Real-Time Big Data Analytics

This book provides you with the skills required to quickly design, implement, and deploy your real-time analytics using real-world examples of Big Data use cases. From the beginning of the book, we will cover the basics of varied real-time data processing frameworks and technologies.

Moving on, we'll familiarize you with Amazon Kinesis for real-time data processing on cloud. We will further develop your understanding of real-time analytics through a comprehensive review of Apache Spark along with the high-level architecture and the building blocks of a Spark program.

You will learn how to transform your data, get an output from transformations, and persist your results using Spark RDDs, using an interface called Spark SQL to work with Spark.

At the end of this book, we will introduce Spark Streaming, and we will walk you through the emerging Lambda Architecture (LA), which provides a hybrid platform for Big Data processing by combining real-time and precomputed batch data to provide near-real-time view of incoming data.

Who this book is written for

If you are a Big Data architect, developer, or a programmer who wants to develop applications/frameworks to implement real-time analytics using open source technologies, then this book is for you.

What you will learn from this book

- Explore Big Data technologies and frameworks
- Work through practical challenges and use cases of real-time analytics versus batch analytics
- Develop real-world use cases for processing and analyzing data in real-time using the programming paradigm of Apache Spark
- Handle and process real-time transactional data
- Optimize and tune Apache Spark for varied workloads and production deployments
- Process and stream data with Amazon Kinesis and Elastic MapReduce
- Perform interactive and exploratory data analytics using Spark SQL
- Perform real-time analysis using Spark Streaming
- Develop common enterprise architectures and applications for real-time and batch analytics

In this package, you will find:

- The author's biography
- A preview chapter from the book, Chapter 1 'Introducing the Big Data Technology Landscape and Analytics Platform'
- A synopsis of the book’s content
- More information on Real-Time Big Data Analytics
About the Authors

**Sumit Gupta** is a seasoned professional, innovator, and technology evangelist with over 100 man months of experience in architecting, managing, and delivering enterprise solutions revolving around a variety of business domains, such as hospitality, healthcare, risk management, insurance, and so on. He is passionate about technology and overall he has 15 years of hands-on experience in the software industry and has been using Big Data and cloud technologies over the past 4 to 5 years to solve complex business problems.

**Shilpi Saxena** is an IT professional and also a technology evangelist. She is an engineer who has had exposure to various domains (machine to machine space, healthcare, telecom, hiring, and manufacturing). She has experience in all the aspects of conception and execution of enterprise solutions. She has been architecting, managing, and delivering solutions in the Big Data space for the last 3 years; she also handles a high-performance and geographically-distributed team of elite engineers.

Shilpi has more than 12 years (3 years in the Big Data space) of experience in the development and execution of various facets of enterprise solutions both in the products and services dimensions of the software industry. An engineer by degree and profession, she has worn varied hats, such as developer, technical leader, product owner, tech manager, and so on, and she has seen all the flavors that the industry has to offer. She has architected and worked through some of the pioneers’ production implementations in Big Data on Storm and Impala with autoscaling in AWS.

Preface

Processing historical data for the past 10-20 years, performing analytics, and finally producing business insights is the most popular use case for today’s modern enterprises.

Enterprises have been focusing on developing data warehouses (https://en.wikipedia.org/wiki/Data_warehouse) where they want to store the data fetched from every possible data source and leverage various BI tools to provide analytics over the data stored in these data warehouses. But developing data warehouses is a complex, time consuming, and costly process, which requires a considerable investment, both in terms of money and time.

No doubt that the emergence of Hadoop and its ecosystem have provided a new paradigm or architecture to solve large data problems where it provides a low cost and scalable solution which processes terabytes of data in a few hours which earlier could have taken days. But this is only one side of the coin. Hadoop was meant for batch processes while there are bunch of other business use cases that are required to perform analytics and produce business insights in real or near real-time (subseconds SLA). This was called real-time analytics (RTA) or near real-time analytics (NRTA) and sometimes it was also termed as "fast data" where it implied the ability to make near real-time decisions and enable "orders-of-magnitude" improvements in elapsed time to decisions for businesses.

A number of powerful, easy to use open source platforms have emerged to solve these enterprise real-time analytics data use cases. Two of the most notable ones are Apache Storm and Apache Spark, which offer real-time data processing and analytics capabilities to a much wider range of potential users. Both projects are a part of the Apache Software Foundation and while the two tools provide overlapping capabilities, they still have distinctive features and different roles to play.

Interesting isn't it?
Let's move forward and jump into the nitty gritty of real-time Big Data analytics with Apache Storm and Apache Spark. This book provides you with the skills required to quickly design, implement, and deploy your real-time analytics using real-world examples of Big Data use cases.

**What this book covers**

*Chapter 1, Introducing the Big Data Technology Landscape and Analytics Platform*, sets the context by providing an overview of the Big Data technology landscape, the various kinds of data processing that are handled on Big Data platforms, and the various types of platforms available for performing analytics. It introduces the paradigm of distributed processing of large data in batch and real-time or near real-time. It also talks about the distributed databases to handle high velocity/frequency reads or writes.

