Chapter 4

Digging into Big Data Technology Components

In This Chapter

- Introducing the big data stack
- Redundant physical infrastructure
- Security infrastructure
- Interfaces and feeds to and from applications
- Operational databases
- Organizing data services and tools
- Analytical data warehouses
- Introduction to big analytics
- Introduction to big data applications

As discussed in the first few chapters, big data is about high-volume and often high-velocity data streams with highly diverse data types. Many seasoned software architects and developers know how to address one or even two of these situations quite readily. For example, if you are faced with high-volume transactional data with fault tolerance requirements, you might choose to deploy redundant relational database clusters in a data center with a very fast network infrastructure. Similarly, if the requirements are to integrate different data types from many known and anonymous sources, the choice might be to construct an extensible meta-model driving a customized data warehouse.

However, you may not have had the luxury of creating specific deployments in a much more dynamic big data world. When you move out of the world where you own and tightly control your data, you need to create an architectural model for addressing this type of hybrid environment. This new environment requires an architecture that understands both the dynamic nature

of big data and the requirement to apply the knowledge to a business solution. In this chapter, we examine the architectural considerations associated with big data. We also dig a bit deeper into the big data technology stack we introduce in Chapter 1.

Exploring the Big Data Stack

Like any important data architecture, you should design a model that takes a holistic look at how all the elements need to come together. Although this will take some time in the beginning, it will save many hours of development and lots of frustration during the subsequent implementations. You need to think about big data as a strategy, not a project.

Good design principles are critical when creating (or evolving) an environment to support big data — whether dealing with storage, analytics, reporting, or applications. The environment must include considerations for hardware, infrastructure software, operational software, management software, well-defined application programming interfaces (APIs), and even software developer tools. Your architecture will have to be able to address all the foundational requirements that we discuss in Chapter 1:

- Capture
- Integrate
- Organize
- Analyze
- Act

Figure 4-1 presents the layered reference architecture we introduce in Chapter 1. It can be used as a framework for how to think about big data technologies that can address functional requirements for your big data projects.

This is a comprehensive stack, and you may focus on certain aspects initially based on the specific problem you are addressing. However, it is important to understand the entire stack so that you are prepared for the future. You'll no doubt use different elements of the stack depending on the problem you're addressing.

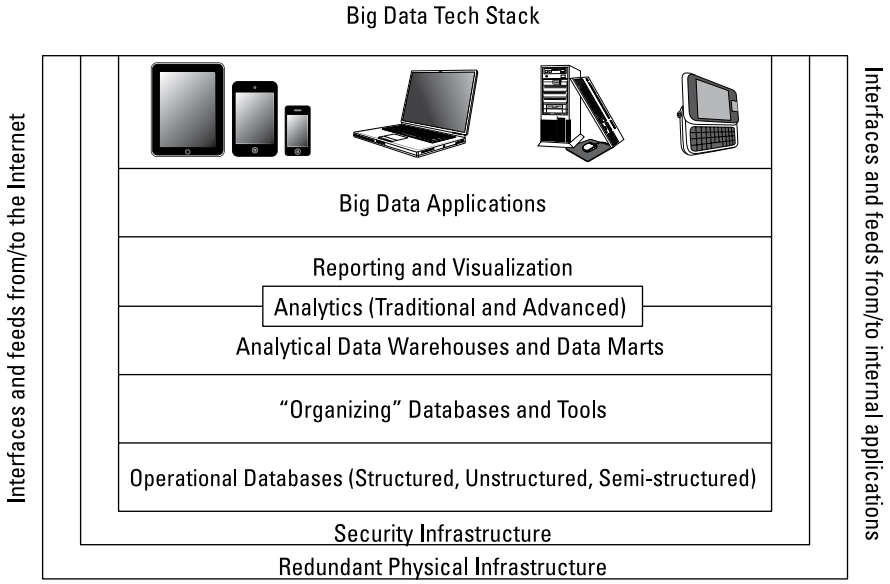

Figure 4-1: The big data technology stack.

Layer 0: Redundant Physical Infrastructure

At the lowest level of the stack is the physical infrastructure — the hardware, network, and so on. Your company might already have a data center or made investments in physical infrastructures, so you're going to want to find a way to use the existing assets. Big data implementations have very specific requirements on all elements in the reference architecture, so you need to examine these requirements on a layer-by-layer basis to ensure that your implementation will perform and scale according to the demands of your business. As you start to think about your big data implementation, it is important to have some overarching principles that you can apply to the approach. A prioritized list of these principles should include statements about the following:

- **Performance:** How responsive do you need the system to be? Performance, also called *latency,* is often measured end to end, based on a single transaction or query request. Very fast (high-performance, low-latency) infrastructures tend to be very expensive.
- **Availability:** Do you need a 100 percent uptime guarantee of service? How long can your business wait in the case of a service interruption or failure? Highly available infrastructures are also very expensive.

