Distributed Data Processing

using Apache Spark

Faaiz SHAH

PhD. Computer Science
LIRMM, University of Montpellier

Pradeo Security Systems
Manager AI

May 13, 2024

Outline

Introduction

Distributed Computing

What is Distributed Computing?

A computing paradigm or environment in which components of a software system are shared among multiple computers to improve efficiency and performance.

Figure 1: Distributed Computing [1]

[1] https://cloudxlab.com/blog/introduction-to-big-data-and-distributed-computing/

Distributed Computing: Key Characteristics

- **Scalability**: Systems can easily scale out to accommodate increased load
- **Fault Tolerance**: Systems are designed to continue operating even if parts fail
- **Concurrency**: Multiple components can operate simultaneously

Distributed Computing: Benefits of Distributed Systems

- Increased computational power
- Redundancy and reliability
- Resource sharing across different geographies

Distributed Computing: Challenges

- **Network Issues**: Latency, bandwidth limitations
- **Security Concerns**: More endpoints, more vulnerabilities
- **Complexity in Management**: Difficulty in synchronizing and managing multiple systems

About the Course

Distributed Processing

Distributed processing architecture, such as Apache Spark, is a software infrastructures designed to process large amounts of data across a **cluster** of interconnected machines

They enable the **distribution** of computational **tasks** across multiple processing nodes **in parallel**, which provides **greater processing** capacity, better **scalability**, and **enhanced performance**.

Spark is one of the most popular and powerful distributed processing architectures.

Spark Key Features: Programming Model

Abstract Programming Model

An abstract programming model provides a simplified, high-level interface for programming, abstracting away complex lower-level details

- Spark offers an abstract programming model called **Resilient Distributed Datasets (RDD)**, which allows data to be processed transparently across the cluster
- RDDs are **immutable** and **fault-tolerant** collections, meaning they can be distributed across multiple computing nodes and retrieved in case of failure
- RDD is defined as an abstract class (i.e., it can not be instantiated) in Spark library

Spark Key Features: In-Memory Processing

- Spark uses **RAM** to store intermediate data and computation results, which allows for rapid access to data without having to read from disk
- This enables faster response times and more efficient task execution

Spark's in-memory processing is particularly advantageous for iterative processing because it significantly reduces the time taken to read from and write to disk, thus speeding up the iterations

Spark Key Features: Batch and Real-Time Processing

- Spark supports both **batch** data processing and **real-time** (streaming) processing
- It enables continuous analysis on real-time data streams, as well as **iterative processing** for machine learning algorithms

Iterative processing in machine learning

It involves repeatedly applying the same steps to refine the model's parameters until the model meets a specific criterion, such as a set number of iterations or a minimum error threshold

Spark Key Features: Extensive Ecosystem

- Spark has a rich ecosystem with a comprehensive library of components, including **Spark SQL** for SQL processing, **Spark Streaming** for real-time processing, **MLlib** for machine learning, **GraphX** for graph processing, and many more.
- It facilitates the development of complex applications using a coherent set of tools

Distributed processing architectures

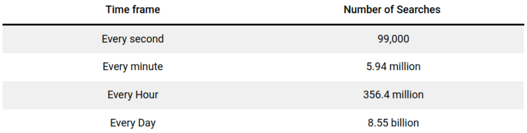
- Distributed processing architectures like Spark are used for various applications, such as:
  - big data analysis
  - real-time stream processing
  - distributed machine learning
  - personalized recommendation
  - predictive analytics, etc.
- They leverage the parallel computing power of distributed clusters for fast and efficient processing of large volumes of data

Big Data Technology

- Data has <u>not only</u> become the lifeblood of any organization, <u>but also</u> it is growing exponentially
- The challenge is how to get <u>business value</u> out of this data

What is big data ?