*Chapter 2, Getting Acquainted with Storm*, introduces the concepts, architecture, and programming with Apache Storm as a real-time or near real-time data processing framework. It talks about the various concepts of Storm, such as spouts, bolts, Storm parallelism, and so on. It also explains the usage of Storm in the world of real-time Big Data analytics with sufficient use cases and examples.

*Chapter 3, Processing Data with Storm*, is focused on various internals and operations, such as filters, joins, and aggregators exposed by Apache Storm to process the streaming of data in real or near real-time. It showcases the integration of Storm with various input data sources, such as Apache Kafka, sockets, filesystems, and so on, and finally leverages the Storm JDBC framework for persisting the processed data. It also talks about the various enterprise concerns in stream processing, such as reliability, acknowledgement of messages, and so on, in Storm.

*Chapter 4, Introduction to Trident and Optimizing Storm Performance*, examines the processing of transactional data in real or near real-time. It introduces Trident as a real time processing framework which is used primarily for processing transactional data. It talks about the various constructs for handling transactional use cases using Trident. This chapter also talks about various concepts and parameters available and their applicability for monitoring, optimizing, and performance tuning the Storm framework and its jobs. It touches the internals of Storm such as LMAX, ring buffer, ZeroMQ, and more.

*Chapter 5, Getting Acquainted with Kinesis*, talks about the real-time data processing technology available on the cloud—the Kinesis service for real-time data processing from Amazon Web Services (AWS). It starts with the explanation of the architecture and components of Kinesis and then illustrates an end-to-end example of real-time alert generation using various client libraries, such as KCL, KPL, and so on.
Preface

Chapter 6, *Getting Acquainted with Spark*, introduces the fundamentals of Apache Spark along with the high-level architecture and the building blocks for a Spark program. It starts with the overview of Spark and talks about the applications and usage of Spark in varied batch and real-time use cases. Further, the chapter talks about high-level architecture and various components of Spark and finally towards the end, the chapter also discusses the installation and configuration of a Spark cluster and execution of the first Spark job.

Chapter 7, *Programming with RDDs*, provides a code-level walkthrough of Spark RDDs. It talks about various kinds of operations exposed by RDD APIs along with their usage and applicability to perform data transformation and persistence. It also showcases the integration of Spark with NoSQL databases, such as Apache Cassandra.

Chapter 8, *SQL Query Engine for Spark – Spark SQL*, introduces a SQL style programming interface called Spark SQL for working with Spark. It familiarizes the reader with how to work with varied datasets, such as Parquet or Hive and build queries using DataFrames or raw SQL; it also makes recommendations on best practices.

Chapter 9, *Analysis of Streaming Data Using Spark Streaming*, introduces another extension of Spark—Spark Streaming for capturing and processing streaming data in real or near real-time. It starts with the architecture of Spark and also briefly talks about the varied APIs and operations exposed by Spark Streaming for data loading, transformations, and persistence. Further, the chapter also talks about the integration of Spark SQL and Spark Streaming for querying data in real time. Finally, towards the end, it also discusses the deployment and monitoring aspects of Spark Streaming jobs.

Chapter 10, *Introducing Lambda Architecture*, walks the reader through the emerging Lambda Architecture, which provides a hybrid platform for Big Data processing by combining real-time and pre-computed batch data to provide a near real-time view of the data. It leverages Apache Spark and discusses the realization of Lambda Architecture with a real life use case.
The Big Data paradigm has emerged as one of the most powerful in next-generation data storage, management, and analytics. IT powerhouses have actually embraced the change and have accepted that it’s here to stay.

What arrived just as Hadoop, a storage and distributed processing platform, has really graduated and evolved. Today, we have whole panorama of various tools and technologies that specialize in various specific verticals of the Big Data space.

In this chapter, you will become acquainted with the technology landscape of Big Data and analytics platforms. We will start by introducing the user to the infrastructure, the processing components, and the advent of Big Data. We will also discuss the needs and use cases for near real-time analysis.

This chapter will cover the following points that will help you to understand the Big Data technology landscape:

- Infrastructure of Big Data
- Components of the Big Data ecosystem
- Analytics architecture
- Distributed batch processing
- Distributed databases (NoSQL)
- Real-time and stream processing
Big Data – a phenomenon
The phrase Big Data is not just a new buzzword, it's something that arrived slowly and captured the entire arena. The arrival of Hadoop and its alliance marked the end of the age for the long undefeated reign of traditional databases and warehouses.

Today, we have a humongous amount of data all around us, in each and every sector of society and the economy; talk about any industry, it's sitting and generating loads of data—for instance, manufacturing, automobiles, finance, the energy sector, consumers, transportation, security, IT, and networks. The advent of Big Data as a field/domain/concept/theory/idea has made it possible to store, process, and analyze these large pools of data to get intelligent insight, and perform informed and calculated decisions. These decisions are driving the recommendations, growth, planning, and projections in all segments of the economy and that's why Big Data has taken the world by storm.