- **Scalability:** How big does your infrastructure need to be? How much disk space is needed today and in the future? How much computing power do you need? Typically, you need to decide what you need and then add a little more scale for unexpected challenges.
- **Flexibility:** How quickly can you add more resources to the infrastructure? How quickly can your infrastructure recover from failures? The most flexible infrastructures can be costly, but you can control the costs with cloud services, where you only pay for what you actually use (see Chapter 6 for more on cloud computing).
- **Cost:** What can you afford? Because the infrastructure is a set of components, you might be able to buy the "best" networking and decide to save money on storage (or vice versa). You need to establish requirements for each of these areas in the context of an overall budget and then make trade-offs where necessary.

As big data is all about high-velocity, high-volume, and high-data variety, the physical infrastructure will literally "make or break" the implementation. Most big data implementations need to be highly available, so the networks, servers, and physical storage must be both resilient and redundant. Resiliency and redundancy are interrelated. An infrastructure, or a system, is resilient to failure or changes when sufficient redundant resources are in place, ready to jump into action. In essence, there are always reasons why even the most sophisticated and resilient network could fail, such as a hardware malfunction. Therefore, redundancy ensures that such a malfunction won't cause an outage.

Resiliency helps to eliminate single points of failure in your infrastructure. For example, if only one network connection exists between your business and the Internet, no network redundancy exists, and the infrastructure is not resilient with respect to a network outage. In large data centers with business continuity requirements, most of the redundancy is in place and can be leveraged to create a big data environment. In new implementations, the designers have the responsibility to map the deployment to the needs of the business based on costs and performance.

As more vendors provide cloud-based platform offerings, the design responsibility for the hardware infrastructure often falls to those service providers.

This means that the technical and operational complexity is masked behind a collection of services, each with specific terms for performance, availability, recovery, and so on. These terms are described in service-level agreements (SLAs) and are usually negotiated between the service provider and the customer, with penalties for noncompliance.

For example, if you contract with a managed service provider, you are theoretically absolved from the worry associated with the specifics of the physical environment and the core components of the data center. The networks, servers, operating systems, virtualization fabric, requisite management tools, and day-to-day operations are inclusive in your service agreements. In effect, this creates a virtual data center. Even with this approach, you should still know what is needed to build and run a big data deployment so that you can make the most appropriate selections from the available service offerings. Despite having an SLA, your organization still has the ultimate responsibility for performance.

Physical redundant networks

Networks should be redundant and must have enough capacity to accommodate the anticipated volume and velocity of the inbound and outbound data in addition to the "normal" network traffic experienced by the business. As you begin making big data an integral part of your computing strategy, it is reasonable to expect volume and velocity to increase.

Infrastructure designers should plan for these expected increases and try to create physical implementations that are "elastic." As network traffic ebbs and flows, so too does the set of physical assets associated with the implementation. Your infrastructure should offer monitoring capabilities so that operators can react when more resources are required to address changes in workloads.

Managing hardware: Storage and servers

Likewise, the hardware (storage and server) assets must have sufficient speed and capacity to handle all expected big data capabilities. It's of little use to have a high-speed network with slow servers because the servers will most likely become a bottleneck. However, a very fast set of storage and compute servers can overcome variable network performance. Of course, nothing will work properly if network performance is poor or unreliable.

Infrastructure operations

Another important design consideration is infrastructure operations management. The greatest levels of performance and flexibility will be present only in a well-managed environment. Data center managers need to be able to anticipate and prevent catastrophic failures so that the integrity of the data,

and by extension the business processes, is maintained. IT organizations often overlook and therefore underinvest in this area. We talk more about what's involved with operationalizing big data in Chapter 17.

