Big Data Engineering

Apache Spark Optimization Techniques and Performance Tuning

by Chandan Gaur | 04 October 2020

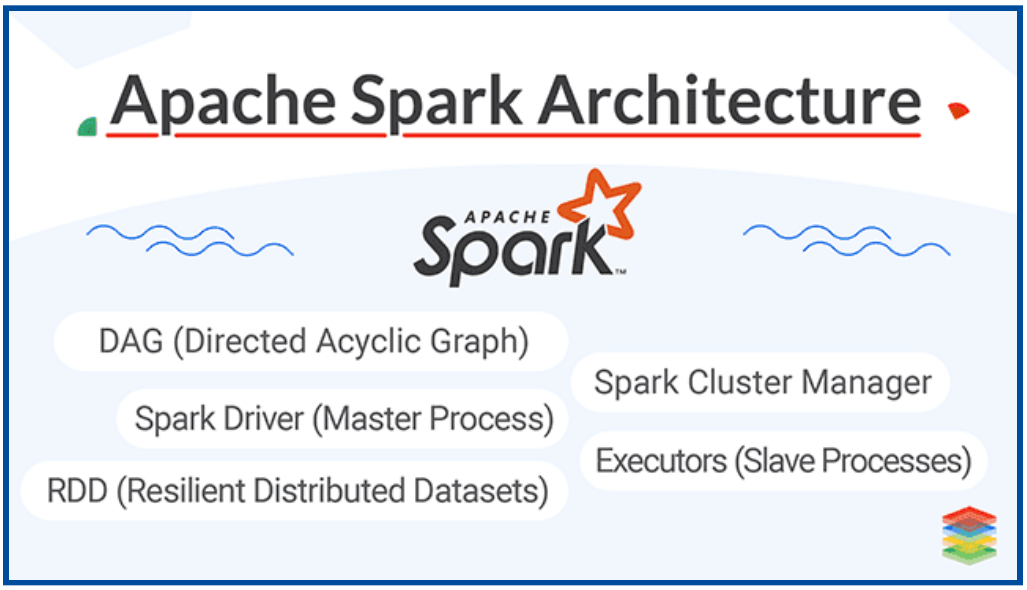


Table of Contents

What is Apache Spark?

Apache, in 2012, described the Resilient Distributed Dataset (RDD in Apache Spark) foundation with read-only Distributed datasets on distributed clusters and named it Apache Spark. Later, they introduce Dataset API and then Dataframe APIs for batch and structured streaming of data. This article lists out the best Apache Spark Optimization Techniques. Apache Spark is a fast cluster computing platform developed for performing more computations and stream processing. Spark can handle a wide variety of workloads compared to traditional systems that require multiple systems to run and support. Data analysis pipelines are facilitated by Spark in Combination with different processing types necessary for production. Apache Spark is created to operate with an external cluster manager such as YARN or its stand-alone manager.

Why Apache Spark Optimization?

We all know that during the development of any program, taking care of the performance is critical Apache Spark optimization helps with in-memory data computations. A Spark job can be optimized by many techniques so let's dig deeper into those techniques one by one.

Features of Apache Spark

Some features of Apache Spark include:-

- Unified Platform for writing big data applications.
- Ease of development.
- Designed to be highly accessible.
- Spark can run independently. Thus it gives flexibility.
- Cost Efficient.

Understanding How Apache Spark Optimization Works?

In order to understand how Apache Spark optimization works, you need to understand its architecture first and in the subsequent section, we will elaborate the same.

The architecture of Apache Spark

The Run-time architecture of Spark consists of three parts -

1. Spark Driver (Master Process)

The Spark Driver converts the programs into tasks and schedules the tasks for Executors. The Task Scheduler is the part of the Driver and helps to distribute tasks to Executors.

2. Spark Cluster Manager

A cluster manager is the core in Spark that allows launching executors, and sometimes drivers can be launched by it also. Spark Scheduler schedules the actions and jobs in Spark Application in FIFO way on cluster manager itself. You should also read about Apache Airflow .

3. Executors (Slave Processes)

Slave processes or Executors are the individual entities on which the individual task of the job runs. They will always run till the lifecycle of a spark Application once they are launched. Failed executors don't stop the execution of spark job.

4. RDD (Resilient Distributed Datasets)

An RDD is a distributed collection of immutable datasets on distributed nodes of the cluster. An RDD is partitioned into one or many partitions. RDD is the core of Spark as its distribution among various cluster nodes leverages data locality. To achieve parallelism inside the application, Partitions are the units for it. Repartition or coalesce transformations can help to maintain the number of partitions. Data access is optimized utilizing RDD shuffling. As Spark is close to data, it sends data across various nodes through it and creates required partitions as needed.

