of them fail, the whole operation must fail to ensure both balances remain valid.

Hadoop itself has no concept of transactions (or even records, for that matter), so it clearly isn't an ACID-compliant system. Thinking more specifically about data storage and processing projects in the entire Hadoop ecosystem (we tell you more about these projects later in this chapter), none of them is fully ACID-compliant, either. However, they *do* reflect properties that you often see in NoSQL data stores, so there is some precedent to the Hadoop approach.

One key concept behind NoSQL data stores is that not every application truly needs ACID-compliant transactions. Relaxing on certain ACID properties (and moving away from the relational model) has opened up a wealth of possibilities, which have enabled some NoSQL data stores to achieve massive scalability and performance for their niche applications. Whereas ACID defines the key characteristics required for reliable transaction processing, the NoSQL world requires different characteristics to enable flexibility and scalability. These opposing characteristics are cleverly captured in the acronym BASE:

- ***Basically Available***: The system is guaranteed to be available for querying by all users. (No isolation here.)
- ***Soft State***: The values stored in the system may change because of the eventual consistency model, as described in the next bullet.
- ***Eventually Consistent***: As data is added to the system, the system's state is gradually replicated across all nodes. For example, in Hadoop, when a file is written to the HDFS, the replicas of the data blocks are created in different data nodes after the original data blocks have been written. For the short period before the blocks are replicated, the state of the file system isn't consistent.

The acronym BASE is a bit contrived, as most NoSQL data stores don't completely abandon *all* the ACID characteristics — it's not really the polar opposite concept that the name implies, in other words. Also, the Soft State and Eventually Consistent characteristics amount to the same thing, but the point is that by relaxing consistency, the system can horizontally scale (many nodes) and ensure availability.

TECHNICAL STUFF No discussion of NoSQL would be complete without mentioning the CAP theorem, which represents the three kinds of guarantees that architects aim to provide in their systems:

- **Consistency**: Similar to the C in ACID, all nodes in the system would have the same view of the data at any time.
- **Availability**: The system always responds to requests.
- **Partition tolerance**: The system remains online if network problems occur between system nodes.

The CAP theorem states that in distributed networked systems, architects have to choose two of these three guarantees — you can't promise your users all three. That leaves you with the three possibilities shown in Figure 11-1:

- **Systems using traditional relational technologies** normally aren't partition tolerant, so they can guarantee consistency and availability. In short, if one part of these traditional relational technologies systems is offline, the whole system is offline.
- **Systems where partition tolerance and availability are of primary importance** can't guarantee consistency, because updates (that destroyer of consistency) can be made on either side of the partition. The key-value stores Dynamo and CouchDB and the column-family store Cassandra are popular examples of partition tolerant/availability (PA) systems.
- **Systems where partition tolerance and consistency are of primary importance** can't guarantee availability because the systems return errors until the partitioned state is resolved.

  REMEMBER Hadoop-based data stores are considered CP systems (*c*onsistent and *p*artition tolerant). With data stored redundantly across many slave nodes, outages to large portions (partitions) of a Hadoop cluster can be tolerated. Hadoop is considered to be consistent because it has a central metadata store (the NameNode) which maintains a single, consistent view of data stored in the cluster. We can't say that Hadoop guarantees availability, because if the NameNode fails applications cannot access data in the cluster.

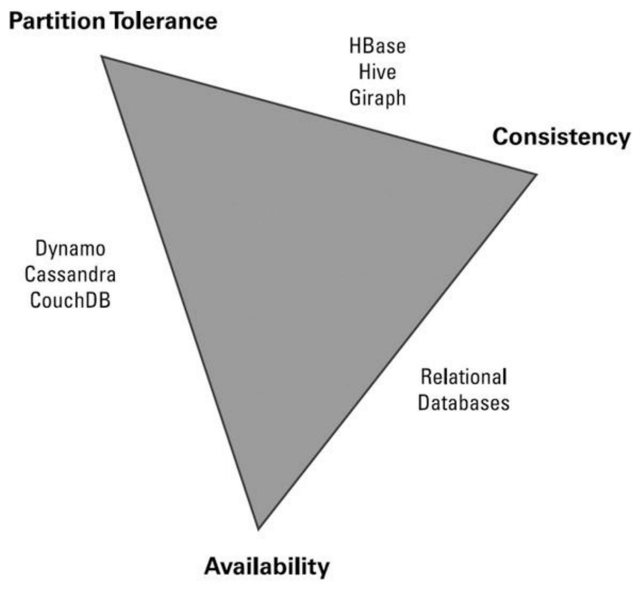

Figure 11-1: CAP theorem guarantees and implementation examples

Structured Data Storage and Processing in Hadoop

When considering Hadoop's capabilities for working with structured data (or working with data of any type, for that matter), remember Hadoop's core characteristics: Hadoop is, first and foremost, a general-purpose data storage and processing platform designed to scale out to thousands of compute nodes and petabytes of data. There's no data model in Hadoop itself; data is simply stored on the Hadoop cluster as raw files. As such, the core components of Hadoop itself have no special capabilities for cataloging, indexing, or querying structured data.