Layer 1: Security Infrastructure

Security and privacy requirements for big data are similar to the requirements for conventional data environments. The security requirements have to be closely aligned to specific business needs. Some unique challenges arise when big data becomes part of the strategy, which we briefly describe in this list:

- **Data access:** User access to raw or computed big data has about the same level of technical requirements as non-big data implementations. The data should be available only to those who have a legitimate business need for examining or interacting with it. Most core data storage platforms have rigorous security schemes and are often augmented with a federated identity capability, providing appropriate access across the many layers of the architecture.
- **Application access:** Application access to data is also relatively straightforward from a technical perspective. Most application programming interfaces (APIs) offer protection from unauthorized usage or access. This level of protection is probably adequate for most big data implementations.
- **Data encryption:** Data encryption is the most challenging aspect of security in a big data environment. In traditional environments, encrypting and decrypting data really stresses the systems' resources. With the volume, velocity, and varieties associated with big data, this problem is exacerbated. The simplest (brute-force) approach is to provide more and faster computational capability. However, this comes with a steep price tag — especially when you have to accommodate resiliency requirements. A more temperate approach is to identify the data elements requiring this level of security and to encrypt only the necessary items.
- **Threat detection:** The inclusion of mobile devices and social networks exponentially increases both the amount of data and the opportunities for security threats. It is therefore important that organizations take a multiperimeter approach to security.

We talk more about big data security and governance in Chapter 19. We also discuss how big data is being used to *help* detect threats and other security issues.

Interfaces and Feeds to and from Applications and the Internet

So, physical infrastructure enables everything and security infrastructure protects all the elements in your big data environment. The next level in the stack is the interfaces that provide bidirectional access to all the components of the stack — from corporate applications to data feeds from the Internet. An important part of the design of these interfaces is the creation of a consistent structure that is shareable both inside and perhaps outside the company as well as with technology partners and business partners.

For decades, programmers have used APIs to provide access to and from software implementations. Tool and technology providers will go to great lengths to ensure that it is a relatively straightforward task to create new applications using their products. Although very helpful, it is sometimes necessary for IT professionals to create custom or proprietary APIs exclusive to the company. You might need to do this for competitive advantage, a need unique to your organization, or some other business demand, and it is not a simple task. APIs need to be well documented and maintained to preserve the value to the business. For this reason, some companies choose to use API toolkits to get a jump-start on this important activity.

API toolkits have a couple of advantages over internally developed APIs. The first is that the API toolkits are products that are created, managed, and maintained by an independent third party. Second, they are designed to solve a specific technical requirement. If you need APIs for web applications or mobile applications, you have several alternatives to get you started.

Take a REST

No discussion of big data APIs would be complete without examining a technology called Representational State Transfer (REST). REST was designed specifically for the Internet and is the most commonly used mechanism for connecting one web resource (a server) to another web resource (a client). A RESTful API provides a standardized way to create a temporary relationship (also called *loose coupling*) between and among web resources. As the name implies, loosely coupled resources are not rigidly connected and are resilient to changes in the networks and other infrastructure components. For example, if your refrigerator breaks in the middle of the night, you need to buy a new one. You might have to wait until a retail store opens before you can do so. In addition, you may need to wait longer for delivery. This is very similar to web resources using RESTful APIs. Your request may not be answered until the service is available to address it. Many, if not all, big data technologies support REST, as you see in subsequent chapters.

Big data challenges require a slightly different approach to API development or adoption. Because much of the data is unstructured and is generated outside of the control of your business, a new technique, called Natural Language Processing (NLP), is emerging as the preferred method for interfacing between big data and your application programs. NLP allows you to formulate queries with natural language syntax instead of a formal query language like SQL. For most big data users, it will be much easier to ask "List all married male consumers between 30 and 40 years old who reside in the southeastern United States and are fans of NASCAR" than to write a 30-line SQL query for the answer.

One way to deal with interfaces is to implement a "connector" factory. This connector factory adds a layer of abstraction and predictability to the process, and it leverages many of the lessons and techniques used in Service Oriented Architecture (SOA). For more information on SOA, check out *Service Oriented Architecture (SOA) For Dummies,* 2nd Edition (written by our team and published by John Wiley & Sons, Inc.).

Because most data gathering and movement have very similar characteristics, you can design a set of services to gather, cleanse, transform, normalize, and store big data items in the storage system of your choice. To create as much flexibility as necessary, the factory could be driven with interface descriptions written in Extensible Markup Language (XML). This level of abstraction allows specific interfaces to be created easily and quickly without the need to build specific services for each data source.

In practice, you could create a description of SAP or Oracle application interfaces using something like XML. Each interface would use the same underlying software to migrate data between the big data environment and the production application environment independent of the specifics of SAP or Oracle. If you need to gather data from social sites on the Internet (such as Facebook, Google+, and so on), the practice would be identical. Describe the interfaces to the sites in XML, and then engage the services to move the data back and forth. Typically, these interfaces are documented for use by internal and external technologists.