If we look at the trends in the IT industry, there was an era when people were moving from manual computation to automated, computerized applications, then we ran into an era of enterprise level applications. This era gave birth to architectural flavors such as SAAS and PaaS. Now, we are into an era where we have a huge amount of data, which can be processed and analyzed in cost-effective ways. The world is moving towards open source to get the benefits of reduced license fees, data storage, and computation costs. It has really made it lucrative and affordable for all sectors and segments to harness the power of data. This is making Big Data synonymous with low cost, scalable, highly available, and reliable solutions that can churn huge amounts of data at incredible speed and generate intelligent insights.

The Big Data dimensional paradigm
To begin with, in simple terms, Big Data helps us deal with the three Vs: volume, velocity, and variety. Recently, two more Vs—veracity and value—were added to it, making it a five-dimensional paradigm:
• **Volume:** This dimension refers to the amount of data. Look around you; huge amounts of data are being generated every second—it may be the e-mail you send, Twitter, Facebook, other social media, or it can just be all the videos, pictures, SMS, call records, or data from various devices and sensors. We have scaled up the data measuring metrics to terabytes, zettabytes and yottabytes—they are all humongous figures. Look at Facebook, it has around 10 billion messages each day; consolidated across all users, we have nearly 5 billion "likes" a day; and around 400 million photographs are uploaded each day. Data statistics, in terms of volume, are startling; all the data generated from the beginning of time to 2008 is kind of equivalent to what we generate in a day today, and I am sure soon it will be an hour. This volume aspect alone is making the traditional database unable to store and process this amount of data in a reasonable and useful time frame, though a Big Data stack can be employed to store, process, and compute amazingly large datasets in a cost-effective, distributed, and reliably efficient manner.

• **Velocity:** This refers to the data generation speed, or the rate at which data is being generated. In today’s world, where the volume of data has made a tremendous surge, this aspect is not lagging behind. We have loads of data because we are generating it so fast. Look at social media; things are circulated in seconds and they become viral, and the insight from social media is analyzed in milliseconds by stock traders and that can trigger lot of activity in terms of buying or selling. At target point of sale counters, it takes a few seconds for a credit card swipe and, within that, fraudulent transaction processing, payment, bookkeeping, and acknowledgement are all done. Big Data gives me power to analyze the data at tremendous speed.

• **Variety:** This dimension tackles the fact that the data can be unstructured. In the traditional database world, and even before that, we were used to a very structured form of data that kind of neatly fitted into the tables. But today, more than 80 percent of data is unstructured; for example, photos, video clips, social media updates, data from a variety of sensors, voice recordings, and chat conversations. Big Data lets you store and process this unstructured data in a very structured manner; in fact, it embraces the variety.
Introducing the Big Data Technology Landscape and Analytics Platform

- **Veracity**: This is all about validity and the correctness of data. How accurate and usable is the data? Not everything out of millions and zillions of data records is corrected, accurate, and referable. That's what veracity actually is: how trustworthy the data is, and what the quality of data is. Two examples of data with veracity are Facebook and Twitter posts with nonstandard acronyms or typos. Big Data has brought to the table the ability to run analytics on this kind of data. One of the strong reasons for the volume of data is its veracity.

- **Value**: As the name suggests, this is the value the data actually holds. Unarguably, it's the most important V or dimension of Big Data. The only motivation for going towards Big Data for the processing of super-large datasets is to derive some valuable insight from it; in the end, it's all about cost and benefits.

**The Big Data ecosystem**

For a beginner, the landscape can be utterly confusing. There is vast arena of technologies and equally varied use cases. There is no single go-to solution; every use case has a custom solution and this widespread technology stack and lack of standardization is making Big Data a difficult path to tread for developers. There are a multitude of technologies that exist which can draw meaningful insight out of this magnitude of data.

Let's begin with the basics: the environment for any data analytics application creation should provide for the following:

- Storing data
- Enriching or processing data
- Data analysis and visualization

If we get to specialization, there are specific Big Data tools and technologies available; for instance, ETL tools such as Talend and Pentaho; Pig batch processing, Hive, and MapReduce; real-time processing from Storm, Spark, and so on; and the list goes on. Here's the pictorial representation of the vast Big Data technology landscape, as per Forbes:
It clearly depicts the various segments and verticals within the Big Data technology canvas:

- Platforms such as Hadoop and NoSQL
- Analytics such as HDP, CDH, EMC, Greenplum, DataStax, and more
- Infrastructure such as Teradata, VoltDB, MarkLogic, and more
- **Infrastructure as a Service (IaaS)** such as AWS, Azure, and more
- Structured databases such as Oracle, SQL server, DB2, and more
- **Data as a Service (DaaS)** such as INRIX, LexisNexis, Factual, and more

And, beyond that, we have a score of segments related to specific problem area such as **Business Intelligence (BI)**, analytics and visualization, advertisement and media, log data and vertical apps, and so on.