Big Data Technology



Time frame	Number of Searches
Every second	99,000
Every minute	5.94 million
Every Hour	356.4 million
Every Day	8.55 billion

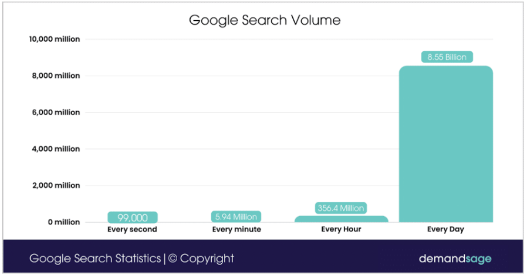


Figure 3: Google Search Volume [1]

[1] https://blog.hubspot.com/marketing/google-search-statistics

Big Data Technology

- Data has not only become the lifeblood of any organization, but also it is growing exponentially
- The challenge is how to get business value out of this data

What is big data ?

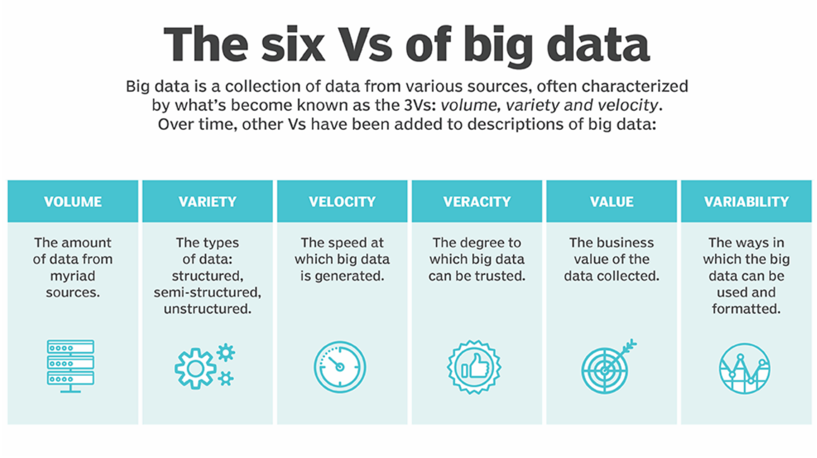
Big Data Technology

What is big data ?

1. Is it a dataset whose volume exceeds petabytes or several terabytes ?
2. A relational database table with billions of rows ?
3. a relational database table with thousands of columns ?

Although the term “big data” is hot, its definition is quite vague

Big Data Technology



The six Vs of big data

Big data is a collection of data from various sources, often characterized by what's become known as the 3Vs: *volume, variety and velocity*. Over time, other Vs have been added to descriptions of big data:

VOLUME	VARIETY	VELOCITY	VERACITY	VALUE	VARIABILITY
The amount of data from myriad sources.	The types of data: structured, semi-structured, unstructured.	The speed at which big data is generated.	The degree to which big data can be trusted.	The business value of the data collected.	The ways in which the big data can be used and formatted.

Figure 4: The six Vs of big data [1]

[1] https://www.quora.com/What-are-the-six-Vs-of-Big-Data

Big Data Technology

- Standard relational databases could not easily handle big data
- The core technology for these databases was designed several decades ago when few organizations had petabytes or even terabytes of data
- Today it is normal for some organizations to generate terabytes of data every day

Hence there was a need for new technologies that could not only process and analyze large volume of data, but also ingest large volume of data at a fast pace.

Big Data Technology

- Key driving factors for the big data technologies include:
  - **Scalability**
  - **High availability**
  - **Fault tolerance**

Big Data Technology: Apache Hadoop

- Hadoop was one of the first popular open source big data technologies.
- It is a scalable fault-tolerant system for processing large datasets across a cluster of commodity (general purpose, less expensive, easy to install and configure) servers
- Hadoop is inspired by a system invented at Google to create inverted index for its search product [1]. The another paper was titled "The Google File System" [2].
- Inspired by these papers, Doug Cutting and Mike Cafarella developed an open source implementation, which later became **Hadoop** in 2006.

2

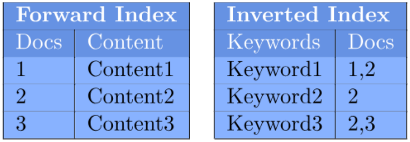
[1] "MapReduce: Simplified Data Processing on Large Clusters", by Jeffrey Dean and Sanjay Ghemawat published in 2004

[2] Ghemawat, S., Gobioff, H. and Leung, S.T., 2003, October. The Google file system

Big Data Technology: Apache Hadoop

Inverted Index

- An inverted index is a data structure used to store and organize information for efficient search and retrieval [1]
- An inverted index works by mapping each unique word or term in a set of documents to the documents in which it occurs.