5. DAG (Directed Acyclic Graph)

Spark tends to generate an operator graph when we enter our code to the Spark console. When an action is triggered to Spark RDD, Spark submits that graph to the DAGScheduler. It then divides those operator graphs into stages of the task inside the DAGScheduler. Every step may contain jobs based on several partitions of the incoming data. The DAGScheduler

pipelines those individual operator graphs together. For Instance, Map operator graphs schedule for a single stage, and these stages pass on to the. Task Scheduler in cluster manager for their execution. This is the task of Work or Executors to execute these tasks on the slave.

6. Distributed processing using partitions efficiently

Increasing the number of Executors on clusters also increases parallelism in processing Spark Job. But for this, one must have adequate information about how that data would be distributed among those executors via partitioning. RDD is helpful for this case with negligible traffic for data shuffling across these executors. One can customize the partitioning for pair RDD (RDD with key-value Pairs). Spark assures that set of keys will always appear together in the same node because there is no explicit control in this case.

> Apache Spark security aids authentication through a shared secret. Spark authentication is the configuration parameter through which authentication can be configured. It is the parameter which checks whether the protocols of the spark communication are doing authentication using a shared secret or not.
>
> From the Article, Apache Spark Security

Mistakes to avoid while Optimising Apache Spark

1. reduceByKey or groupByKey

Both groupByKey and reduceByKey produce the same answer, but the concept to produce results is different. reduceByKey is best suitable for large datasets because, in Spark, it combines output with a shared key for each partition before shuffling of data. While on the other side, groupByKey shuffles all the key-value pairs. GroupByKey causes unnecessary shuffles and transfer of data over the network.

2. Maintain the required size of the shuffle blocks

By default, the Spark shuffle block cannot exceed 2GB. The better use is to

increase partitions and reduce its capacity to ~128MB per partition that will reduce the shuffle block size. We can use repartition or coalesce in regular applications. Large partitions make the process slow due to a limit of 2GB, and few partitions don't allow to scale the job and achieve parallelism.

3. File Formats and Delimiters

Choosing the right File formats for each data-related specification is a headache. One must choose wisely the data format for Ingestion types, Intermediate type, and Final output type. We can also Classify the data file formats for each type in several ways, such as we can use the AVRO file format for storing Media data as Avro is best optimized for binary data than Parquet. Parquet can be used for storing metadata information as it is highly compressed.

4. Small Data Files

Broadcasting is a technique to load small data files or datasets into Blocks of memory so that they can be joined with more massive data sets with less overhead of shuffling data. For Instance, We can store Small data files into n number of Blocks, and Large data files can be joined to these data Blocks in the future as Large data files can be distributed among these blocks in a parallel fashion.

5. No Monitoring of Job Stages

DAG is a data structure used in Spark that describes various stages of tasks in Graph format. Most of the developers write and execute the code, but monitoring of Job tasks is essential. This monitoring is best achieved by managing DAG and reducing the stages. The job with 20 steps is prolonged as compared to a job with 3-4 Stages.

6. ByKey, repartition or any other operations which trigger shuffles

Most of the time, we need to avoid shuffles as much as we can as data shuffles across many, and sometimes it becomes very complex to obtain Scalability out of those shuffles. GroupByKey can be a valuable asset, but its need must be described first.

7. Reinforcement Learning

Reinforcement Learning is not only the concept to obtain a better Machine learning environment but also to process decisions in a better way. One must apply deep reinforcement Learning in Spark if the transition model

and reward model are built correctly on data sets and also agents are capable enough to estimate the results.

Apache Spark Optimization Factors and Techniques

One of the best features of Apache Spark optimization is it helps with In-memory data computations. The bottleneck for these spark optimization computations can be CPU, memory, or any resource in the cluster. A need to serialize the data, reduce the memory may arise in such cases. These factors for spark optimization, if properly used, can -

- Eliminate the long-running job process
- Correction execution engine
- Improves performance time by managing resources

13 Simple Techniques for Apache Spark Optimization

1. Using Accumulators

Accumulators are global variables to the executors that can only be added through an associative and commutative operation. It can, therefore, be efficient in parallel. Accumulators can be used to implement counters (same as in Map Reduce) or another task such as tracking API calls. By default, Spark supports numeric accumulators, but programmers have the advantage of adding support for new types. Spark ensures that each task's update will only be applied once to the accumulator variables. During transformations, users should have an awareness of each task's update as these can be applied more than once if job stages are re-executed.

2. Hive Bucketing Performance

Bucketing results with a fixed number of files as we specify the number of buckets with a bucket. Hive took the field, calculate the hash and assign a record to that particular bucket. Bucketing is more stable when the field has high cardinality, Large Data Processing, and records are evenly distributed among all buckets whereas partitioning works when the cardinality of the partitioning field is low. Bucketing reduces the overhead of sorting files. For Instance, if we are joining two tables that have an equal number of buckets in it, spark joins the data directly as keys already sorted buckets. The number of bucket files can be calculated as several partitions into several buckets.