The beauty of a general-purpose data storage system is that it can be extended for highly specific purposes. The Hadoop community has done just that with a number of Apache projects — projects that, in totality, make up the Hadoop *ecosystem*. When it comes to structured data storage and processing, the projects described in this list are the most commonly used:

- **Hive**: A data warehousing framework for Hadoop. Hive catalogs data in structured files and provides a query interface with the SQL-like language named HiveQL. (We tell you tons more about Hive in Chapter 13.)

- **HBase**: A *distributed* database — a NoSQL database that relies on multiple computers rather than on a single CPU, in other words — that's built on top of Hadoop. (For more on HBase, see Chapter 12.)
- **Giraph**: A graph processing engine for data stored in Hadoop. (See the earlier discussion in this chapter on NoSQL and graph databases.)

REMEMBER Many other Apache projects support different aspects of structured data analysis, and some projects focus on a number of frameworks and interfaces. Chapter 14 takes a look at another structured data analysis tool — the aptly named Sqoop — and Chapter 15 takes a look at SQL interfaces to Hadoop data.

When determining the optimal architecture for your analytics needs, be sure to evaluate the attributes and capabilities of the systems you're considering. Table 11-1 compares Hadoop-based data stores (Hive, Giraph, and HBase) with traditional RDBMS.

Table 11-1: A Comparison of Hadoop-Based Storage and RDBMS

Criteria	*Hive*	*Giraph*	*HBase*	*RDBMS*
Changeable data	No		Yes	Yes
Data layout	Raw files stored in HDFS; Hive supports proprietary row-oriented or column-oriented formats.		A sparse, distributed, persistent multidimensional sorted map	Row-oriented or column-oriented
Data types	Bytes; data types are interpreted on query.			Rich data type support
Hardware	Hadoop-clustered commodity x86 servers; five or more is typical because the underlying storage technology is HDFS, which by default requires three replicas.			Typically large, scalable multiprocessor systems
High availability	Yes; built into the Hadoop architecture			Yes, if the hardware and RDBMS are configured correctly
Indexes	Yes	No	Row-key only or special table required	Yes
Query language	HiveQL	Giraph API	HBase API commands (`get`, `put`, `scan`, `delete`, `increment`, `check`), HiveQL	SQL
Schema	Schema defined as files are catalogued with the Hive Data Definition Language (DDL)	Schema on read	Variability in schema between rows	Schema on load
Throughput	Millions of reads and writes per second			Thousands of reads and writes per second
Transactions	None		Provides ACID support on only a single row	Provides multi-row and cross-table transactional support with full ACID property compliance
Transaction speed	Modest speed for interactive queries; fast for full table scans		Fast for interactive queries; fast for full table scans	Fast for interactive queries; slower for full table scans
Typical size	Ranges from terabytes to petabytes (from hundreds of millions to billions of rows)			From gigabytes to terabytes (from hundreds of thousands to millions of rows)

Modernizing the Warehouse with Hadoop

We want to stress the fact that Hadoop and traditional RDBMS technologies are more complementary than competitive. The sensationalist marketing and news media articles that pit these technologies against each other are missing the point: By using the strengths of these technologies together, you can build a highly flexible and scalable analytics environment.

Rather than have you simply trust us on that assertion, we use the rest of this chapter to lay out four (specific) ways that Hadoop can modernize the warehouse. Get ready to delve into the messy details of these use cases:

- Landing Zone for All Data
- Queryable Archive of Cold Data
- Preprocessing Engine
- Data Discovery Zone

The Landing Zone

When we try to puzzle out what an analytics environment might look like in the future, we stumble across the pattern of the Hadoop-based landing zone time and time again. In fact, it's no longer even a futures-oriented discussion because the landing zone has become *the* way that forward-looking companies now try to save IT costs, and provide a platform for innovative data analysis.

So what exactly is the landing zone? At the most basic level, the *landing zone* is merely the central place where data will land in your enterprise — weekly extractions of data from operational databases, for example, or from systems generating log files. Hadoop is a useful repository in which to land data, for these reasons:

- It can handle all kinds of data.
- It's easily scalable.
- It's inexpensive.
- Once you land data in Hadoop, you have the flexibility to query, analyze, or process the data in a variety of ways.

A Hadoop-based landing zone, seen in Figure 11-2, is the foundation of the other three use cases we describe later in this chapter.