Forward Index	
Docs	Content
1	Content1
2	Content2
3	Content3

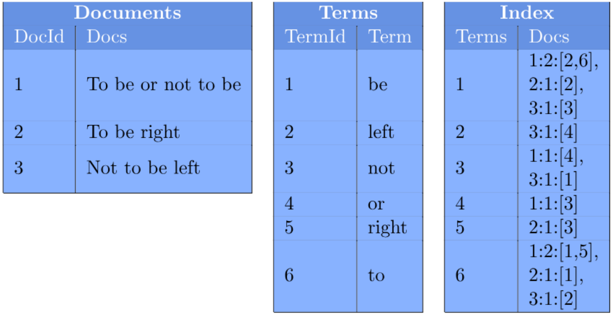
Inverted Index	
Keywords	Docs
Keyword1	1,2
Keyword2	2
Keyword3	2,3

Figure 5: Inverted Index [1]

[1] https://www.baeldung.com/cs/indexing-inverted-index

Big Data Technology: Apache Hadoop

Inverted Index



Documents	
DocId	Docs
1	To be or not to be
2	To be right
3	Not to be left

Terms	
TermId	Term
1	be
2	left
3	not
4	or
5	right
6	to

Index	
Terms	Docs
1	1:2:[2,6], 2:1:[2], 3:1:[3]
2	3:1:[4]
3	1:1:[4], 3:1:[1]
4	1:1:[3]
5	2:1:[3]
6	1:2:[1,5], 2:1:[1], 3:1:[2]

Figure 6: Inverted Index [1]

The first element of this list is 1:2:[2,6], which means that the term occurs in document 1, with frequency 2, at positions 2 and 6.

[1] https://www.baeldung.com/cs/indexing-inverted-index

Big Data Technology: Apache Hadoop

- Hadoop is open source and runs on a cluster of commodity hardware i.e., it can be scaled easily by adding cheap servers
- It is cheaper to process 100 terabyte of data on a cluster of 100 commodity computers with each computer having 1 terabyte, than on a single high-end server
- It uses scale-out architecture instead of scale-up architecture
- It provides a framework that hides the complexities of writing distributed applications
  - separating core data processing logic from distributed computing logic

[1] https://www.baeldung.com/cs/indexing-inverted-index

Big Data Technology: Apache Hadoop

- Until version 2.0, Hadoop's architecture was monolithic. All the components were tightly coupled and bundled together.
- Starting with version 2.0, Hadoop adopted a modular architecture, which allows you to mix and match Hadoop components with non-Hadoop technologies.

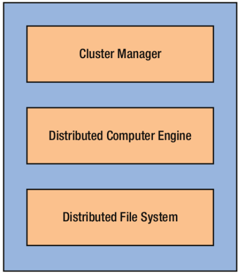


Figure 7: Hadoop Conceptual Components [1]

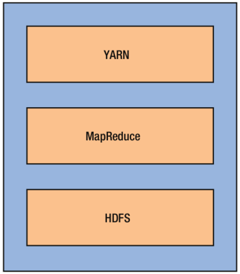


Figure 8: Hadoop Key Components [1]

[1] Guller, M. (2015). "Big data analytics with Spark", Apress.

Big Data Technology: Hadoop Distributed File System (HDFS)

- A block-structured file system that stores a file across a cluster of servers
- HDFS splits a file into fixed-size blocks, also known as *partitions* or *splits*
- The default block size is 128 MB, but it is configurable.
- Therefore, an application can parallelize file-level read and write operations

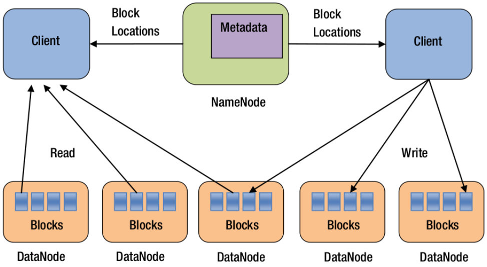


Figure 9: HDFS Architecture [1]

[1] Guller, M. (2015). "Big data analytics with Spark", Apress.

