Big Data Ecosystem

Target Complete Date	@December 2, 2022
Status	Completed
Reviewed	To review
Time to complete (Hours)	0.5
Type	Cloud Guru

💡 We are looking at the fundamentals of BigData, these are the frameworks and theories that underpin the Google Cloud services.

MapReduce

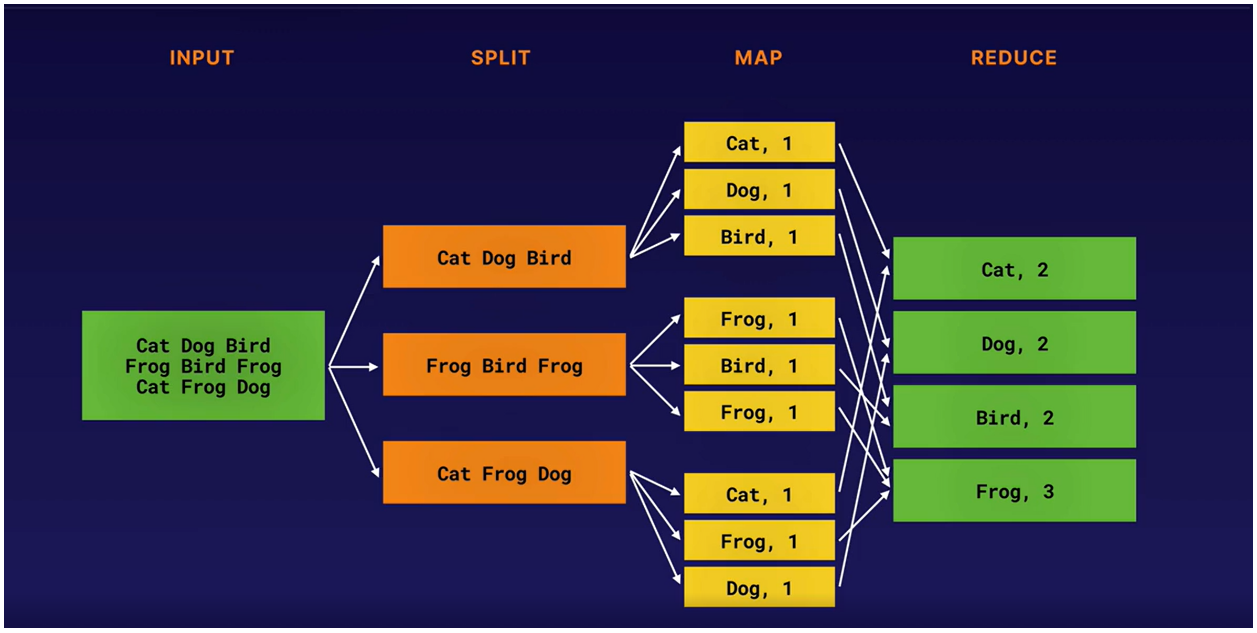
⚠️ The problem MapReduce solved is that companies would have massive jobs to be done across data centers over many machines. But they did not know how to structure the computation to easily be run across many machines. How to take advantage of many machines.

- Programming model
  - Map and Reduce functions
  - Combined to perform a task
- Distributed implementation of the model
- Created at Google
  - Abstracted away the distributed systems management work
- Standardized framework to design solutions for their data processing problems

- The model handle the parallelizing and executing across several machines
    - All cluster management was taken care of behind the scenes
    - Including partitioning, scheduling and fault tolerance of jobs
    - Handling networking and storage
- MapReduce jobs split into small chunks
- Master and worker cluster model
- Failed worker jobs reassigned
- Partitioned output files

High level view

💡 **The map function**, takes an input from a user and produces a set of intermediate key/value pairs. **The reduce function** merges intermediate values associated with the same intermediate key. Forms a smaller set of values.



Stages

MapReduce is done over three strages:

1. Map
    a. The map phase does not allow nor require communication across machines, to be completed properly
2. Shuffle
    a. Sort the keys so that the computers handling the reduce stage, get the same keys
3. Reduce
    a. Reduce will then take the shuffled keys to combine the results together

Hadoop & HDFS

💡 Hadoop is an open source file processing system and distributed storage system. It provides HDFS as a distributed file system that allows you to store large datasets across a cluster of commodity hardware. HDFS is designed to handle large files and is fault-tolerant, which means that data is replicated across multiple nodes in the cluster to ensure that it remains available in the event of hardware failures.

