Having started with strategy and identified the SMART questions around customers, finance, operations, resources and risk you need to figure out what metrics and data you actually need access to in order to answer those questions, which in turn will help you to deliver your strategy.

1.9. BIG DATA TOOLS AND TECHNIQUES [18]

1.9.1 Understanding Big Data Storage [18]

As we have discussed in much of the book so far, most, if not all big data applications achieve their performance and scalability through deployment on a collection of storage and computing resources bound together within a runtime environment. In essence, the ability to design, develop, and implement a big data application is directly dependent on an awareness of the architecture of the underlying computing platform, both from a hardware and more importantly from a software perspective.
One commonality among the different appliances and frameworks is the adaptation of tools to leverage the combination of collections of four key computing resources:

1. ***Processing capability*** often referred to as a CPU, processor, or node. Generally speaking, modern processing nodes often incorporate multiple cores that are individual CPUs that share the node's memory and are managed and scheduled together, allowing multiple tasks to be run simultaneously; this is known as multithreading.
2. ***Memory***, which holds the data that the processing node is currently working on. Most single node machines have a limit to the amount of memory.
3. ***Storage***, providing persistence of data—the place where datasets are loaded, and from which the data is loaded into memory to be processed.
4. ***Network***, which provides the "pipes" through which datasets are exchanged between different processing and storage nodes.

Because single-node computers are limited in their capacity, they cannot easily accommodate massive amounts of data. That is why the high-performance platforms are composed of collections of computers in which the massive amounts of data and requirements for processing can be distributed among a pool of resources.

1.9.2 A Generals Overview of High-Performance Architecture [18]

Most high-performance platforms are created by connecting multiple nodes together via a variety of network topologies. Specialty appliances may differ in the specifics of the configurations, as do software appliances. However, the general architecture distinguishes the management of computing resources (and corresponding allocation of tasks) and the management of the data across the network of storage nodes, as is seen in Figure 1.12.

In this configuration, a master job manager oversees the pool of processing nodes, assigns tasks, and monitors the activity. At the same time, a storage manager oversees the data storage pool and distributes datasets across the collection of storage resources. While there is no a priori requirement that there be any colocation of data and processing tasks, it is beneficial from a performance perspective to ensure that the threads process data that is local, or close to minimize the costs of data access latency.

To get a better understanding of the layering and interactions within a big data platform, we will examine the Apache Hadoop software stack, since the architecture is published and open for review.

Hadoop is essentially a collection of open source projects that are combined to enable a software-based big data appliance. We begin with the core aspects of Hadoop's utilities, upon which the next layer in the stack is propped, namely Hadoop distributed file systems (HDFS) and MapReduce. A new generation framework for job scheduling and cluster management is being developed under the name YARN.

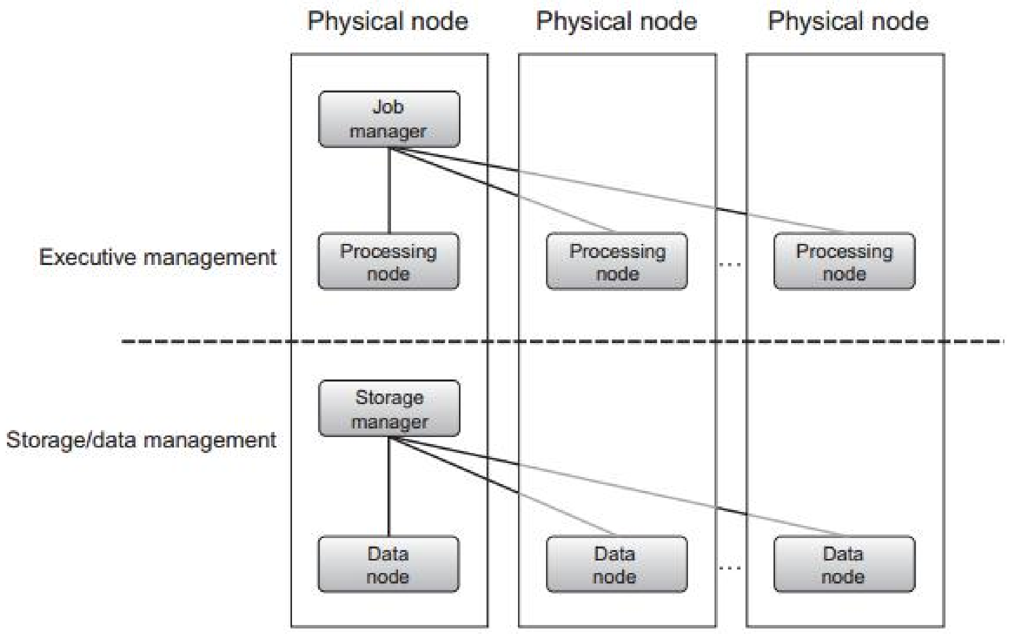

Figure 1.12 Typical organization of resources in a big data platform [18]