Big Data Technology: Hadoop Distributed File System (HDFS)

- Distributing a file to multiple machines increases the risk of a file becoming unavailable if one of the machines in a cluster fails
- HDFS mitigates this risk by replicating each file block on multiple machines
- The default replication factor is 3.

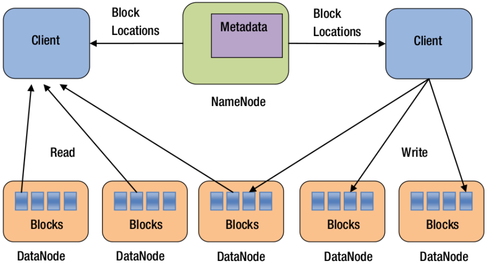


Figure 9: HDFS Architecture

Big Data Technology: Hadoop Distributed File System (HDFS)

- A HDFS cluster consists of two types of nodes: NameNode and DataNode
- A **NameNode**: manages the file system namespace, stores all the metadata for a file e.g., it tracks file names, permissions, and file block locations.
- To provide fast access to the metadata, a NameNode stores all the metadata in memory.

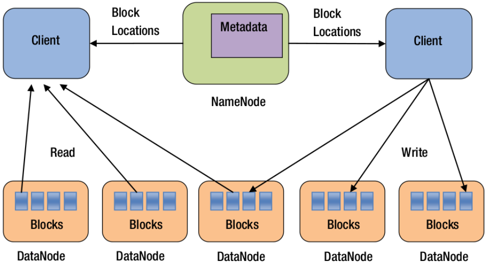


Figure 9: HDFS Architecture

Big Data Technology: Hadoop Distributed File System (HDFS)

- A **DataNode** stores the actual file content in the form of file blocks
- The DataNodes periodically sends two types of messages (Heartbeat and Blockreport) to NameNode.
- Client application read and write operations takes place by active communication between NameNode and DataNode

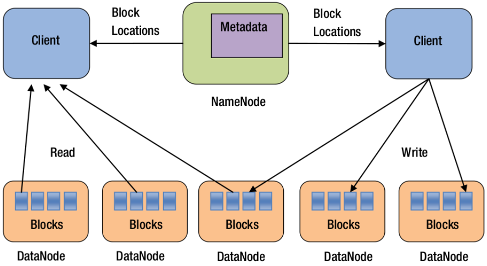


Figure 9: HDFS Architecture

Big Data Technology: MapReduce

- MapReduce is a distributed compute engine
- It is a computing framework for processing large datasets in parallel across a cluster

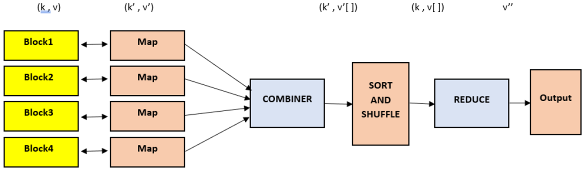
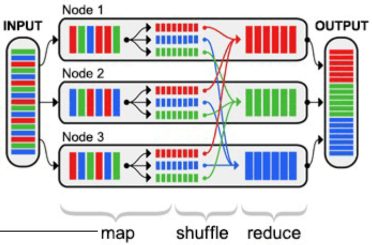


Figure 10: MapReduce [1]



[1] https://www.geeksforgeeks.org/mapreduce-understanding-with-real-life-example/, https://www.computerhope.com/jargon/m/mapreduce.htm

Big Data Technology: MapReduce

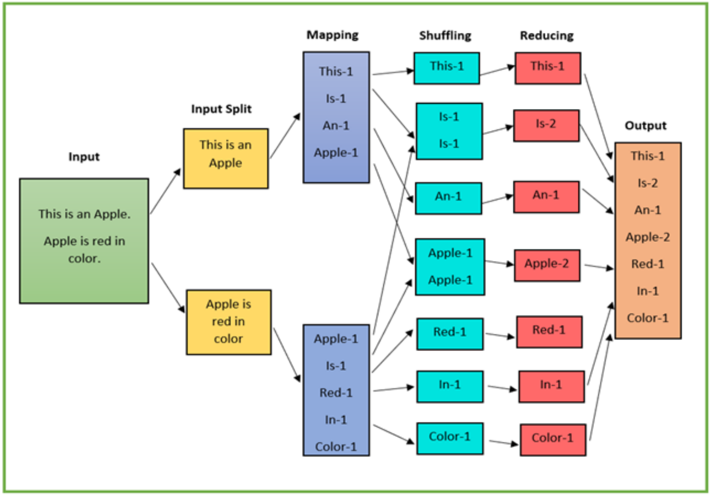


Figure 11: MapReduce Processing [1]