**The Big Data infrastructure**

Technologies providing the capability to store, process, and analyze data are the core of any Big Data stack. The era of tables and records ran for a very long time, after the standard relational data store took over from file-based sequential storage. We were able to harness the storage and compute power very well for enterprises, but eventually the journey ended when we ran into the five Vs.
At the end of its era, we could see our, so far, robust RDBMS struggling to survive in a cost-effective manner as a tool for data storage and processing. The scaling of traditional RDBMS at the compute power expected to process a huge amount of data with low latency came at a very high price. This led to the emergence of new technologies that were low cost, low latency, and highly scalable at low cost, or were open source. Today, we deal with Hadoop clusters with thousands of nodes, hurling and churning thousands of terabytes of data.

The key technologies of the Hadoop ecosystem are as follows:

- **Hadoop**: The yellow elephant that took the data storage and computation arena by surprise. It's designed and developed as a distributed framework for data storage and computation on commodity hardware in a highly reliable and scalable manner. Hadoop works by distributing the data in chunks over all the nodes in the cluster and then processing the data concurrently on all the nodes. Two key moving components in Hadoop are mappers and reducers.

- **NoSQL**: This is an abbreviation for No-SQL, which actually is not the traditional structured query language. It's basically a tool to process a huge volume of multi-structured data; widely known ones are HBase and Cassandra. Unlike traditional database systems, they generally have no single point of failure and are scalable.

- **MPP** (short for Massively Parallel Processing) databases: These are computational platforms that are able to process data at a very fast rate. The basic working uses the concept of segmenting the data into chunks across different nodes in the cluster, and then processing the data in parallel. They are similar to Hadoop in terms of data segmentation and concurrent processing at each node. They are different from Hadoop in that they don't execute on low-end commodity machines, but on high-memory, specialized hardware. They have SQL-like interfaces for the interaction and retrieval of data, and they generally end up processing data faster because they use in-memory processing. This means that, unlike Hadoop that operates at disk level, MPP databases load the data into memory and operate upon the collective memory of all nodes in the cluster.
Components of the Big Data ecosystem

The next step on journey to Big Data is to understand the levels and layers of abstraction, and the components around the same. The following figure depicts some common components of Big Data analytical stacks and their integration with each other. The caveat here is that, in most of the cases, HDFS/Hadoop forms the core of most of the Big-Data-centric applications, but that's not a generalized rule of thumb.

Talking about Big Data in a generic manner, its components are as follows:

- A storage system can be one of the following:
  - HDFS (short for Hadoop Distributed File System) is the storage layer that handles the storing of data, as well as the metadata that is required to complete the computation
  - NoSQL stores that can be tabular stores such as HBase or key-value based columnar Cassandra

![Analytic Big Data Components](http://wikibon.org/w/images/0/03/BigDataComponents.JPG)
Introducing the Big Data Technology Landscape and Analytics Platform

• A computation or logic layer can be one of the following:
  ° **MapReduce**: This is a combination of two separate processes, the mapper and the reducer. The mapper executes first and takes up the raw dataset and transforms it to another key-value data structure. Then, the reducer kicks in, which takes up the map created by the mapper job as an input, and collates and converges it into a smaller dataset.
  ° **Pig**: This is another platform that's put on top of Hadoop for processing, and it can be used in conjunction with or as a substitute for MapReduce. It is a high-level language and is widely used for creating processing components to analyze very large datasets. One of the key aspects is that its structure is amendable to various degrees of parallelism. At its core, it has a compiler that translates Pig scripts to MapReduce jobs.
    It is used very widely because:
    ° Programming in Pig Latin is easy
    ° Optimizing the jobs is efficient and easy
    ° It is extendible

• Application logic or interaction can be one of the following:
  ° **Hive**: This is a data warehousing layer that's built on top of the Hadoop platform. In simple terms, Hive provides a facility to interact with, process, and analyze HDFS data with Hive queries, which are very much like SQL. This makes the transition from the RDBMS world to Hadoop easier.
  ° **Cascading**: This is a framework that exposes a set of data processing APIs and other components that define, share, and execute the data processing over the Hadoop/Big Data stack. It's basically an abstracted API layer over Hadoop. It's widely used for application development because of its ease of development, creation of jobs, and job scheduling.

• Specialized analytics databases, such as:
  ° Databases such as Netezza or Greenplum have the capability for scaling out and are known for a very fast data ingestion and refresh, which is a mandatory requirement for analytics models.
The Big Data analytics architecture

Now that we have skimmed through the Big Data technology stack and the components, the next step is to go through the generic architecture for analytical applications.