1.9.3 HDFS [18]

HDFS attempts to enable the storage of large files, and does this by distributing the data among a pool of data nodes. A single name node (sometimes referred to as NameNode) runs in a cluster, associated with one or more data nodes, and provide the management of a typical hierarchical file organization and namespace. The name node effectively coordinates the interaction with the distributed data nodes.

The creation of a file in HDFS appears to be a single file, even though it blocks "chunks" of the file into pieces that are stored on individual data nodes.
The name node maintains metadata about each file as well as the history of changes to file metadata. That metadata includes an enumeration of the managed files, properties of the files, and the file system, as well as the mapping of blocks to files at the data nodes.
The data node itself does not manage any information about the logical HDFS file; rather, it treats each data block as a separate file and shares the critical information with the name node.
Once a file is created, as data is written to the file, it is actually cached in a temporary file. When the amount of the data in that temporary file is enough to fill a block in an HDFS file, the name node is alerted to transition that temporary file into a block that is committed to a permanent data node, which is also then incorporated into the file management scheme.
HDFS provides a level of fault tolerance through data replication. An application can specify the degree of replication (i.e., the number of copies made) when a file is created. The name node also manages replication, attempting to optimize the marshaling and communication of replicated data in relation to the cluster's configuration and corresponding efficient use of network bandwidth. This is increasingly important in larger environments consisting of multiple racks of data servers, since communication among nodes on the same rack is generally faster than between server node sin different racks. HDFS attempts to maintain awareness of data node locations across the hierarchical configuration.
In essence, HDFS provides performance through distribution of data and fault tolerance through replication. The result is a level of robustness for reliable massive file storage. Enabling this level of reliability should be facilitated through a number of key tasks for failure management, some of which are already deployed within HDFS while others are not currently implemented:

• ***Monitoring***: There is a continuous "heartbeat" communication between the data nodes to the name node. If a data node's heartbeat is not heard by the name node, the data node is considered to have failed and is no longer available. In this case, a replica is employed to replace the failed node, and a change is made to the replication scheme.
• ***Rebalancing***: This is a process of automatically migrating blocks of data from one data node to another when there is free space, when there is an increased demand for the data and moving it may improve performance (such as moving from a traditional disk drive to a solid-state drive that is much faster or can accommodate increased numbers of simultaneous accesses), or an increased need to replication in reaction to more frequent node failures.
• ***Managing integrity***: HDFS uses checksums, which are effectively "digital signatures", associated with the actual data stored in a file (often calculated as a numerical function of

the values within the bits of the files) that can be used to verify that the data stored corresponds to the data shared or received. When the checksum calculated for a retrieved block does not equal the stored checksum of that block, it is considered an integrity error. In that case, the requested block will need to be retrieved from a replica instead.
• ***Metadata replication***: The metadata files are also subject to failure, and HDFS can be configured to maintain replicas of the corresponding metadata files to protect against corruption.
• ***Snapshots***: This is incremental copying of data to establish a point in time to which the system can be rolled back.[19, 20]

These concepts map to specific internal protocols and services that HDFS uses to enable a large-scale data management file system that can run on commodity hardware components. The ability to use HDFS solely as a means for creating a scalable and expandable file system for maintaining rapid access to large datasets provides a reasonable value proposition from an Information Technology perspective:

• decreasing the cost of specialty large-scale storage systems;
• providing the ability to rely on commodity components;
• enabling the ability to deploy using cloud-based services;
• reducing system management costs.