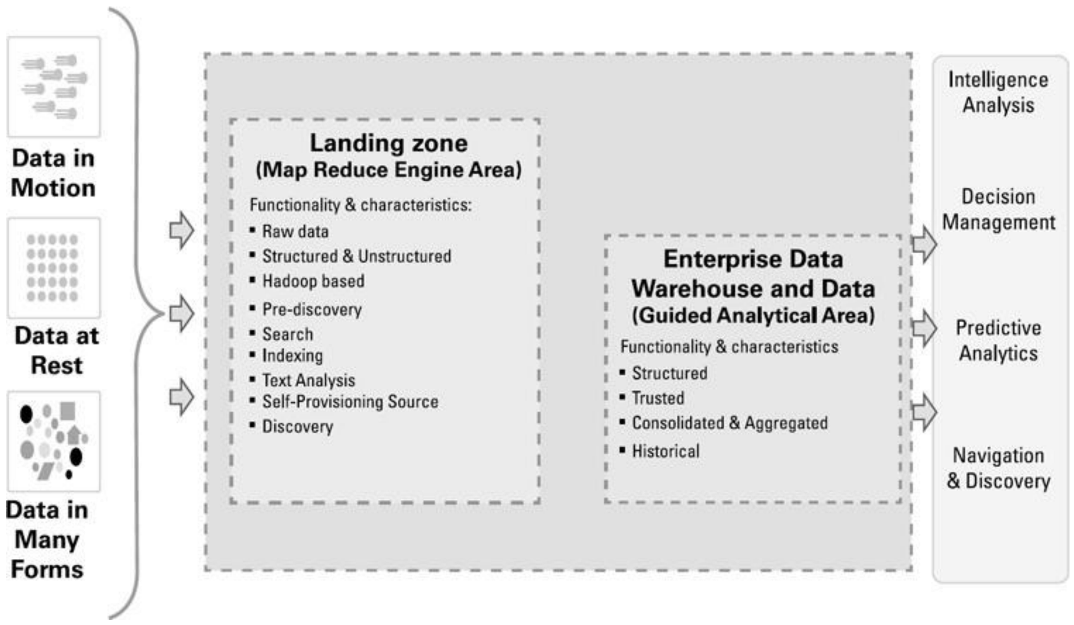

Figure 11-2: The enterprise doorstep: Hadoop serves as a landing zone for incoming data

REMEMBER This diagram only shows part of the story and is by no means complete. After all, you need to know how the data moves from the landing zone to the data warehouse, and so on. (We get around to answering such questions and filling in some of these blanks as we add more Hadoop use cases in this chapter.)

The starting point for the discussion on modernizing a data warehouse has to be how organizations use data warehouses and the challenges IT departments face with them. In the 1980s, once organizations became good at storing their operational information in relational databases (sales transactions, for example, or supply chain statuses), business leaders began to want reports generated from this relational data. The earliest relational stores were operational databases and were designed for Online Transaction Processing (OLTP), so that records could be inserted, updated, or deleted as quickly as possible. This is an impractical architecture for large scale reporting and analysis, so Relational Online Analytical Processing (ROLAP) databases were developed to meet this need. This led to the evolution of a whole new kind of RDBMS: a *data warehouse*, which is a separate entity and lives alongside an organization's operational data stores. This comes down to using purpose-built tools for greater efficiency: we have operational data stores, which are designed to efficiently process transactions, and data warehouses, which are designed to support repeated analysis and reporting.

Data warehouses are under increasing stress though, for the following reasons:

- Increased demand to keep longer periods of data online.
- Increased demand for processing resources to transform data for use in other warehouses and data marts.
- Increased demand for innovative analytics, which requires analysts to pose questions on the warehouse data, on top of the regular reporting that's already being done. This can incur significant additional processing.

The use cases we cover later in this chapter address these pain points, and actually frees data warehouses to do what they're designed to do, which is support the regular reporting activities that keep organizations running.

In Figure 11-2, we can see the data warehouse presented as the primary resource for the various kinds of analysis listed on the far right side of the figure. Here we also see the concept of a landing zone represented, where Hadoop will store data from a variety of incoming data sources. To enable a Hadoop landing zone, you'll need to ensure you can write data from the various data sources to HDFS. For relational databases, a good solution would be to use Sqoop, which we talk about in Chapter 14.

But landing the data is only the beginning. What you do with it is where the real value comes in, and that's what we'll get into with the remaining three use cases — all of which depend on a Hadoop-based landing zone populated with data from a variety of sources.

TIP When you're moving data from many sources into your landing zone, one issue that you'll inevitably run into is data quality. It's common for companies to have many operational databases where key details are different, for example, that a customer might be known as "D. deRoos" in one database, and "Dirk deRoos" in another. Another quality problem lies in systems where there's a heavy reliance on manual data entry, either from customers or staff — here, it's not uncommon to find first names and last names switched around or other misinformation in the data fields. Data quality issues are a big deal for data warehouse environments, and that's why a lot of effort goes into cleansing and validation steps as data from other systems are processed as it's loaded into the warehouse. It all comes down to *trust*: if the data you're asking questions against is dirty, you can't trust the answers in your reports. So while there's huge potential in having access to many different data sets from different sources in your Hadoop landing zone, you have to factor in data quality and how much you can trust the data.