We will continue the discussion with reference to the following figure:

If you look at the diagram, there are four steps on the workflow of an analytical application, which in turn lead to the design and architecture of the same:

- Business solution building (dataset selection)
- Dataset processing (analytics implementation)
- Automated solution
- Measured analysis and optimization

Now, let's dive deeper into each segment to understand how it works.
Building business solutions

This is the first and most important step for any application. This is the step where the application architects and designers identify and decide upon the data sources that will be providing the input data to the application for analytics. The data could be from a client dataset, a third party, or some kind of static/dimensional data (such as geo coordinates, postal code, and so on). While designing the solution, the input data can be segmented into business-process-related data, business-solution-related data, or data for technical process building. Once the datasets are identified, let's move to the next step.

Dataset processing

By now, we understand the business use case and the dataset(s) associated with it. The next steps are data ingestion and processing. Well, it's not that simple; we may want to make use of an ingestion process and, more often than not, architects end up creating an ETL (short for Extract Transform Load) pipeline. During the ETL step, the filtering is executed so that we only apply processing to meaningful and relevant data. This filtering step is very important. This is where we are attempting to reduce the volume so that we have to only analyze meaningful/valued data, and thus handle the velocity and veracity aspects. Once the data is filtered, the next step could be integration, where the filtered data from various sources reaches the landing data mart. The next step is transformation. This is where the data is converted to an entity-driven form, for instance, Hive table, JSON, POJO, and so on, and thus marking the completion of the ETL step. This makes the data ingested into the system available for actual processing.

Depending upon the use case and the duration for which a given dataset is to be analyzed, it's loaded into the analytical data mart. For instance, my landing data mart may have a year's worth of credit card transactions, but I just need one day's worth of data for analytics. Then, I would have a year's worth of data in the landing mart, but only one day's worth of data in the analytics mart. This segregation is extremely important because that helps in figuring out where I need real-time compute capability and which data my deep learning application operates upon.

Solution implementation

Now, we will implement various aspects of the solution and integrate them with the appropriate data mart. We can have the following:

- **Analytical Engine**: This executes various batch jobs, statistical queries, or cubes on the landing data mart to arrive at the projections and trends based on certain indexes and variances.
• **Dashboard/Workbench**: This depicts some close to real-time information on to some UX interfaces. These components generally operate on low latency, close to real-time, analytical data marts.

• **Auto learning synchronization mechanism**: Perfect for advanced analytical application, this captures patterns and evolves the data management methodologies. For example, if I am a mobile operator at a tourist place, I might be more cautious about my resource usage during the day and at weekends, but over a period of time I may learn that during vacations I see a surge in roaming, so I can learn and build these rules into my data mart and ensure that data is stored and structured in an analytics-friendly manner.

**Presentation**

Once the data is analyzed, the next and most important step in the life cycle of any application is the presentation/visualization of the results. Depending upon the target audience of the end business user, the data visualization can be achieved using a custom-built UI presentation layer, business insight reports, dashboards, charts, graphs, and so on.

The requirement could vary from autorefreshing UI widgets to canned reports and ADO queries.

**Distributed batch processing**

The first and foremost point to understand is what are the different kinds of processing that can be applied to data. Well, they fall in two broad categories:

- Batch processing
- Sequential or inline processing

The key difference between the two is that the sequential processing works on a per tuple basis, where the events are processed as they are generated or ingested into the system. In case of batch processing, they are executed in batches. This means tuples/events are not processed as they are generated or ingested. They’re processed in fixed-size batches; for example, 100 credit card transactions are clubbed into a batch and then consolidated.

Some of the key aspects of batch processing systems are as follows:

- Size of a batch or the boundary of a batch
- Batching (starting a batch and terminating a batch)
- Sequencing of batches (if required by the use case)
The batch can be identified by size (which could be \(x\) number of records, for example, a 100-record batch). The batches can be more diverse and be divided into time ranges such as hourly batches, daily batches, and so on. They can be dynamic and data-driven, where a particular sequence/pattern in the input data demarcates the start of the batch and another particular one marks its end.

Once a batch boundary is demarcated, said bundle of records should be marked as a batch, which can be done by adding a header/trailer, or maybe one consolidated data structure, and so on, bundled with a batch identifier. The batching logic also performs bookkeeping and accounting for each batch being created and dispatched for processing.

In certain specific use cases, the order of records or the sequence needs to be maintained, leading to the need to sequence the batches. In these specialized scenarios, the batching logic has to do extra processing to sequence the batches, and extra caution needs to be applied to the bookkeeping for the same.

Now that we understand what batch processing is, the next step and an obvious one is to understand what distributed batch processing is. It's a computing paradigm where the tuples/records are batched and then distributed for processing across a cluster of nodes/processing units. Once each node completes the processing of its allocated batch, the results are collated and summarized for the final results. In today's application programming, when we are used to processing a huge amount of data and get results at lightning-fast speed, it is beyond the capability of a single node machine to meet these needs. We need a huge computational cluster. In computer theory, we can add computation or storage capability by two means:

- By adding more compute capability to a single node
- By adding more than one node to perform the task

Source: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSy_pG3f3Lq7spA6rp5aVZjxKxYzBI5y2xCh0XX_CLK49kH2IyG
Vertical scaling is a paradigm where we add more compute capability; for example, add more CPUs or more RAM to an existing node or replace the existing node with a more powerful machine. This model works well only up to an extent. You may soon hit the ceiling and your needs would outgrow what the biggest possible machine can deliver. So, this model has a flaw in the scaling, and it’s essentially an issue when it comes to a single point of failure because, as you see, the entire application is running on one machine.