Big Data Technology: MapReduce

- It automatically schedules an application's execution, handles load balancing, node failures, and complex internode communication.
- The basic building blocks of a MapReduce application are two functions: **Map** and **Reduce**. Both primitives are borrowed from functional programming

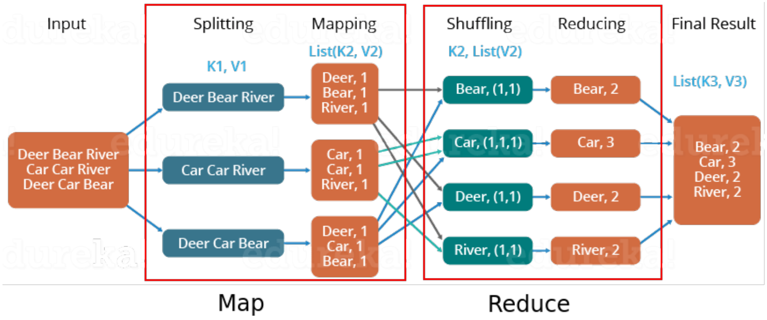


Figure 12: MapReduce [1]

[1] https://www.edureka.co/blog/mapreduce-tutorial/

Big Data Technology: MapReduce

- The map function takes as input a key-value pair and outputs a set of intermediate key-value pairs
- Next, it sorts the output from map functions and groups all intermediate values associated with the respective key.
- It then passes them as input to the reduce function, that aggregates those values and outputs them

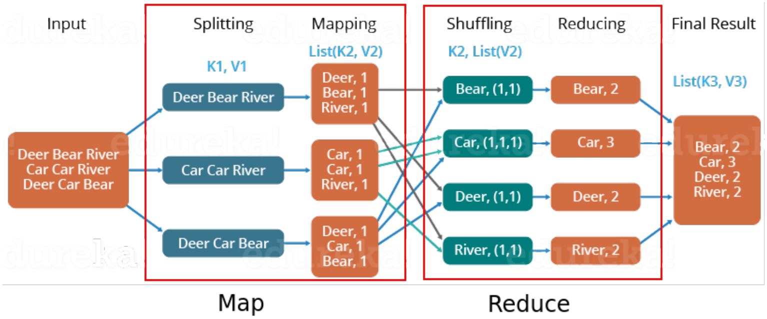


Figure 13: MapReduce Processing [1]

Big Data Technology: MapReduce

Spark, is considered a successor to MapReduce.

Stay tuned for Spark ...

Big Data Technology: Apache Hive

- Apache Hive is a data warehouse software
- It is built on top of Hadoop
- It provides a SQL-like query language **HiveQL** for processing and analyzing data stored in HDFS and other Hadoop-compatible storage systems, such as Cassandra and Amazon S3.
- Under the hood Hive translated HiveQL queries into MapReduce jobs
- Spark SQL is considered as successor to Hive

Big Data Technology: Data Serialization

- The format in which data is **stored on disk** or sent over a network is different from the format in which it **lives in memory**

Serialization

The process of converting data in memory to a format in which it can be stored on disk or sent over a network is called "serialization" [1]

Deserialization

The reverse process of reading data from disk or network into memory is called "deserialization" [1]

[1] Guller, M. (2015). "Big data analytics with Spark", Apress.