A Queryable Archive of Cold Warehouse Data

A multitude of studies show that most data in an enterprise data warehouse is rarely queried. Database vendors have responded to such observations by implementing their own methods for sorting out what data gets placed where. One method orders the data universe into designations of hot, warm, or cold, where *hot* data (sometimes called *active* data) is used often, *warm* data is used from time to time; and *cold* data is rarely used. The proposed solution for many vendors is to store the cold data on slower disks within the data warehouse enclosures or to create clever caching strategies to keep the hot data in-memory, among others. The problem with this approach is that even though slower storage is used, it's still expensive to store cold, seldom used data in a warehouse. The costs here stems from both hardware and software licensing. At the same time, cold and dormant data is often archived to tape. This traditional model of archiving data breaks down when you want to query all cold data in a cost-effective and relatively efficient way — without having to request old tapes, in other words.

If you look at the cost and operational characteristics of Hadoop, indeed it seems that it's set to become the new backup tape. Hadoop is inexpensive largely because Hadoop systems are designed to use a lower grade of hardware than what's normally deployed in data warehouse systems. Another significant cost savings is software licensing. Commercial Hadoop distribution licenses require a fraction of the cost of relational data warehouse software licenses, which are notorious for being expensive. From an operational perspective, Hadoop is designed to easily scale just by adding additional slave nodes to an existing cluster. And as slave nodes are added and data sets grow in volume, Hadoop's data processing frameworks enable your applications to seamlessly handle the increased workload. Hadoop represents a simple, flexible, and inexpensive way to push processing across literally thousands of servers. To put this statement into perspective: In 1955, 1 megabyte of storage cost about US$6,235. By the middle of 1993, the price per megabyte dipped below US$1. The cost to purchase 1 megabyte of storage is now US$0.0000467 — in other words, at the time this book was published, US$1 could get you about 22 gigabytes of storage.

With its scalable and inexpensive architecture, Hadoop would seem to be a perfect choice for archiving warehouse data … except for one small matter: Most of the IT world runs on SQL, and SQL on its own doesn't play well with Hadoop. Sure, the more Hadoop-friendly NoSQL movement is alive and well, but most power users now use SQL by way of common, off-the-shelf toolsets that generate SQL queries under the hood — products such as Tableau, Microsoft Excel, and IBM Cognos BI. It's true that the Hadoop ecosystem includes Hive, but Hive supports only a subset of SQL, and although performance is improving (along with SQL support), it's not nearly as fast at answering smaller queries as relational systems are. Recently, there has been major progress around SQL access to Hadoop, which has paved the way for Hadoop to become the new destination for online data warehouse archives.

Depending on the Hadoop vendor, SQL (or SQL-like) APIs are becoming available so that the more common off-the-shelf reporting and analytics tools can seamlessly issue SQL that executes on data stored in Hadoop. For example, IBM has its Big SQL API, Cloudera has Impala, and Hive itself, via the Hortonworks Stinger initiative, is becoming increasingly SQL compliant. Though various points of view exist (some aim to enhance Hive; some, to extend Hive; and others, to provide an alternative), all these solutions attempt to tackle two issues: MapReduce is a poor solution for executing smaller queries, and SQL access is — for now — the key to enabling IT workers to use their existing SQL skills to get value out of data stored in Hadoop.

To add it all up — the inexpensive cost of storage for Hadoop plus the ability to query Hadoop data with SQL — we think that Hadoop is the prime destination for archival data. We consider this use case to have a low impact on your organization because you can start building your Hadoop skill set on data that's not stored on performance-mission-critical systems. What's more, you don't have to work hard to get at the data. (Since archived data is normally stored on systems that have low usage, it's easier to get at than data that's in "the limelight" on performance-mission-critical systems, like data warehouses.) If you're already using Hadoop as a landing zone, you have the foundation for your archive! You simply keep what you want to archive and delete what you don't.

If you think about the Landing Zone use case (refer to Figure 11-2), the queryable archive, shown in Figure 11-3, extends the value of Hadoop and starts to integrate pieces that likely already exist in your enterprise. It's a great example of finding economies of scale and cost take-out opportunities using Hadoop.