1.9.4 MapReduce and YARN [18]

In Hadoop, MapReduce originally combined both job management and oversight and the programming model for execution. The MapReduce execution environment employs a master/slave execution model, in which one master node (called the JobTracker) manages a pool of slave computing resources (called TaskTrackers) that arecalled upon to do the actual work. The role of the JobTracker is to manage the resources with some specific responsibilities, including managing the TaskTrackers, continually monitoring their accessibility and availability, and the different aspects of job management that include scheduling tasks, tracking the progress of assigned tasks, reacting to identified failures, and ensuring fault tolerance of the execution. The role of the TaskTracker is much simpler: wait for a task assignment, initiate and execute the requested task, and provide status back to the JobTracker on a periodic basis. Different clients can make requests from the JobTracker, which becomes the sole arbitrator for allocation of resources.
There are limitations within this existing MapReduce model. First, the programming paradigm is nicely suited to applications where there is locality between the processing and the data, but applications that demand data movement will rapidly become bogged down by network latency issues. Second, not all applications are easily mapped to the MapReduce model, yet applications developed using alternative programming methods

would still need the MapReduce system for job management. Third, the allocation of processing nodes within the cluster is fixed through allocation of certain nodes as "map slots" versus "reduce slots." When the computation is weighted toward one of the phases, the nodes assigned to the other phase are largely unused, resulting in processor underutilization.

This is being addressed in future versions of Hadoop through the segregation of duties within a revision called YARN. In this approach, overall resource management has been centralized while management of resources at each node is now performed by a local NodeManager. In addition, there is the concept of an ApplicationMaster that is associated with each application that directly negotiates with the central ResourceManager for resources while taking over the responsibility for monitoring progress and tracking status. Pushing this responsibility to the application environment allows greater flexibility in the assignment of resources as well as be more effective in scheduling to improve node utilization.

Last, the YARN approach allows applications to be better aware of the data allocation across the topology of the resources within a cluster. This awareness allows for improved colocation of compute and data resources, reducing data motion, and consequently, reducing delays associated with data access latencies. The result should be increased scalability and performance.[21]

1.9.5 Expanding the Big Data Application Ecosystem [18]

At this point, a few key points regarding the development of big data applications should be clarified. First, despite the simplicity of downloading and installing the core components of a big data development and execution environment like Hadoop, designing, developing, and deploying analytic applications still requires some skill and expertise. Second, one must differentiate between the tasks associated with application design and development and the tasks associated with architecting the big data system, selecting and connecting its components, system configuration, as well as system monitoring and continued maintenance.

In other words, transitioning from an experimental "laboratory" system into a production environment demands more than just access to the computing, memory, storage, and network resources. There is a need to expand the ecosystem to incorporate a variety of additional capabilities, such as configuration management, data organization, application development, and optimization, as well as additional capabilities to support analytical processing. Our examination of a prototypical big data platform engineered using Hadoop continues by looking at a number of additional components that might typically be considered as part of the ecosystem.

1.9.6 ZOOKEEPER [18]

Whenever there are multiple tasks and jobs running within a single distributed environment, there is a need for configuration management and synchronization of various aspects of naming and coordination. The project's web page specifies it more clearly: "Zookeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services."[22]
Zookeeper manages a naming registry and effectively implements a system for managing the various static and ephemeral named objects in a hierarchical manner, much like a file system. In addition, it enables coordination for exercising control over shared resources that are impacted by race conditions (in which the expected output of a process is impacted by variations in timing) and deadlock (in which multiple tasks vying for control of the same resource effectively lock each other out of any task's ability to use the resource). Shared coordination services like those provided in Zookeeper allow developers to employ these controls without having to develop them from scratch.

1.9.7 HBASE [18]

HBase is another example of a nonrelational data management environment that distributes massive datasets over the underlying Hadoop framework. HBase is derived from Google's BigTable and is a column-oriented data layout that, when layered on top of Hadoop, provides a fault-tolerant method for storing and manipulating large data tables. Data stored in a columnar layout is amenable to compression, which increases the amount of data that can be represented while decreasing the actual storage footprint. In addition, HBase supports in-memory execution.
HBase is not a relational database, and it does not support SQL queries. There are some basic operations for HBase: **Get** (which access a specific row in the table), **Put** (which stores or updates a row in the table), **Scan** (which iterates over a collection of rows in the table), and **Delete** (which removes a row from the table). Because it can be used to organize datasets, coupled with the performance provided by the aspects of the columnar orientation, HBase is a reasonable alternative as a persistent storage paradigm when running MapReduce applications.

1.9.8 HIVE [18]

One of the often-noted issues with MapReduce is that although it provides a methodology for developing and executing applications that use massive amounts of data, it is not more than that. And while the data can be managed within files using HDFS, many business applications expect representations of data in structured database tables. That was the motivation for the development of Hive, which (according to the Apache Hive web site[23]) is a "data warehouse system for Hadoop that facilitates easy data summarization, ad-hoc queries, and the analysis of large datasets stored in Hadoop compatible file systems." Hive is specifically engineered for data warehouse querying and reporting and is not intended for use as within transaction processing systems that require real-time query execution or transaction semantics for consistency at the row level.