In addition, Hadoop provides MapReduce as a programming model and processing framework that allows you to process the data stored in HDFS in parallel across the cluster. MapReduce breaks down data processing tasks into smaller sub-tasks that can be processed in parallel across multiple nodes in the cluster. This allows large datasets to be processed quickly and efficiently, and is a key feature of Hadoop that makes it well-suited for big data processing.

Hadoop also provides a variety of other tools and technologies that extend its capabilities and make it more flexible and powerful. These include Spark, Hive, Pig, and HBase, which are all part of the Hadoop ecosystem and can be used for different data processing and analysis tasks.

⚠️ The Hadoop project was looking to be able to index the entire worldwide web. Too much data to be handled on a single machine.

Core modules

- Hadoop common
    - Base framework of Hadoop, all the libraries, system abstractions and startup systems required to run the rest of hadoop
- Hadoop distributed file system (HDFS) - Storage unit
    - Data is distributed among many computers and stored in blocks
        - HDFS splits the data among several blocks of data across several data nodes
    - Fault tolerant file system
        - HDFS makes copy of the data across several nodes and stores it accross multiple systems
- Hadoop MapReduce
    - Partitioning jobs across several machines
    - Reduction
- Hadoop YARN (Yet another resource negotiator)
    - Manages MapReduce jobs across a cluster of machines
    - Allows you to run multiple jobs across the cluster of nodes which efficiently allocates resources
        - Resource manager: Resource management, assigns resources
        - Node manager: Handles the nodes and monitor resource usage in the node
        - Application master: Requests containers from node manager
        - Container: Holds a collection of physical resources
    - Job scheduling

    - Processes job requests and manages cluster resources

- [ ] Look into master worker cluster architecture

Apache Pig

> 💡 Pig is a high level framework for running MapReduce jobs on Hadoop clusters.
> Platform for analyzing large datasets. Pig Latin defines analytics jobs. It is an abstraction for MapReduce. Handles ETL jobs. It is also a procedural job.

- Pig will compile jobs into Hadoop jobs to be executed
- It comes with Built-in functions that can perform transformations and aggregation functions
- You can also write your user-defined functions

Apache Spark

> 💡 Apache Spark was developed to solve the limitations presented by MapReduce implemented through the Hadoop framework.

Limitations of MapReduce

- Linear dataflow
    - Functions
        - Read data
        - Map functions across data
        - Reduce results
        - Write to disk
    - Made it harder to run complex calculations

Design

- It is designed to be a general purpose cluster-computing framework allowing for concurrent computational job to be run across massive datasets
- Uses concept of Resilient distributed data multisets
- Can be accessed as working sets as a form of **distributed shared memory**

Modules

- Spark SQL
    - Designed to work with structured data
- Spark streaming
    - Configure streaming data ingestion
    - Run Ad-hoc queries against current state of a stream
- MLLib: Machine learning library
    - Logistic regression much faster on Spark (100x)
- GraphX
    - Module designed for iterative graph computation