So you can see that vertical scaling is limited and failure prone. The higher end machines are pretty expensive too. So, the solution is horizontal scaling. I rely on clustering, where the computational capability is basically not derived from a single node, but from a collection of nodes. In this paradigm, I am operating in a model that’s scalable and there is no single point of failure.

**Batch processing in distributed mode**

For a very long time, Hadoop was synonymous with Big Data, but now Big Data has branched off to various specialized, non-Hadoop compute segments as well. At its core, Hadoop is a distributed, batch-processing compute framework that operates upon MapReduce principles.

It has the ability to process a huge amount of data by virtue of batching and parallel processing. The key aspect is that it moves the computation to the data, instead of how it works in the traditional world, where data is moved to the computation. A model that is operative on a cluster of nodes is horizontally scalable and fail-proof.

Hadoop is a solution for offline, batch data processing. Traditionally, the NameNode was a single point of failure, but the advent of newer versions and YARN (short for Yet Another Resource Negotiator) has actually changed that limitation. From a computational perspective, YARN has brought about a major shift that has decoupled MapReduce and Hadoop, and has provided the scope of integration with other real-time, parallel processing compute engines like Spark, MPI (short for Message Processing Interface), and so on.
**Push code to data**

So far, the general computational models have a data flow where the data is ingested and moved to the compute engine.

The advent of distributed batch processing made changes to this and this is depicted in the following figure. The batches of data were moved to various nodes in the compute-engine cluster. This shift was seen as a major advantage to the processing arena and has brought the power of parallel processing to the application.

Moving data to compute makes sense for low volume data. But, for a Big Data use case that has humongous data computation, moving data to the compute engine may not be a sensible idea because network latency can cause a huge impact on the overall processing time. So Hadoop has shifted the world by creating batches of input data called blocks and distributing them to each node in the cluster. Take a look at this figure:
At the initialization stage, the Big Data file is pushed into HDFS. Then, the file is split into chunks (or file blocks) by the Hadoop NameNode (master node) and is placed onto individual DataNodes (slave nodes) in the cluster for concurrent processing.

The process in the cluster called Job Tracker moves execution code or processing to the data. The compute component includes a Mapper and a Reduce class. In very simple terms, a Mapper class does the job of data filtering, transformation, and splitting. By nature of a localized compute, a Mapper instance only processes the data blocks which are local to or co-located on the same data node. This concept is called data locality or proximity. Once the Mappers are executed, their outputs are shuffled through to the appropriate Reduce nodes. A Reduce class, by its functionality, is an aggregator for compiling all the results from the mappers.

**Distributed databases (NoSQL)**

We have discussed the paradigm shift from data to computation to the paradigm of computation to data in case of Hadoop. We understand on the conceptual level how to harness the power of distributed computation. The next step is to apply the same to database level in terms of having distributed databases.

In very simple terms, a database is actually a storage structure that lets us store the data in a very structured format. It can be in the form of various data structural representations internally, such as flat files, tables, blobs, and so on. Now when we talk about a database, we generally refer to single/clustered server class nodes with huge storage and specialized hardware to support the operations. So, this can be envisioned as a single unit of storage controlled by a centralized control unit.
Distributed database, on the contrary, is a database where there is no single control unit or storage unit. It's basically a cluster of homogenous/heterogeneous nodes, and the data and the control for execution and orchestration is distributed across all nodes in the cluster. So to understand it better, we can use an analogy that, instead of all the data going into a single huge box, now the data is spread across multiple boxes. The execution of this distribution, the bookkeeping and auditing of this data distribution, and the retrieval process are managed by multiple control units. In a way, there is no single point of control or storage. One important point is that these multiple distributed nodes can exist physically or virtually.

Do not relate this to the concept of parallel systems, where the processors are tightly coupled and it all constitutes a single database system. A distributed database system is a relatively loosely coupled entity that shares no physical components.

Now we understand the differentiating factors for distributed databases. However, it is necessary to understand that, due to their distributed nature, these systems have some added complexity to ensure correctness and the accuracy of the day. There are two processes that play a vital role:

- **Replication**: This is tracked by a special component of distributed databases. This piece of software does the tedious job of bookkeeping for all the updates/additions/deletions that are being made to the data. Once all changes are logged, this process updates so that all copies of the data look the same and represent the state of truth.

- **Duplication**: The popularity of distributed databases is due to the fact that they don't have a single point of failure and there is always more than one copy of data available to deal with a failure situation if it happens. The process that copies one instance of data to multiple locations on the cluster generally executes at some defined interval.