Big Data Technology: Data Serialization

- **Human-readable:** Not efficient in terms of storage and parsing

Feature	CSV	XML	JSON
Human-readable	✓	✓	✓
Schemaless	×	×	✓
Support for arrays	×	✓	✓
Support for nested structures	×	✓	✓
Row-oriented	✓	✓	×

- **Binary:** Compact and quicker parsing

Feature	Avro	Protobuf	Thrift
Cross platform	✓	✓	✓
Programming language neutral	✓	✓	✓
Schema based	✓	✓	✓
Support for unions	×	×	✓

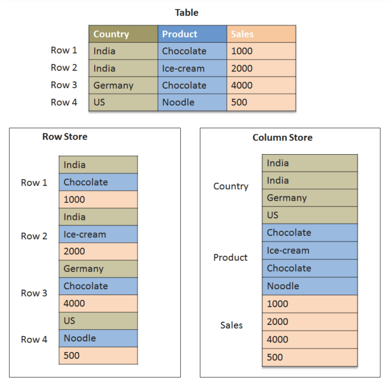
Big Data Technology: Columnar Storage

- Data can be stored in either a **row-oriented** or a **column-oriented** format
- Row-oriented storage is ideal for applications that mostly perform CRUD (create, read, update, delete) operations on data
- Row-oriented storage is not efficient for **analytics applications** most of the times, as they **operate on the columns** in a dataset, such that they read and analyze only a small subset of columns across multiple rows.
- Therefore, reading all columns is a waste of memory, CPU cycles, and disk I/O, which is an expensive operation.
- Another disadvantage of row-oriented storage is that data cannot be efficiently compressed

Big Data Technology: Columnar Storage

Row Based vs. Column-Based Storage

- In a big data scenarios, data is typically needed only from some specific columns
- So, why to parse the entire row ?
- Column-based storage saves a lot of disk and network I/O consumption
- Due to similar type of data (all String, or all Int), high data compression is achieved as compared to row-based storage
- more detail, Apache ORC [1]



Table

	Country	Product	Sales
Row 1	India	Chocolate	1000
Row 2	India	Ice-cream	2000
Row 3	Germany	Chocolate	4000
Row 4	US	Noodle	500

Figure 14: Row-Based vs. Column-Based [2]

[1] https://www.alibabacloud.com/blog/aliorc-a-combination-of-maxcompute-and-apache-orc

[2] https://orc.apache.org/specification/ORCv1/

Big Data Technology: Columnar Storage

Feature	**Row-Oriented**	**Column-Oriented**
Ideal Use Case	CRUD operations (Create, Read, Update, Delete)	Analytics applications
Efficiency for Analytics	Not efficient, as analytics often operate on columns across multiple rows	Efficient, as analytics focus on specific columns across all rows
Memory, CPU, Disk I/O	Inefficient use of resources when reading all columns	More efficient use of resources, only necessary columns are read
Compression	Inefficient, heterogeneous data types result in poor compression	More efficient, homogeneous data types allow for better compression
Disk Storage	Larger file size due to high entropy across rows	Smaller file size due to contiguous storage of columns

Big Data Technology: Columnar Storage

Feature	RCFile	ORC	Parquet
Compression	Supports configurable compression codecs like ZLIB, Snappy	Supports configurable compression codecs like ZLIB, Snappy, LZO	Uses advanced compression techniques like run-length encoding (RLE), dictionary encoding, and bit-packing
Predicate Pushdown	Limited support	Supports efficient predicate pushdown for filtering rows at read time	Efficiently pushes down filter predicates and column projection
Schema Evolution	Limited support for schema evolution	Better support for schema evolution with features like column addition and type evolution	Supports schema evolution with backward and forward compatibility
Indexing	No native indexing support	Supports indexes for faster data retrieval	Supports multiple types of indexes including Bloom filters, min/max indexes, and dictionary indexes
Query Performance	Moderate query performance	Improved query performance with predicate pushdown and indexing	Excellent query performance with efficient storage layout and advanced compression

RCFile (Record Columnar File), ORC (Optimized Row Columnar)

Big Data Technology: Messaging Systems

Producer

the application generating or sending data is referred to as a producer

Consumer

the application receiving the data is referred to as a consumer

- Sometimes there is an **asymmetry** between the number of applications producing data and the number of applications consuming that data
- e.g., one application may produce data, which gets consumed by multiple consumers. Similarly, one application may consume data from multiple producers

Big Data Technology: Messaging Systems

Producer

the application generating or sending data is referred to as a producer

Consumer

the application receiving the data is referred to as a consumer

- Also sometimes asymmetry between the **rate** at which one application produces data and the rate at which another application can consume it
- e.g., An application may produce data faster than the rate at which consumers can consume that data