Support

- Python, java, scala and R

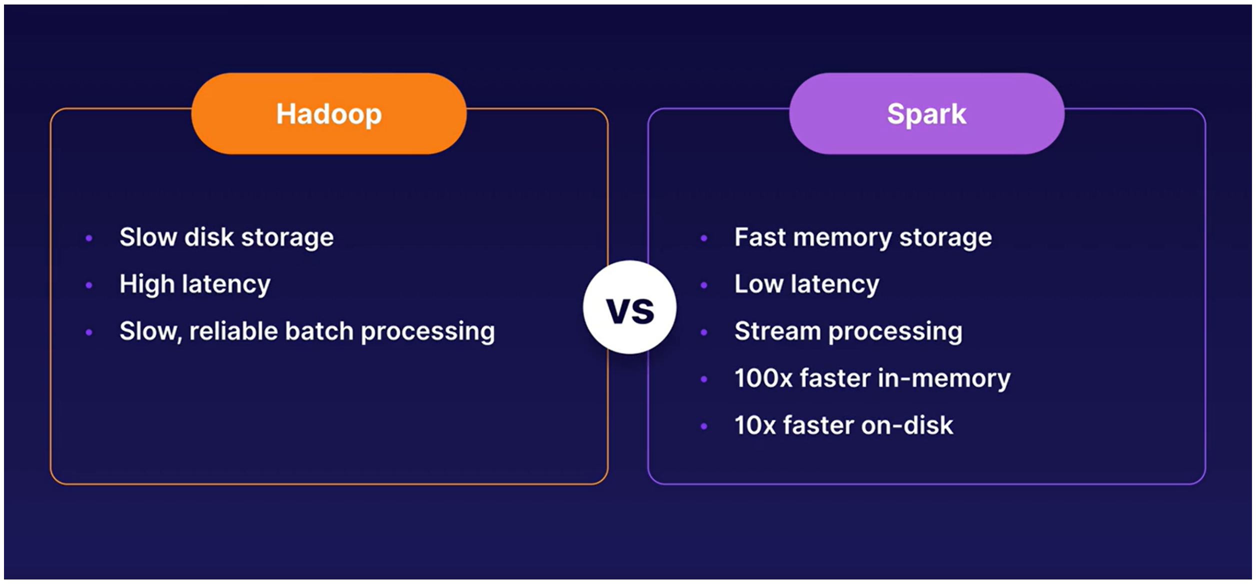
Features

To run apache spark you need two things:

1. A cluster manager
    a. Hadoop Yarn
    b. Kubernetes
2. Storage system
    a. HDFS
    b. Cassandra

Hadoop vs Spark

💡 The main difference is the way data is processed during computations



Hadoop

- Slow disk storage
- High latency
- Slow, reliable batch processing

VS

Spark

- Fast memory storage
- Low latency
- Stream processing
- 100x faster in-memory
- 10x faster on-disk

- [ ] Why is in memory so much faster than on disk

Apache Kafka

💡 A distributed streaming platform: A distributed system that allows you to pub/sub to streams of records.

- Like a message bus, but for data
- High throughput and low-latency
- Made by LinkedIn handling 800 billion messages per day