Both these processes ensure that at a given point in time, more than one copy of data exists in the cluster, and all the copies of the data have the same representation of the state of truth for the data.

A NoSQL database environment is a non-relational and predominately distributed database system. Its clear advantage is that it facilitates the rapid analysis of extremely high-volume, disparate data types. With the advent of Big Data, NoSQL databases have become the cheap and scalable alternative to traditional RDBMS. The USPs they have to offer are availability and fault tolerance, which are big differentiating factors.
NoSQL offers a flexible and extensible schema model with added advantages of endless scalability, distributed setup, and the liberty of interfacing with non-SQL interfaces.

We can distinguish NoSQL databases as follows:

- **Key-value store**: This type of database belongs to some of the least complex NoSQL options. Its USP is the design that allows the storage of data in a schemaless way. All the data in this store contains an index key and an associate value (as the name suggests). Popular examples of this type of database are Cassandra, DynamoDB, Azure Table Storage (ATS), Riak, Berkeley DB, and so on.

- **Column store or wide column store**: This is designed for storing the data in rows and its data in data tables, where there are columns of data rather than rows of data, like in a standard database. They are the exact opposite of row-based databases, and this design is highly scalable and offers very high performance. Examples are HBase and Hypertable.

- **Document database**: This is an extension to the basic idea of a key-value store where documents are more complex and elaborate. It's like each document has a unique ID associated with it. This ID is used for document retrieval. These are very widely used for storing and managing document-oriented information. Examples are MongoDB and CouchDB.

- **Graph database**: As the name suggests, it's based on the graph theory of discreet mathematics. It's well designed for data where relationships can be maintained as a graph and elements are interconnected based on relations. Examples are Neo4j, polyglot, and so on.

The following table lays out the key attributes and certain dimensions that can be used for selecting the appropriate NoSQL databases:

- **Column 1**: This captures the storage structure for the data model.
- **Column 2**: This captures the performance of the distributed database on the scale of low, medium, and high.
- **Column 3**: This captures the ease of scalability of the distributed database on the scale of low, medium, and high. It notes how easily the system can be scaled in terms of capacity and processing by adding more nodes to the cluster.
- **Column 4**: Here we talk about the scale of flexibility of use and the ability to cater to diverse structured or unstructured data and use cases.
• **Column 5**: Here we talk about how complex it is to work with the system in terms of the complexity of development and modeling, the complexity of operation and maintainability, and so on.

<table>
<thead>
<tr>
<th>Data model</th>
<th>Performance</th>
<th>Scalability</th>
<th>Flexibility</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key-value store</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>None</td>
</tr>
<tr>
<td>Column Store</td>
<td>High</td>
<td>High</td>
<td>Moderate</td>
<td>Low</td>
</tr>
<tr>
<td>Document Store</td>
<td>High</td>
<td>Variable (high)</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Graph Database</td>
<td>Variable</td>
<td>Variable</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>

**Advantages of NoSQL databases**

Let's have a look at the key reasons for adopting an NoSQL database over a traditional RDBMS. Here are the key drivers that have been attributed to the shift:

• **Advent and growth of Big Data**: This is one of the prime attribute forces driving the growth and shift towards use of NoSQL.

• **High availability systems**: In today's highly competitive world, downtime can be deadly. The reality of business is that hardware failures will occur, but NoSQL database systems are built over a distributed architecture so there is no single point of failure. They also have replication to ensure redundancy of data that guarantees availability in the event of one or more nodes being down. With this mechanism, the availability across data centers is guaranteed, even in the case of localized failures. This all comes with guaranteed horizontal scaling and high performance.

• **Location independence**: This refers to the ability to perform read and write operations to the data store regardless of the physical location where that input-output operation actually occurs. Similarly, we have the ability to have any write percolated out from that location. This feature is a difficult wish to make in the RDBMS world. This is a very handy tool when it comes to designing an application that services customers in many different geographies and needs to keep data local for fast access.
• **Schemaless data models**: One of the major motivators for moving to a NoSQL database system from an old-world relational database management system (RDBMS) is the ability to handle unstructured data and it's found in most NoSQL stores. The relational data model is based on strict relations defined between tables, which themselves are very strict in definition by a determined column structure. All of this then gets organized in a schema. The backbone of RDBMS is structure and it's the biggest limitation as this makes it fall short for handling and storing unstructured data that doesn't fit into strict table structure. A NoSQL data model on the contrary doesn't have any structure and it's flexible to fit in any form, so it's called *schemaless*. It's like one size fits all, and it's able to accept structured, semistructured, or unstructured data. All this flexibility comes along with a promise of low-cost scalability and high performance.