Big Data Technology: Messaging Systems

Producer

the application generating or sending data is referred to as a producer

Consumer

the application receiving the data is referred to as a consumer

- A simple way to send data from one application to another is to connect them to each other directly
- However, this will not work if there is asymmetry either in the number of data producers and consumers or the rate at which data is produced and consumed
- A flexible and scalable solution is to use a **message broker or messaging system**

Big Data Technology: Messaging Systems

Producer

the application generating or sending data is referred to as a producer

Consumer

the application receiving the data is referred to as a consumer

- A flexible and scalable solution is to use a **message broker or messaging system**
- This architecture makes it easy to add producers or consumers to a data pipeline
- It also allows applications to produce and consume data at different rates

Big Data Technology: Kafka

- Kafka is a distributed messaging system or message broker
- It is a **distributed**, **partitioned**, **replicated** commit log service, which can be used as a **publish-subscribe** messaging system by multiple consumers

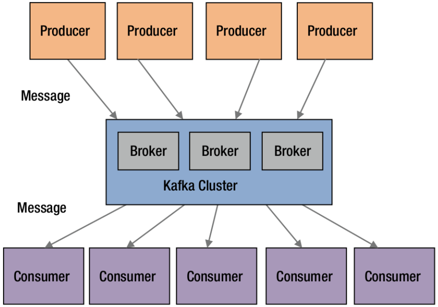


Figure 15: Kafka [1]

Big Data Technology: Kafka

- A single broker can handle several hundred megabytes of reads and writes per second from thousands of applications
- It can be easily scaled by adding more nodes to a cluster.
- For durability, it saves messages on disk
- Messages sent through Kafka are categorized into topics

Producer

An application that publishes messages to a Kafka topic

Consumer

An application that subscribes to a Kafka topic and processes messages

Big Data Technology: Kafka

- Kafka splits a topic into partitions
- Each partition is an ordered immutable sequence of messages
- New messages are appended to a partition
- Each message in a partition is assigned a unique sequential identifier called **offset**
- In addition, they are replicated for fault tolerance. Partitioning of topics helps with scalability and parallelism

Scala

What is Scala?

Scala is a modern (developed by Martin Odersky and released in 2004), multi-paradigm programming language designed to express common programming patterns in a concise, elegant, and type-safe way

Features:

- **Static Typing:** Scala is statically typed, which helps catch errors at compile-time rather than at run-time. Scala compiler enforces type safety at compile time. This helps reduce the number of bugs in an application
- **Object-Oriented:** Every value is an object and every operation is a method-call in Scala, similar to Java
- **Functional Programming:** Scala is also a functional language, supporting **first-class functions**, **immutability**, and **pattern matching**

Scala and Its Ecosystem

- **JVM Compatibility:** Scala runs on the Java Virtual Machine (JVM), allowing interoperability with Java and access to all Java libraries.
- **Concurrency & Distributed Computing:** It easier to build applications that are scalable and fault-tolerant ins Scala
- **Use Case:** Scala is frequently used in big data applications
- **Community and Adoption:** Scala has a strong community and is used by many companies worldwide for commercial applications and large-scale systems

Scala

Spark is written in Scala

But it supports multiple programming languages, including Scala, Java Python, and R

Scala

- Objective here is not to be an expert in Scala, but to be able learn enough Scala to understand and write Spark applications in Scala
- To effectively use Scala, it is important to know **functional programming**

Scala: Functional Programming

- Functional programming is a programming style that uses functions as a building block and avoids mutable variables, loops, and other imperative control structures
- It treats computation as an evaluation of mathematical functions, where the output of a function depends only on the arguments to the function.
- A program is composed of such functions.
- In addition, functions are first-class citizens in a functional programming language

Scala Multi-paradigm: Functional Programming

- Multi-paradigm languages are versatile, allowing developers to choose the best approach for a given problem
  - **Object-Oriented:** Organizes software design around data, or objects, rather than functions and logic. Objects represent data and methods to manipulaScala compiler enforces type safety at compile time. This helps reduce the number of bugs in an applicationte that data
  - **Functional:** Focuses on the evaluation of expressions rather than execution of commands. This paradigm emphasizes immutability (unchanging over time), higher-order functions (functions that take other functions as parameters or return them)
  - In addition, functions are first-class citizens in a functional programming language
- Scala integrates features from both object-oriented (like Java) and functional programming (like Haskell), allowing developers to use the most appropriate paradigm for their needs