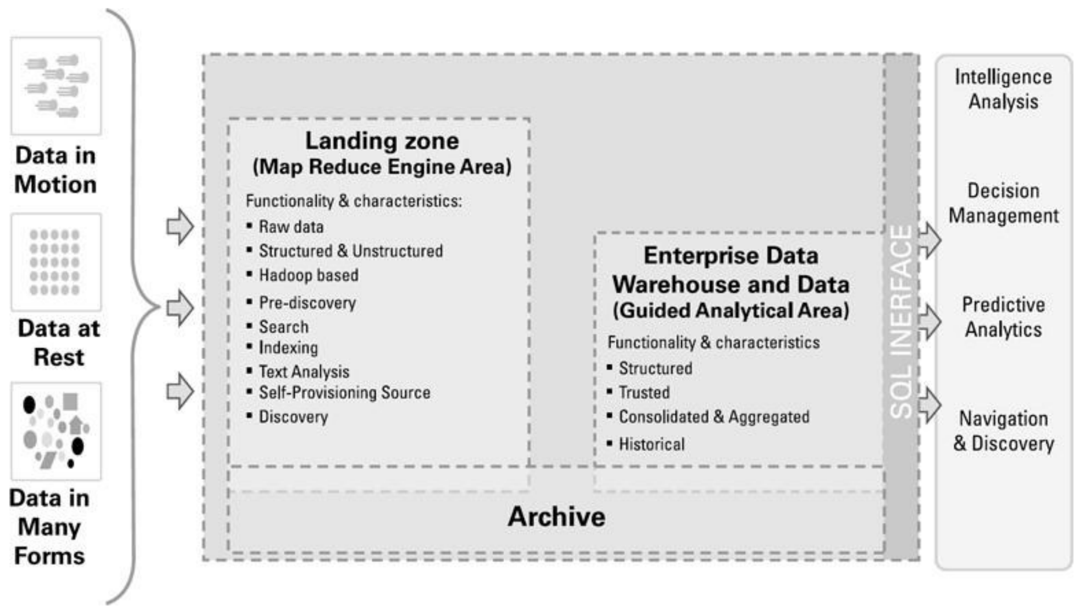

Figure 11-3: Hadoop as a queryable archive in support of an enterprise data warehouse

In Figure 11-3, we show the archive component connecting the landing zone and the data warehouse. The data being archived originates in the warehouse and is then stored in the Hadoop cluster, which is also provisioning the landing zone. In short, you can use the same Hadoop cluster to archive data and act as your landing zone.

The key Hadoop technology you would use to perform the archiving is Sqoop, which can move the data to be archived from the data warehouse into Hadoop. You will need to consider what form you want the data to take in your Hadoop cluster. In general, compressed Hive files are a good choice. You can, of course, transform the data from the warehouse structures into some other form (for example, a normalized form to reduce redundancy), but this is generally not a good idea. Keeping the data in the same structure as what's in the warehouse will make it much easier to perform a full data set query across the archived data in Hadoop and the active data that's in the warehouse.

The concept of querying both the active and archived data sets brings up another consideration: how much data should you archive? There are really two common choices: archive everything as data is added and changed in the data warehouse, or only archive the data you deem to be cold. Archiving everything has the benefit of enabling you to easily issue queries from one single interface across the entire data set — without a full archive, you'll need to figure out a federated query solution where you would have to union the results from the archive and the active data warehouse. But the downside here is that regular updates of your data warehouse's hot data would cause headaches for the Hadoop-based archive. This is because any changes to data in individual rows and columns would require wholesale deletion and re-cataloging of existing data sets.

Now that archival data is stored in your Hadoop-based landing zone (assuming you're using an option like the compressed Hive files mentioned above), you can query it. This is where the SQL on Hadoop solutions we talk about in Chapter 15 can become interesting. An excellent example of what's possible is for the analysis tools we see on the right in Figure 11-3 to directly run reports or analysis on the archived data stored in Hadoop. This is not to replace the data warehouse — after all, Hadoop would not be able to match the warehouse's performance characteristics for supporting hundreds or more concurrent users asking complex questions. The point here is that you can use reporting tools against Hadoop to experiment and come up with new questions to answer in a dedicated warehouse or mart.

TIP When you start your first Hadoop-based project for archiving warehouse data, don't break the current processes until you've fully tested them on your new Hadoop solution. In other words, if your current warehousing strategy is to archive to tape, keep that process in place, and dual-archive the data into Hadoop and tape until you've fully tested the scenario (which would typically include restoring the warehouse data in case of a warehouse failure). Though you're maintaining (in the short term) two archive repositories, you'll have a robust infrastructure in place and tested before you decommission a tried-and-true process. Personal observation makes us believe that this process can ensure that you remain employed — with your current employer.

This use case is simple because there's no change to the existing warehouse. The business goal is still the same: cheaper storage and licensing costs by migrating rarely-used data to an archive. The difference in this case is that the technology behind the archive is Hadoop rather than offline storage, like tape. In addition, we've seen various archive vendors start to incorporate Hadoop into their solutions (for example, allowing their proprietary archive files to reside on HDFS), so expect capabilities in this area to expand soon.

As you develop Hadoop skills (like exchanging data between Hadoop and relational databases and querying data in HDFS) you can use them to tackle bigger problems, such as analysis projects, which could provide additional value for your organization's Hadoop investment. This will be especially relevant in the data discovery sandbox use case we describe a bit later.

Hadoop as a Data Preprocessing Engine

One of the earliest use cases for Hadoop in the enterprise was as a programmatic transformation engine used to preprocess data bound for a data warehouse. Essentially, this use case leverages the power of the Hadoop ecosystem to manipulate and apply transformations to data *before* it's loaded into a data warehouse. Though the actual transformation engine is new (it's Hadoop, so transformations and data flows are coded in Pig or MapReduce, among other languages), the approach itself has been in use awhile. What we're talking about here is Extract, Transform, Load (ETL) processes.