Layer 2: Operational Databases

At the core of any big data environment are the database engines containing the collections of data elements relevant to your business. These engines need to be fast, scalable, and rock solid. They are not all created equal, and certain big data environments will fare better with one engine than another, or more likely with a mix of database engines. For example, although it is possible to use relational database management systems (RDBMSs) for all your big data implementations, it is not practical to do so because of performance, scale, or even cost. A number of different database technologies are

available, and you must take care to choose wisely. We talk more about these choices in Chapter 7.

No single right choice exists regarding database languages. Although SQL is the most prevalent database query language in use today, other languages may provide a more effective or efficient way of solving your big data challenges. It is useful to think of the engines and languages as tools in an "implementer's toolbox." Your job is to choose the right tool.

For example, if you use a relational model, you will probably use SQL to query it. However, you can also use alternative languages like Python or Java. It is very important to understand what types of data can be manipulated by the database and whether it supports true transactional behavior. Database designers describe this behavior with the acronym *ACID.* It stands for

- **Atomicity:** A transaction is "all or nothing" when it is atomic. If any part of the transaction or the underlying system fails, the entire transaction fails.
- **Consistency:** Only transactions with valid data will be performed on the database. If the data is corrupt or improper, the transaction will not complete and the data will not be written to the database.
- **Isolation:** Multiple, simultaneous transactions will not interfere with each other. All valid transactions will execute until completed and in the order they were submitted for processing.
- **Durability:** After the data from the transaction is written to the database, it stays there "forever."

Table 4-1 offers a comparison of these characteristics of SQL and NoSQL databases.

Table 4-1 Important Characteristics of SQL and NoSQL Databases

Engine	*Query Language*	*MapReduce*	*Data Types*	*Transactions*	*Examples*
Relational	SQL, Python, C	No	Typed	ACID	PostgreSQL, Oracle, DB/2
Columnar	Ruby	Hadoop	Predefined and typed	Yes, if enabled	HBase
Graph	Walking, Search, Cypher	No	Untyped	ACID	Neo4J
Document	Commands	JavaScript	Typed	No	MongoDB, CouchDB
Key-value	Lucene, Commands	JavaScript	BLOB, semityped	No	Riak, Redis

After you understand your requirements and understand what data you're gathering, where to put it, and what to do with it, you need to organize it so that it can be consumed for analytics, reporting, or specific applications.

Layer 3: Organizing Data Services and Tools

Organizing data services and tools capture, validate, and assemble various big data elements into contextually relevant collections. Because big data is massive, techniques have evolved to process the data efficiently and seamlessly. MapReduce, covered in Chapter 8, is one heavily used technique. Suffice it to say here that many of these organizing data services are MapReduce engines, specifically designed to optimize the organization of big data streams.

Organizing data services are, in reality, an ecosystem of tools and technologies that can be used to gather and assemble data in preparation for further processing. As such, the tools need to provide integration, translation, normalization, and scale. Technologies in this layer include the following:

- **A distributed file system:** Necessary to accommodate the decomposition of data streams and to provide scale and storage capacity
- **Serialization services:** Necessary for persistent data storage and multilanguage remote procedure calls (RPCs)
- **Coordination services:** Necessary for building distributed applications (locking and so on)
- **Extract, transform, and load (ETL) tools:** Necessary for the loading and conversion of structured and unstructured data into Hadoop
- **Workflow services:** Necessary for scheduling jobs and providing a structure for synchronizing process elements across layers

In Chapters 9 and 10, we examine Hadoop, the most widely used set of products for organizing big data. It is an open source initiative maintained by the Apache Foundation.

Layer 4: Analytical Data Warehouses

The data warehouse , and its companion the data mart, have long been the primary techniques that organizations use to optimize data to help decision

makers. Typically, data warehouses and marts contain normalized data gathered from a variety of sources and assembled to facilitate analysis of the business. Data warehouses and marts simplify the creation of reports and the visualization of disparate data items. They are generally created from relational databases, multidimensional databases, flat files, and object databases — essentially any storage architecture. In a traditional environment, where performance may not be the highest priority, the choice of the underlying technology is driven by the requirements for the analysis, reporting, and visualization of the company data.

As the organization of the data and its readiness for analysis are key, most data warehouse implementations are kept current via batch processing. The problem is that batch-loaded data warehouses and data marts may be insufficient for many big data applications. The stress imposed by high-velocity data streams will likely require a more real-time approach to big data warehouses. This doesn't mean that you won't be creating and feeding an analytical data warehouse or a data mart with batch processes. Rather, you may end up having multiple data warehouses or data marts, and the performance and scale will reflect the time requirements of the analysts and decision makers.

