**UNIT 2**

How big data analytics works

Big data analytics refers to collecting, processing, cleaning, and analyzing large datasets to help organizations operationalize their big data.

1. Collect Data

Data collection looks different for every organization. With today’s technology, organizations can gather both structured and unstructured data from a variety of sources — from cloud storage to mobile applications to in-store IoT sensors and beyond. [Some data will be stored in data warehouses](https://www.tableau.com/learn/articles/data-warehousing) where [business intelligence tools and solutions](https://www.tableau.com/learn/articles/business-intelligence/choosing-bi-platforms) can access it easily. Raw or unstructured data that is too diverse or complex for a warehouse may be assigned metadata and stored in a data lake.

2. Process Data

Once data is collected and stored, it must be organized properly to get accurate results on analytical queries, especially when it’s large and unstructured. Available data is growing exponentially, making data processing a challenge for organizations. One processing option is **batch processing**, which looks at large data blocks over time. Batch processing is useful when there is a longer turnaround time between collecting and analyzing data. **Stream processing** looks at small batches of data at once, shortening the delay time between collection and analysis for quicker decision-making. Stream processing is more complex and often more expensive.

3. Clean Data

Data big or small requires scrubbing to improve data quality and get stronger results; all data must be formatted correctly, and any duplicative or irrelevant [data must be eliminated or accounted for](https://www.tableau.com/learn/whitepapers/data-prep-best-practices). Dirty data can obscure and mislead, creating flawed insights.

4. Analyze Data

Getting big data into a usable state takes time. Once it’s ready, advanced analytics processes can turn big data into big insights. Some of these big data analysis methods include:

* **Data mining** sorts through large datasets to identify patterns and relationships by identifying anomalies and creating data clusters.
* **Predictive analytics** uses an organization’s historical data to make predictions about the future, identifying upcoming risks and opportunities.
* **Deep learning** imitates human learning patterns by using artificial intelligence and machine learning to layer algorithms and find patterns in the most complex and abstract data.

## Visualizations

The choice of the visualization tools, serving databases and web frameworks is driven by the requirements of the application. Visualizations can be static, dynamic or interactive. Static visualizations are used when you have the analysis results stored in a serving database and you simply want to display the results. However, if your application demands the results to updated regularly, then you would require dynamic visualizations (with live widgets, plots, or gauges). If you want your application to accept inputs from the user and display the results, then you would require interactive visualizations.

# Big Data Stack

While the Hadoop framework has been one of the most popular frameworks for big data analytics, there are several types of computational tasks for which Hadoop does not work well. With the help of the mapping between the analytics types and the computational “giants” as shown in Figure 1.1, we will identify the cases where Hadoop works and where it does not, and describe the motivation for having a Big Data stack that can be used for various types of analytics and computational tasks.

Hadoop is an open source framework for distributed batch processing of massive scale data using the MapReduce programming model. The MapReduce programming model is useful for applications in which the data involved is so massive that it would not fit on a single machine. In such applications, the data is typically stored on a distributed file system (such as Hadoop Distributed File System - HDFS). MapReduce programs take advantage of locality of data and the data processing takes place on the nodes where the data resides. In other words, the computation is moved to where the data resides, as opposed the traditional way of moving the data from where it resides to where the computation is done. MapReduce is best suited for descriptive analytics and the basic statistics computational tasks because the operations involved can be done in parallel (for example, computing counts, mean, max/min, distinct, top-N, filtering and joins). Many of these operations are completed with a single MapReduce job. For more complex tasks, multiple MapReduce jobs can be chained together. However, when the computations are iterative in nature, where a MapReduce job has to be repeatedly run, MapReduce takes a performance hit because of the overhead involved in fetching the data from HDFS in each iteration.

For other types of analytics and computational tasks, there are other alternative frameworks which we will discuss as a part of the Big Data Stack. In this Chapter, we propose and describe a big data stack comprising of proven and open-source big data frameworks that form the foundation of this book. Figure 1.3 shows the big data stack with the Chapter numbers highlighted for the various blocks in the stack. The successive chapters in the book describe these blocks in detail along with hands-on examples and case studies. We have used

Python as the primary programming language for the examples and case studies throughout the book. Let us look at each block one-by-one:

## Raw Data Sources

In any big data analytics application or platform, before the data is processed and analyzed, it must be captured from the raw data sources into the big data systems and frameworks. Some of the examples of raw big data sources include:

**Logs**: Logs generated by web applications and servers which can be used for performance monitoring

*•*

**Transactional Data**: Transactional data generated by applications such as eCommerce, Banking and Financial

*•*

**Social Media**: Data generated by social media platforms

*•*

**Databases**: Structured data residing in relational databases

*•*

**Sensor Data**: Sensor data generated by Internet of Things (IoT) systems **Clickstream Data**: Clickstream data generated by web applications which can be used to analyze browsing patterns of the users

*•*

*•*

**Surveillance Data**: Sensor, image and video data generated by surveillance systems **Healthcare Data**: Healthcare data generated by Electronic Health Record (EHR) and other healthcare applications

*•*

*•*

**Network Data**: Network data generated by network devices such as routers and firewalls

*•*

## Data Access Connectors

The Data Access Connectors includes tools and frameworks for collecting and ingesting data from various sources into the big data storage and analytics frameworks. The choice of the data connector is driven by the type of the data source. Let us look at some data connectors and frameworks which can be used for collecting and ingesting data. These connectors can include both wired and wireless connections.

**Publish-Subscribe Messaging**: Publish-Subscribe is a communication model that involves publishers, brokers and consumers. Publishers are the source of data. Publishers send the data to the topics which are managed by the broker

*•*

**Source-Sink Connectors**: Source-Sink connectors allow efficiently collecting, aggregating and moving data from various sources (such as server logs, databases, social media, streaming sensor data from Internet of Things devices and other sources) into a centralized data store (such as a distributed file system). Flume uses a data flow model that comprises sources, channels and sinks.

*•*

**Database Connectors**: Database connectors can be used for importing data from relational database management systems into big data storage and analytics frameworks for analysis. In Chapter-5 we have described Apache Sqoop, which is a tool that allows importing data from relational databases.

*•*

 

**Raw Data**

Streams

Databases

Records

Sensors

Logs

Figure 1.3: Big Data Stack

**Interactive**

**Querying Ch-9**

**Batch Analysis Ch-7,11**

**Serving Databases,**

**Web Frameworks, Visualization Frameworks Ch-10,12**

**Connectors**

**NoSQL**

(HBase, Cassandra, DynamoDB, MongoDB)

**Web/App**

**Servers**

**Real-time Analysis Ch-8**

**Data Access Connectors Ch-5**

**Custom Connectors** (REST,

WebSocket, AWS IoT,

Azure IoT Hub)

**Queues** (RabbitMQ, ZeroMQ, REST MQ,

Amazon SQS)

**SQL**

(Sqoop)

**Source-Sink**

(Flume)

**Publish- Subscribe** (Kafka, Amazon Kinesis)

**Data Storage Ch-6**

**NoSQL**

(HBase)

**Distributed Filesystem** (HDFS)

**Visualization Frameworks** (Lightning, pyGal, Seaborn)

**In-Memory** (Spark Streaming)

**Stream Processing** (Storm)

**Web Frameworks** (Django)

**Search**

(Solr)

**Machine Learning**

(Spark Mlib, H2O)

**SQL**

(MySQL)

**Workﬂow Scheduling** (Oozie)

**Script**

(Pig)

**DAG**

(Spark)

**MapReduce**

(Hadoop)

**Analytic SQL** (Hive, BigQuery, Spark SQL, Redshift)

*•*

**Messaging Queues**: Messaging queues are useful for push-pull messaging where the producers push data to the queues and the consumers pull the data from the queues. The producers and consumers do not need to be aware of each other.

**Custom Connectors**: Custom connectors can be built based on the source of the data and the data collection requirements. Some examples of custom connectors include: custom connectors for collecting data from social networks, custom connectors for NoSQL databases and connectors for Internet of Things (IoT). In Chapter-5 we have described custom connectors based on REST, WebSocket and MQTT.

*•*

## Data Storage

The data storage block in the big data stack includes distributed filesystems and non-relational (NoSQL) databases, which store the data collected from the raw data sources using the data access connectors. With the data stored in HDFS, it can be analyzed with various big data analytics frameworks built on top of HDFS. For certain analytics applications, it is preferable to store data in a NoSQL database such as HBase. HBase is a scalable, non-relational, distributed, column-oriented database that provides structured data storage for large tables. The architecture of HBase and its use cases are described in Chapter-4.