Think back for a minute to our description of the evolution of OLTP and ROLAP databases in the Landing Zone section earlier in the chapter. The outcome of this is that many organizations with operational databases also deployed data warehouses. So how do IT departments get data from their operational databases into their data warehouses? (Remember that the operational data is typically not in a form that lends itself to analysis.) The answer here is ETL, and as data warehouses increased in use and importance, the steps in the process became well understood and best practices were developed. Also, a number of software companies started offering interesting ETL solutions so that IT departments could minimize their own custom code development.

The basic ETL process is fairly straightforward: you *E*xtract data from an operational database, *T*ransform it into the form you need for your analysis and reporting tools, and then you *L*oad this data into your data warehouse.

One common variation to ETL is ELT — Extract, Load, and Transform. In the ELT process, you perform transformations (in contrast to ETL) *after* loading the data into the target repository. This approach is often used when the transformation stands to greatly benefit from a very fast SQL processing engine on structured data. (Relational databases may not excel at processing unstructured data, but they perform very fast processing of — guess what? — structured data.) If the data you're transforming is destined for a data warehouse, and many of those transformations can be done in SQL, you may choose to run the transformations in the data warehouse itself. ELT is especially appealing if the bulk of your skill set lies with SQL-based tooling. With Hadoop now able to process SQL queries, both ETL and ELT workloads can be hosted on Hadoop. In Figure 11-4 we show ETL services added to our reference architecture.

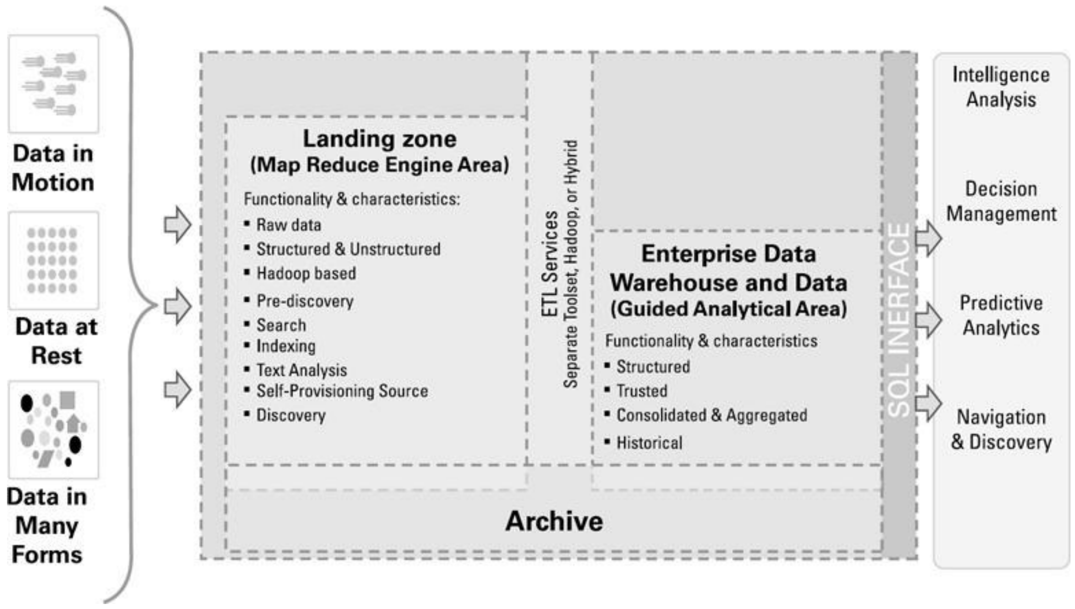

Figure 11-4: Hadoop can be used as a data transformation engine

If you've deployed a Hadoop-based landing zone, which you can again see in Figure 11-4, you've got almost everything you need in place to use Hadoop as a transformation engine. You're already landing data from your operational systems into Hadoop using Sqoop, which covers the extraction step. At this point you'll need to implement your transformation logic into MapReduce or Pig applications. After the data is transformed, you can load the data into the data warehouse using Sqoop.

Thinking back to the archive use case we just discussed, using Hadoop as a data transformation engine raises possibilities there as well. What we described initially was a scenario where the archive consists of warehouse data that's dumped into the landing zone. But if your data warehouse doesn't modify its data (it's for reporting only), you can simply keep the data you generate with the transformation process. In this model, data only flows from left-to-right in Figure 11-4, where data is extracted from operational databases, transformed in the landing zone, and then loaded into the data warehouse. With all the transformed data already in the landing zone, there's no need to copy it back to Hadoop — unless, of course, the data gets modified in the warehouse.

The Hybrid Data Preprocess Option (Or, Hybrids Aren't Just for Cars)

In addition to having to store larger volumes of cold data, one pressure we see in traditional data warehouses is that increasing amounts of processing resources are being used for transformation (ELT) workloads. The idea behind using Hadoop as a preprocessing engine to handle data transformation means that precious processing cycles are freed up, allowing the data warehouse to adhere to its original purpose: Answer repeated business questions to support analytic applications. Again, we're seeing how Hadoop can complement traditional data warehouse deployments and enhance their productivity.