Hive is layered on top of the file system and execution framework for Hadoop and enables applications and users to organize data in a structured data warehouse and therefore query the data using a query language called HiveQL that is similar to SQL (the standard Structured Query Language used for most modern relational database management systems). The Hive system provides tools for extracting/transforming/loading data (ETL) into a variety of different data formats. And because the data warehouse system is built on top of Hadoop, it enables native access to the MapReduce model, allowing programmers to develop custom Map and Reduce functions that can be directly integrated into HiveQL queries. Hive provides scalability and extensibility for batch-style queries for reporting over large datasets that are typically being expanded while relying on the fault tolerant aspects of the underlying Hadoop execution model.

1.9.9 PIG [18]

Even though the MapReduce programming model is relatively straightforward, it still takes some skill and understanding of both parallel and distributed programming and Java to best take advantage of the model. The Pig project is an attempt at simplifying the application development process by abstracting some of the details away through a higher level programming language called Pig Latin. According to the project's web site[24], Pig's high-level programming language allows the developer to specify how the analysis is performed. In turn, a compiler transforms the Pig Latin specification into MapReduce programs.

The intent is to embed a significant set of parallel operators and functions contained within a control sequence of directives to be applied to datasets in a way that is somewhat similar to the way SQL statements are applied to traditional structured databases. Some examples include generating datasets, filtering out subsets, joins, splitting datasets, removing duplicates. For simple applications, using Pig provides significant ease of development, and more complex tasks can be engineered as sequences of applied operators.

In addition, the use of a high-level language also allows the compiler to identify opportunities for optimization that might have been ignored by an inexperienced programmer. At the same time, the Pig environment allows developers to create new user defined functions (UDFs) that can subsequently be incorporated into developed programs.

1.9.10 MAHOUT [18]

Attempting to use big data for analytics would be limited without any analytics capabilities. Mahout is a project to provide a library of scalable implementations of machine learning algorithms on top of MapReduce and Hadoop. As is described at the project's home page[25],
Mahout's library includes numerous well-known analysis methods including:

• ***Collaborative filtering and other user and item-based recommender algorithms***, which is used to make predictions about an individual's interest or preferences through comparison with a multitude of others that may or may not share similar characteristics.
• ***Clustering***, including K-Means, Fuzzy K-Means, Mean Shift, and Dirichlet process clustering algorithms to look for groups, patterns, and commonality among selected cohorts in a population.
• ***Categorization*** using Naïve Bayes or decision forests to place items into already defined categories.
• ***Text mining*** and topic modeling algorithms for scanning text and assigning contextual meanings.
• ***Frequent pattern mining***, which is used for market basket analysis, comparative health analytics, and other patterns of correlation within large datasets.

Mahout also supports other methods and algorithms. The availability of implemented libraries for these types of analytics free the development team to consider the types of problems to be analyzed and more specifically, the types of analytical models that can be applied to seek the best answers.

Variable	Intent	Technical Requirements
Predisposition to parallelization	Number and type of processing node(s)	Number or processors Types of processors
Size of data to be persistently stored	Amount and allocation of disk space for distributed file system	Size of disk drives
		Number of disk drives
		Type of drives (SSD versus magnetic versus optical)
		Bus configuration (shared everything versus shared nothing, for example)
Amount of data to be accessible in memory	Amount and allocation of core memory	Amount of RAM memory Cache memories
Need for cross-node communication	Optimize speed and bandwidth	Network/cabinet configuration
		Network speed
		Network bandwidth
Types of data organization	Data management requirements	File management organization
		Database requirements
		Data orientation (row versus column)
		Type of data structures
Developer skill set	Development tools	Types of programming tools, compilers, execution models, debuggers, etc.
Types of algorithms	Analytic functionality requirements	Data warehouse/marts for OLAP Data mining and predictive analytics

Table 1.9 Variables to Consider When Framing Big Data Environment [18]

1.9.11 Considerations [18]

Big data analytics applications employ a variety of tools and techniques for implementation. When organizing your thoughts about developing those applications, it is important to think about the parameters that will frame your needs for technology evaluation and acquisition, sizing and configuration, methods of data organization, and required algorithms to be used or developed from scratch.

Prior to diving directly into downloading and installing software, focus on the types of big data business applications and their corresponding performance scaling needs, such as those listed in Table 1.9.

The technical requirements will guide both the hardware and the software configurations. This also allows you to align the development of the platform with the business application development needs.