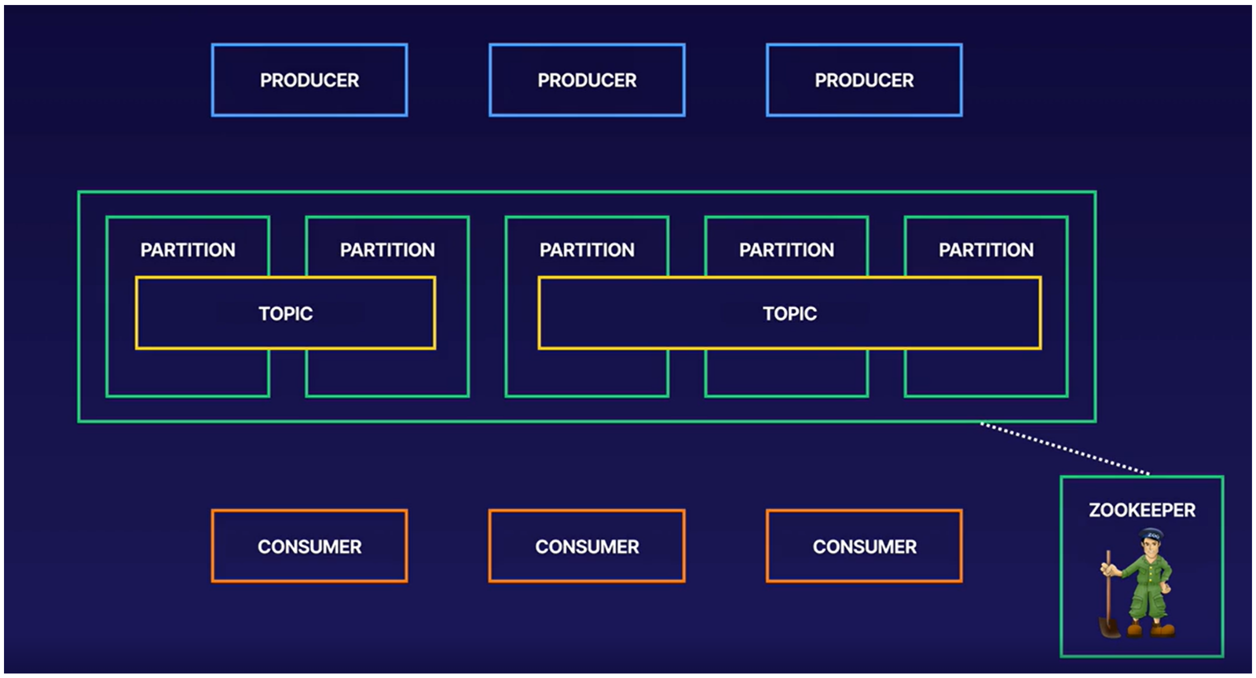
- [ ] What is a message bus

Kafka APIs

- Producer

    - Allows application to publish stream of records to Kafka topic
- Consumer
    - Subscribe to topics and process stream of records
- Streams
    - Allow application to act as stream processor itself
    - Useful when you want to transform stream data and feed it straight back to Kafka as outgoing stream
- Connector
    - Extend Kafka by connecting producers or consumer to external systems

Kafka Architecture



- Cloud Pub/Sub provides very similar functionality

Kafka vs Pub/Sub

Pub/Sub

- [ ] Return to this

> 💡 Lots of events are produced across different industries that produces data. Kafka manages and processes those events.

Producer

> 💡 A producer is an application that you write, that produces events to the cluster Kafka. Writes the events to a Kafka cluster. Once it receives the event Kafka sends an acknowledgment and the producer moves on.

- Producers and consumers are decoupled
  - Consumers don’t know who produced the information
  - You do not know anything about who is consuming topics
  - This decoupling is intentional
  - Slow consumer do not affect producers
  - Independent failure and independent evolution
- Producer write Data as messages
  - Can be written as Python
  - Rest Proxy

Cluster

Kafka brokers

> 💡 A broker is a server that has it’s own disk that runs Kafka on it. Many brokers working together acts as a Kafka cluster. Each broker comes with their own retention time to hold data. Their basic function are to manage partitions. Partitions are store locally on a broker disk.

- Every kafka cluster is composed of brokers
- Infrastructure managed by cloud
- Partitions are replicated to avoid data loss across brokers
  - There are leader and follower partitions
  - The follower partition will extract information from leader partition to keep up to date with leader

Consumer

> 💡 A program you write that reads the data from a Kafka cluster, it will transform the data, and will read data out. The app can feed dashboards, generate reports, or whatever you program it to do.

- Consumers pull messages from 1…n topics
- New inflowing messages are automatically retrieved
- Consumer offset
  - Keeps track of the last message read
  - is stored in special topic
- CLI tools exist to read from cluster

Zookeeper ensemble

> 💡 Kafka uses Zookeeper to manage consensus across many brokers. The might be deprecated soon. Distributed consensus manager for the cluster.

- Management of failure
  - Election of new leaders
- Cluster managed

- Store ACLs & secrets

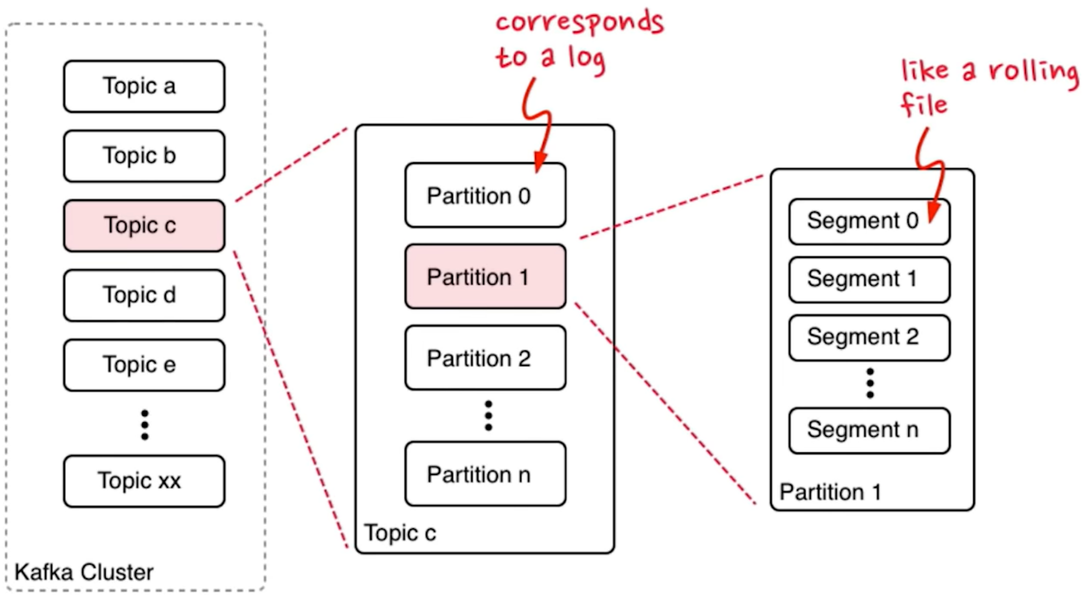
Topic

💡 A topic is a collection of related messages or of events. Think of it as a sequence of events. Such as a log. It is a logical representation that categorizes messages into groups of related messages.