Perhaps a tiny, imaginary light bulb has lit up over your head and you're thinking, "Hey, maybe there *are* some transformation tasks perfectly suited for Hadoop's data processing ability, but I know there's also a lot of transformation work steeped in algebraic, step-by-step tasks where running SQL on a relational database engine would be the better choice. Wouldn't it be cool if I could run SQL on Hadoop?" As we've been hinting, SQL on Hadoop is already here, and you can see the various offerings in Chapter 15. With the ability to issue SQL queries against data in Hadoop, you're not stuck with only an ETL approach to your data flows — you can also deploy ELT-like applications.

Another hybrid approach to consider is where to run your transformation logic: in Hadoop or in the data warehouse? Although some organizations are concerned about running anything but analytics in their warehouses, the fact remains that relational databases are excellent at running SQL, and could be a more practical place to run a transformation than Hadoop.

Data transformation is more than just data transformation

The idea of Hadoop-inspired ETL engines has gained a lot of traction in recent years. After all, Hadoop is a flexible data storage and processing platform that can support huge amounts of data and operations on that data. At the same time, it's fault tolerant, and it offers the opportunity for capital and software cost reductions.

Despite Hadoop's popularity as an ETL engine, however, many folks (including a famous firm of analysts) don't recommend Hadoop as the sole piece of technology for your ETL strategy. This is largely because developing ETL flows requires a great deal of expertise about your organization's existing database systems, the nature of the data itself, and the reports and applications dependent on it. In other words, the DBAs, developers, and architects in your IT department would need to become familiar enough with Hadoop to implement the needed ETL flows. For example, a lot of intensive hand coding with Pig, Hive, or even MapReduce may be necessary to create even the simplest of data flows — which puts your company on the hook for those skills if it follows this path. You have to code elements such as parallel debugging, application management services (such as check pointing and error and event handling). Also, consider enterprise requirements such as glossarization and being able to show your data's lineage. There are regulatory requirements for many industry standard reports, where data lineage is needed; the reporting organization must be able to show where the data points in the report come from, how the data got to you, and what has been done to the data.

Even for relational database systems, ETL is complex enough that there are popular specialized products that provide interfaces for managing and developing ETL flows. Some of these products now aid in Hadoop-based ETL and other Hadoop-based development. However, depending on your requirements, you may need to write some of your own code to support your transformation logic.

Data Discovery and Sandboxes

Data discovery is becoming an increasingly important activity for organizations that rely on their data to be a differentiator. Today, that describes most businesses, as the ability to see trends and extract meaning from available data sets applies to almost any industry. What this requires is two critical components: analysts with the creativity to think of novel ways of analyzing data sets to ask new questions (often these kinds of analysts are called *data scientists*); and to provide these analysts with access to as much data as possible.

Consider the traditional approach to analytics in today's IT landscape: The business user community now typically determines the business questions to ask — they submit a request, and the IT team builds a system that answers specific questions. From a technical perspective, because this work has traditionally been done in a relational database, it has been the IT team's responsibility to build schemas, remove data duplication, and so on. They're investing a lot of time into making this data queryable and to quickly answering preplanned questions that the business unit wants answered. This is why relational databases are typically considered schema-on-write because you have to do a lot of work in order to write to the database. (In many cases, the amount of work is worth the investment; however, in a world of big data, the value and quality of many newer types of data you work with is unknown.)

This relational database approach is well suited to many common business processes, such as monitoring sales by geography, product, or channel; extracting insight from customer surveys, cost and profitability analyses, and more — basically, the questions are asked time and time again. Data is typically highly structured and is most likely highly trusted in this environment (see the paragraph on trusted data in the earlier section describing the landing zone for more on the concept of *trust*) in this environment; we refer to this activity as *guided analytics* (as shown in Figure 11-5 and as you may have noticed in the use cases described earlier in this chapter).