## Batch Analytics

The batch analytics block in the big data stack includes various frameworks which allow analysis of data in batches. These include the following:

**Hadoop-MapReduce**: Hadoop is a framework for distributed batch processing of big data. The MapReduce programming model is used to develop batch analysis jobs which are executed in Hadoop clusters. Examples of MapReduce jobs and case studies of using Hadoop-MapReduce for batch analysis

*•*

**Pig**: Pig is a high-level data processing language which makes it easy for developers to write data analysis scripts which are translated into MapReduce programs by the Pig compiler. Examples of using Pig for batch data analysis are described in Chapter-7. **Oozie**: Oozie is a workflow scheduler system that allows managing Hadoop jobs. With Oozie, you can create workflows which are a collection of actions (such as MapReduce jobs) arranged as Direct Acyclic Graphs (DAG).

*•*

*•*

**Spark**: Apache Spark is an open source cluster computing framework for data analytics. Spark includes various high-level tools for data analysis such as Spark Streaming for streaming jobs, Spark SQL for analysis of structured data, MLlib machine learning library for Spark, and GraphX for graph processing.

*•*

**Solr**: Apache Solr is a scalable and open-source framework for searching data.

*•*

**Machine Learning:** Spark MLlib is the Spark’s machine learning library which provides implementations of various machine learning algorithms. H2O is an open source predictive analytics framework which provides implementations of various machine learning algorithms.

## Real-time Analytics

The real-time analytics block includes the Apache Storm and Spark Streaming frameworks. Apache Storm is a framework for distributed and fault-tolerant real-time computation. Storm can be used for real-time processing of streams of data. Storm can consume data from a variety of sources such as publish-subscribe messaging frameworks (such as Kafka or Kinesis), messaging queues (such as RabbitMQ or ZeroMQ) and other custom connectors. Spark Streaming is a component of Spark which allows analysis of streaming data such as sensor data, click stream data, web server logs, for instance. The streaming data is ingested and analyzed in micro-batches. Spark Streaming enables scalable, high throughput and fault-tolerant stream processing.

## Interactive Querying

Interactive querying systems allow users to query data by writing statements in SQL-like languages. We describe the following interactive querying systems, with examples

**Spark SQL**: Spark SQL is a component of Spark which enables interactive querying. Spark SQL is useful for querying structured and semi-structured data using SQL-like queries.

*•*

**Hive**: Apache Hive is a data warehousing framework built on top of Hadoop. Hive provides an SQL-like query language called Hive Query Language, for querying data residing in HDFS.

*•*

**Amazon Redshift**: Amazon Redshift is a fast, massive-scale managed data warehouse service. Redshift specializes in handling queries on datasets of sizes up to a petabyte or more parallelizing the SQL queries across all resources in the Redshift cluster.

*•*

**Google BigQuery**: Google BigQuery is a service for querying massive datasets. BigQuery allows querying datasets using SQL-like queries.

*•*

## Serving Databases, Web & Visualization Frameworks

While the various analytics blocks process and analyze the data, the results are stored in serving databases for subsequent tasks of presentation and visualization. These serving databases allow the analyzed data to be queried and presented in the web applications we describe the following SQL and NoSQL databases which can be used as serving databases:

**MySQL**: MySQL is one of the most widely used Relational Database Management System (RDBMS) and is a good choice to be used as a serving database for data analytics applications where the data is structured.

*•*

**Amazon DynamoDB**: Amazon DynamoDB is a fully-managed, scalable,

*•*

high-performance NoSQL database service from Amazon. DynamoDB is an excellent choice for a serving database for data analytics applications as it allows storing and

retrieving any amount of data and the ability to scale up or down the provisioned throughput.

**Cassandra**: Cassandra is a scalable, highly available, fault tolerant open source non-relational database system.

*•*

**MongoDB**: MongoDB is a document oriented non-relational database system. MongoDB is powerful, flexible and highly scalable database designed for web applications and is a good choice for a serving database for data analytics applications.

*•*

Django is based on the

Model-Template-View architecture and provides a separation of the data model from the business rules and the user interface. While web applications can be useful for presenting the results, specialized visualizing tools and frameworks can help in understanding the data, and the analysis results quickly and easily.

**Lightning**: Lightning is a framework for creating web-based interactive visualizations. **Pygal**: The Python Pygal library is an easy to use charting library which supports charts of various types.

*•*

*•*

**Seaborn**: Seaborn is a Python visualization library for plotting attractive statistical plots.

*•*