Because many data warehouses and data marts are comprised of data gathered from various sources within a company, the costs associated with the cleansing and normalizing of the data must also be addressed. With big data, you find some key differences:

- Traditional data streams (from transactions, applications, and so on) can produce a lot of disparate data.
- Dozens of new data sources also exist, each of them needing some degree of manipulation before it can be timely and useful to the business.
- Content sources will also need to be cleansed, and these may require different techniques than you might use with structured data.

Historically, the contents of data warehouses and data marts were organized and delivered to business leaders in charge of strategy and planning. With big data, we are seeing a new set of teams that are leveraging data for decision making. Many big data implementations provide real-time capabilities, so businesses should be able to deliver content to enable individuals with operational roles to address issues such as customer support, sales opportunities, and service outages in near real time. In this way, big data helps move action from the back office to the front office.

In Chapter 11, we examine several technology approaches for big data warehousing, with recommendations for using them effectively and efficiently.

Big Data Analytics

Existing analytics tools and techniques will be very helpful in making sense of big data. However, there is a catch. The algorithms that are part of these tools have to be able to work with large amounts of potentially real-time and disparate data. The infrastructure that we cover earlier in the chapter will need to be in place to support this. And, vendors providing analytics tools will also need to ensure that their algorithms work across distributed implementations. Because of these complexities, we also expect a new class of tools to help make sense of big data.

We list three classes of tools in this layer of our reference architecture. They can be used independently or collectively by decision makers to help steer the business. The three classes of tools are as follows:

- **Reporting and dashboards:** These tools provide a "user-friendly" representation of the information from various sources. Although a mainstay in the traditional data world, this area is still evolving for big data. Some of the tools that are being used are traditional ones that can now access the new kinds of databases collectively called NoSQL (Not Only SQL). We explore NoSQL databases in Chapter 7.
- **Visualization:** These tools are the next step in the evolution of reporting. The output tends to be highly interactive and dynamic in nature. Another important distinction between reports and visualized output is animation. Business users can watch the changes in the data utilizing a variety of different visualization techniques, including mind maps, heat maps, infographics, and connection diagrams. Often, reporting and visualization occur at the end of the business activity. Although the data may be imported into another tool for further computation or examination, this is the final step.
- **Analytics and advanced analytics:** These tools reach into the data warehouse and process the data for human consumption. Advanced analytics should explicate trends or events that are transformative, unique, or revolutionary to existing business practice. Predictive analytics and sentiment analytics are good examples of this science. Issues relating to analytics are covered in greater detail in Part IV, including Chapters 12, 13, and 14.

Big Data Applications

Custom and third-party applications offer an alternative method of sharing and examining big data sources. Although all the layers of the reference

architecture are important in their own right, this layer is where most of the innovation and creativity is evident.

These applications are either horizontal, in that they address problems that are common across industries, or vertical, in that they are intended to help solve an industry-specific problem. Needless to say, you have many applications to choose from, and many more coming. We expect categories of commercially available big data applications to grow as fast or faster than the adoption rate of the underlying technology. The most prevalent categories as of this writing are log data applications (Splunk, Loggly), ad/media applications (Bluefin, DataXu), and marketing applications (Bloomreach, Myrrix). Solutions are also being developed for the healthcare industry, manufacturing, and transportation management, to name a few.

Like any other custom application development initiative, the creation of big data applications will require structure, standards, rigor, and well-defined APIs. Most business applications wanting to leverage big data will need to subscribe to APIs across the entire stack. It may be necessary to process raw data from the low-level data stores and combine the raw data with synthesized output from the warehouses. As you might expect, the operative term is *custom,* and it creates a different type of pressure on the big data implementation.

Big data moves fast and changes in the blink of an eye, so software development teams need to be able to rapidly create applications germane to solving the business challenge of the moment. Companies may need to think about creating development "tiger teams," which rapidly respond to changes in the business environment by creating and deploying applications on demand. In fact, it may be more appropriate to think of these applications as "semicustom" because they involve more assembly than actual low-level coding.

Over time, we expect certain types of applications will be created, in context, by the end user, who can assemble the solution from a palette of components. Needless to say, this is where the structure and standardization are most necessary. Software developers need to create consistent, standardized development environments and devise new development practices for rapid rollout of big data applications.