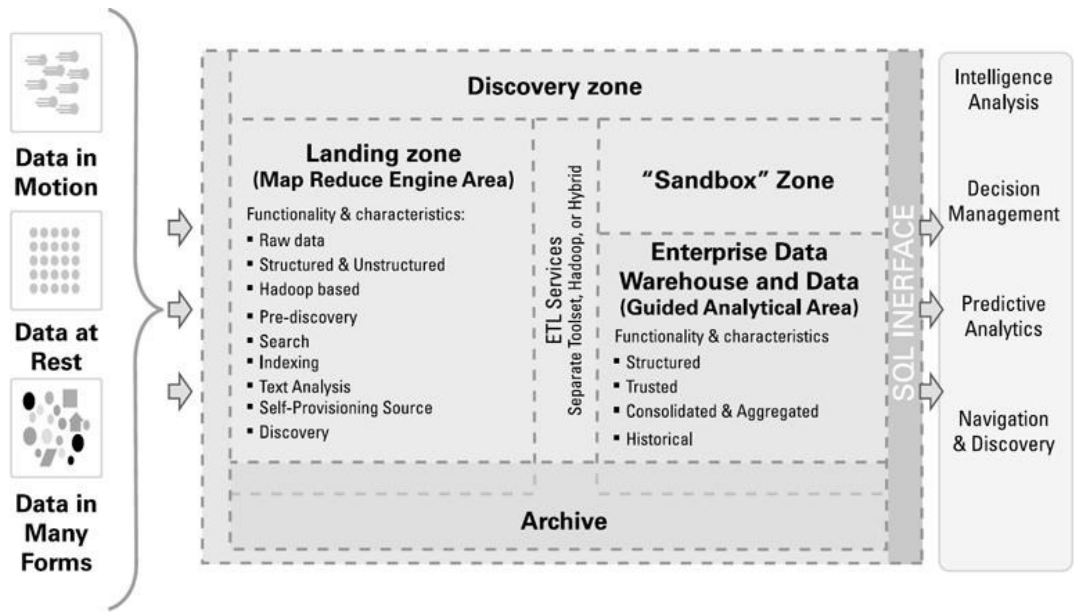

Figure 11-5: Using Hadoop to add Discovery and Sandbox capabilities to a modern-day analytics ecosystem

As an analogy, it's as though your 8-year-old child is taking a break for recess at school. For the most part, she can do whatever she wants within the school's grounds — as long as she remains within the fenced perimeter; however, she can't jump the fence to discover what's on the outside. Specifically, your child can explore a known, safeguarded (within the schema) area and analyze whatever can be found within that area.

Now imagine that your analytics environment has a discovery zone, as shown in Figure 11-5. In this scenario, IT delivers data (it's likely not to be fully trusted, and it's likely "dirty") on a flexible discovery platform for business users to ask virtually any question they want. In our analogy, your child is allowed to climb the schoolyard fence (this area is schema-less), venture into the forest, and return with whatever items she discovers. (Of course, in the IT world, you don't have to worry about business users getting lost or getting poison ivy.)

If you think about it, data discovery mirrors in some respects the evolution of gold mining. During the gold rush years of old, gold strikes would spark resource investment because someone discovered gold — it was visible to the naked eye, it had clear value, and it therefore warranted the investment. Fifty years ago, no one could afford to mine low-grade ore for gold because cost-effective or capable technology didn't exist (equipment to move and handle vast amounts of ore wasn't available) and rich-grade ore was still available (compared to today, gold was relatively easier to find). Quite simply, it wasn't cost effective (or even possible) to work through the noise (low-grade ore) to find the signals (the gold). With Hadoop, IT shops now have the capital equipment to process millions of tons of ore (data with a low value per byte) to find gold that's nearly invisible to the naked eye (data with high value per byte). And that's exactly what discovery is all about. It's about having a low-cost, flexible repository where next-to-zero investment is made to enrich the data until a discovery is made. After a discovery is made, it might make sense to ask for more resources (to mine the gold discovery) and formalize it into an analytics process that can be deployed in a data warehouse or specialized data mart.

When insights are made in the discovery zone, that's likely a good time to engage the IT department and formalize a process, or have those folks lend assistance to more in-depth discovery. In fact, this new pattern could even move into the area of guided analytics. The point is that IT provisioned the discovery zone for business users to ask and invent questions they haven't thought about before. Because that zone resides in Hadoop, it's agile and allows for users to venture into the wild blue yonder.

Notice that Figure 11-5 has a sandbox zone. In some reference architectures, this zone is combined with the discovery zone. We like to keep these zones separate because we see this area being used by application developers and IT shops to do their own research, test applications, and, perhaps, formalize conclusions and findings in the Discovery Zone when IT assistance is required after a potential discovery is made.

We'd be remiss not to note that our reference architecture is flexible, and can easily be tweaked. Nothing is cast in stone: you can take what you need, leave what you don't, and add your own nuances. For instance, some organizations may choose to co-locate all zones into a single Hadoop cluster, some may choose to leverage a single cluster designed for multiple purposes; and others may physically separate them. None of this affects the use cases that we've built into the final reference architecture shown in Figure 11-5.

Looking to the future

The relational database, as we know it, isn't going away any time soon. Pundits will always claim, "RDBMS will go the way of the dinosaur," but we think (at least for now) that IT needs both systems. More importantly, IT needs both systems to work together and complement each other. Suppose that you need to derive client attributes from social media feeds. Assume that your company underwrites a life insurance policy to an individual with a family. Your processes likely run the gamut of medical tests and smoker / nonsmoker classifications, but your actuaries might be better able to assess risk and costs if they know that this particular client participates in extreme sports such as hang gliding. If you could extract this information from social media data that you've stored in a Hadoop landing zone, you could analyze this information and create a risk multiplier based on social activities that your client openly shares with the world via Facebook and Twitter, for example. This information could be updated in your system of record, where the actual policy costs are itemized and maintained. This example explains systems of engagement meeting systems of record, which is a key tenet to a next-generation analytics ecosystem.