Functional Programming Languages

- Besides Scala, there are several other programming languages that either fully support or have significant functional programming capabilities:
  - **Haskell:** Often considered the pure functional programming language due to its emphasis on immutability and pure functions.
  - **Erlang:** Used primarily for applications that require high availability, like telecoms and banking software
  - **F#:** A strongly typed, functional-first language that runs on the .NET framework.It supports both functional and object-oriented programming, making it a good choice for .NET developers looking to adopt functional techniques

Scala: Functions

- A function is a block of executable code
- In functional programming, an application is built entirely by assembling functions
- In functional programming languages, functions are first-class citizen.

Scala: Functions (First Class Citizens)

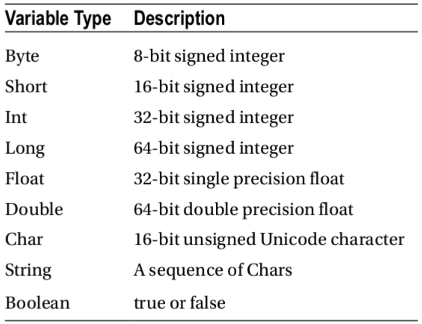
- A function has the same status as a variable or value
- It allows a function to be used just like a variable
- It is easier to understand this concept if you contrast FP functions with functions in imperative languages such as C
- Imperative languages treat variables and functions differently. For example, C does not allow a function to be defined inside another function. It does not allow a function to be passed as an input parameter to another function.

Scala: Functions (First Class Citizens)

In functional programming:

- a function to be passed as an input to another function
- It allows a function to be returned as a return value from another function
- A function can be defined anywhere, including inside another function
- It can be defined as an unnamed function literal just like a string literal and passed as an input to a function
- **Every statement is an expression** that returns a value. For example, the if-else control structure in Scala is an expression that returns a value.

Scala: Basic Scala Variable Types



Variable Type	Description
Byte	8-bit signed integer
Short	16-bit signed integer
Int	32-bit signed integer
Long	64-bit signed integer
Float	32-bit single precision float
Double	64-bit double precision float
Char	16-bit unsigned Unicode character
String	A sequence of Chars
Boolean	true or false

Figure 10: Basic Scala Variable Types [1]

[1] Guller, M. (2015). "Big data analytics with Spark", Apress.

Apache Spark

What is Apache Spark™?

Apache Spark is a unified analytics engine for large-scale data processing. [1]

It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. [1]

Apache Spark™ is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters.[2]

[1] https://spark.apache.org/docs/latest/index.html

[2] https://spark.apache.org/

Apache Spark^{TM} «Components and Libraries»

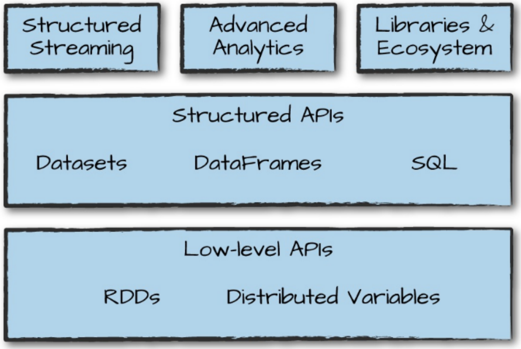


Figure 11: Spark Components & Libraries [1]

[1] Chambers, Bill, and Matei Zaharia. Spark: The definitive guide: Big data processing made simple. " O'Reilly Media, Inc.", 2018.

Conclusion

Conclusion

- We studied about Distributed Computing

Conclusion

- We studied about Distributed Computing
- We studied about Big Data and it various applications

Conclusion

- We studied about Distributed Computing
- We studied about Big Data and it various applications
- We performed the installation of Scala and learned deployments of Spark

Conclusion

- We studied about Distributed Computing
- We studied about Big Data and it various applications
- We performed the installation of Scala and learned deployments of Spark
- We learned basic about **Scala** and **Functional Programming**

Thank you !