**Choosing a NoSQL database**

When it comes to making a choice, there are certain factors that can be taken into account, but the decision is still more use-case-driven and can vary from case to case. Migration of a data store or choosing a data store is an important, conscious decision, and should be made diligently and intelligently depending on the following factors:

- Input data diversity
- Scalability
- Performance
- Availability
- Cost
- Stability
- Community

**Real-time processing**

Now that we have talked so extensively about Big Data processing and Big Data persistence in the context of distributed, batch-oriented systems, the next obvious thing to talk about is real-time or near real-time processing. Big data processing processes huge datasets in offline batch mode. When real-time stream processing is executed on the most current set of data, we operate in the dimension of *now* or the immediate past; examples are credit card fraud detection, security, and so on. Latency is a key aspect in these analytics.
The two operatives here are velocity and latency, and that's where Hadoop and related distributed batch processing systems fall short. They are designed to deliver in batch mode and can't operate at a latency of nanoseconds/milliseconds. In use cases where we need accurate results in fractions of seconds, for example, credit card fraud, monitoring business activity, and so on, we need a Complex Event Processing (CEP) engine to process and derive results at lightning fast speed.

**Storm**, initially a project from the house of Twitter, has graduated to the league of Apache and was rechristened from Twitter Storm. It was a brainchild of Nathan Marz that's now been adopted by CDH, HDP, and so on.

Apache Storm is a highly scalable, distributed, fast, reliable real-time computing system designed to process high-velocity data. Cassandra complements the compute capability by providing lightning fast reads and writes, and this is the best combination available as of now for a data store with Storm. It helps the developer to create a data flow model in which tuples flow continuously through a topology (a collection of processing components). Data can be ingested to Storm using distributed messaging queues such as Kafka, RabbitMQ, and so on. Trident is another layer of abstraction API over Storm that brings microbatching capabilities into it.

Let's take a closer look at a couple of real-time, real-world use cases in various industrial segments.

**The telecoms or cellular arena**

We are living in an era where cell phones are no longer merely calling devices. In fact, they have evolved from being phones to *smartphones*, providing access to not just calling but also facilities such as data, photographs, tracking, GPS, and so on into the hands of the consumers. Now, the data generated by cell phones or telephones is not just call data; the typical CDR (short for Call Data Record) captures voice, data, and SMS transactions. Voice and SMS transactions have existed for more than a decade and are predominantly structured as they are because of telecoms protocols worldwide; for example, CIBER, SMPP, SMSC, and so on. However, the data or IP traffic flowing in/out of these smart devices is pretty unstructured and high volume. It could be a music track, a picture, a tweet, or just about anything in the data dimension. CDR processing and billing is generally a batch job, but a lot of other things are real-time:

- **Geo-tracking of the device**: Have you noticed how quickly we get an SMS whenever we cross a state border?
• **Usage and alerts**: Have you noticed how accurate and efficient the alert that informs you about the broadband consumption limit is and suggests that you top up the same?

• **Prepaid mobile cards**: If you have ever used a prepaid system, you must have been awed at the super-efficient charge-tracking system they have in place.

### Transportation and logistics

Transportation and logistics is another useful segment that's using real-time analytics from vehicular data for transportation, logistics, and intelligent traffic management. Here's an example from McKinney's report that details how Big Data and real-time analytics are helping to handle traffic congestion on a major highway in Tel Aviv, the capital of Israel. Here's what they actually do: they monitor the receipts from the toll constantly and during the peak hours, to avert congestion, they hike the toll prices. This is a deterrent factor for the users. Once the congestion eases out during non-peak hours, the toll rates are reduced.

There may be many more use cases that can be built around the data from checkpoints/tolls to develop intelligent management of traffic, thus preventing congestion, and make better utilization of public infrastructure.

### The connected vehicle

An idea that was still in the realms of fiction until the last decade is now a reality that's being actively used by the consumer segment today. GPS and Google Maps are no news today, they are being imbibed and heavily used features.

My car's control unit has telemetry devices that capture various KPIs, such as engine temperature, fuel consumption pattern, RPM, and so on, and all this information is used by the manufacturers for analysis. In some of the cases, the user is also allowed to set and receive alerts on these KPI thresholds.

### The financial sector

This is the sector that's emerging as the biggest consumer of real-time analytics for very obvious reasons. The volume of data is huge and quickly changing; the impact of analytics and its results boils down to the money aspect. This sector needs real-time instruments for rapid and precise data analysis for data from stock exchanges, various financial institutions, market prices and fluctuations, and so on.
Summary
In this chapter, we have discussed various aspects of the Big Data technology landscape. We have talked about the terminology, definitions, acronyms, components, and infrastructure used in context with Big Data. We have also described the architecture of a Big Data analytical platform. Further on in the chapter, we also discussed the various computational methodologies starting from sequential, to batch, to distributed, and then arrived at real time. At the end of this chapter, we are sure that our readers are now well acquainted with Big Data and its characteristics.

In the next chapter, we will embark on our journey towards the real-time technology — Storm — and will see how it fits well in the arena of real-time analytics platforms.
Where to buy this book

You can buy Real-Time Big Data Analytics from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.