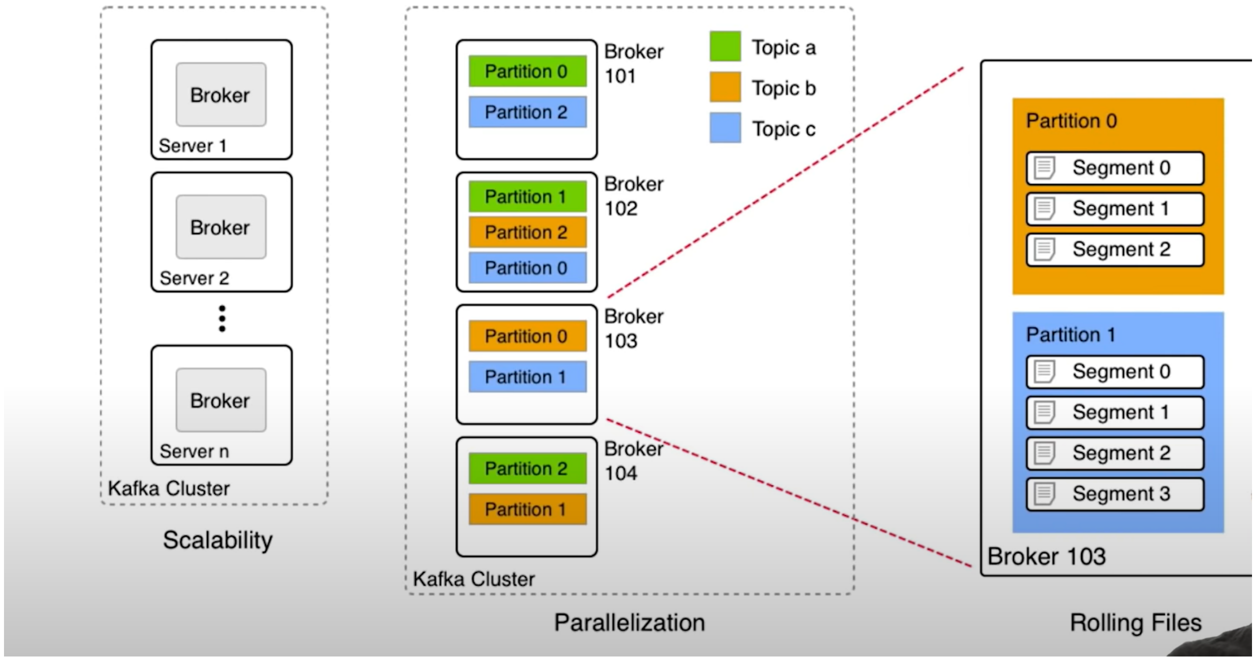
- There is no limit on the number of topics
- N to N relation between producer and topic

Partition

💡 Topics can be partitioned and allocated across several brokers so that it can scale properly. Within a partition you always have order.



- You need to think about partitioning when you are modelling your data



✏️ **Logs:** Immutable record of things. When you add data to a log it is appended to the end.

- There can be multiple consumers on one log consuming information, without destroying the information on a log.
- **This log will be referred to as a stream**

Kafka message

Structure

- Headers
  - String key value pairs as metadata
  - Can be seen on read
- Key
  - Business relevant data
- Value

  - Business relevant data
- Timestamp
  - Creation time or ingestion time
  - Can be overwritten

Load balancing and semantic partitioning

- Producers use a partitioning strategy to assign each message to a partition
- Two purposes:
  - Load balancing
  - Semantic partitioning
- Partitioning strategy specified by Producer
  - Default strategy:

```
hash(key) % number_of_partitions
```

  - This happens if there is a key
  - This results in the same key always being ordered in the same partition
  - Ensures they are strictly ordered all the time
- Custom partitioner possible