3. Predicate Pushdown Optimization

Predicate pushdown is a technique to process only the required data. Predicates can be applied to SparkSQL by defining filters in where conditions. By using explain command to query we can check the query processing stages. If the query plan contains PushedFilter than the query is optimized to select only required data as every predicate returns either True or False. If there is no PushedFilter found in query plan than better is to cast the where condition. Predicate Pushdowns limits the number of files and partitions that SparkSQL reads while querying, thus reducing disk I/O starts In-Memory Analytics. Querying on data in buckets with predicate pushdowns produce results faster with less shuffle.

4. Zero Data Serialization/Deserialization using Apache Arrow

Apache Arrow is used as an In-Memory run-time format for analytical query engines. It provides data serialization/deserialization zero shuffles through shared memory. Arrow flight sends the large datasets over the network. Additionally, it has its arrow file format that allows zero-copy random access to data on-disks. It has a standard data access layer for all spark applications. It reduces the overhead for SerDe operations for shuffling data as it has a common place where all data is residing and in arrow specific format.

5. Garbage Collection Tuning using G1GC Collection

When tuning garbage collectors, we first recommend using G1 GC to run Spark applications. The G1 garbage collector entirely handles growing heaps that are commonly seen with Spark. With G1, fewer options will be needed to provide both higher throughput and lower latency. To control unpredictable characteristics and behaviours of various applications GC tuning needs to be mastered according to generated logs. Before this, other optimization techniques like Streaming and Real-Time Analytics Solutions, in the program's logic and code must be applied. Most of the time, G1GC helps to optimize the pause time between processes that are quite often in Spark applications, thus decreases the Job execution time with a more reliable system.

6. Memory Management and Tuning

As we know that, for computations such as shuffling, sorting and so on, Execution memory is used whereas for caching purposes storage memory is used that also propagates internal data. There might be some cases

where jobs are not using any cache; therefore, cases out of space error during execution. Cached jobs always apply less storage space where the data is not allowed to be evicted by any execution requirement. In addition, Real-Time Streaming Application with Apache Spark can be done. We can set spark.memory.fraction to determine how much JVM heap space is used for Spark execution memory. Commonly, 60% is the default. Executor memory must be kept as less as possible because it may lead to delay of JVM Garbage collection. This fact is also applicable for small executors as multiple tasks may run on a single JVM instance.

7. Data Locality

In Apache Spark, Processing tasks are optimized by placing the execution code close to the processed data, called data locality. Sometimes processing task has to wait before getting data because data is not available. However, when the time of spark.locality.wait expires, Spark tries less local level, i.e., Local to the node to rack to any. Transferring data between disks is very costly, so most of the operations must be performed at the place where data resides. It helps to load only small but required the amount of data along with test-driven development for Apache Spark.

8. Using Collocated Joins

Collocated joins make decisions of redistribution and broadcasting. We can define small datasets to be located into multiple blocks of memory for achieving better use of Broadcasting. While applying joins on two datasets, spark First sort the data of both datasets by key and them merge. But, we can also apply sort partition key before joining them or while creating those data frames INApache Arrow Architecture. This will optimize the run-time of the query as there would be no unnecessary function calls to sort.

9. Caching in Spark

Caching in Apache Spark with GPU is the best technique for Apache Spark Optimization when we need some data again and again. But it is always not acceptable to cache data. We have to use cache () RDD and DataFrames in the following cases -

- When there is an iterative loop such as in Machine learning algorithms.
- RDD is accessed multiple times in a single job or task.
- Or, the cost to generate the RDD partitions again is higher.<l/i>

Cache () and persist (StorageLevel.MEMORY_ONLY) can be used in place of each other. Every RDD partition which gets evicted out of the memory is

required to be build again from the source that still is very expensive. One of the best solutions is to use persist (Storage level.MEMORY_AND_DISK_ONLY) that would spill the partitions of RDD to the Worker's local disk. This case only requires getting data from the Worker's local drive which is relatively fast.

10. Executor Size

When we run executors with high memory, it often results in excessive delays in garbage collection. We need to keep the cores count per executor below five tasks per executor. Too small executors didn't come out be handy in terms of running multiple jobs on single JVM. For Instance,

broadcast variables must be replicated for each executor exactly once, that will result in more copies of the data.

11. Spark Windowing Function

A window function defines a frame through which we can calculate input rows of a table. On individual row level. Each row can have a clear framework. Windowing allows us to define a window for data in the data frame. We can compare multiple rows in the same data frame. We can set the window time to a particular interval that will solve the issue of data dependency with previous data. Shuffling in Apache Beam is less on previously processed data as we are retaining that data for window interval.

12. Watermarks Technique

Watermarking is a useful technique in Apache Spark Optimization that constrains the system by design and helps to prevent it from exploding during the run. Watermark takes two arguments -

- Column for event time and
- A threshold time that specify for how long we are required to process late data

The query in Apache Arrow Architecture will automatically get updated if data fall within that stipulated threshold; otherwise, no processing is triggered for that delayed data. One must remember that we can use Complete-mode side by side with watermarking because full mode first persists all the data to the resulting table.

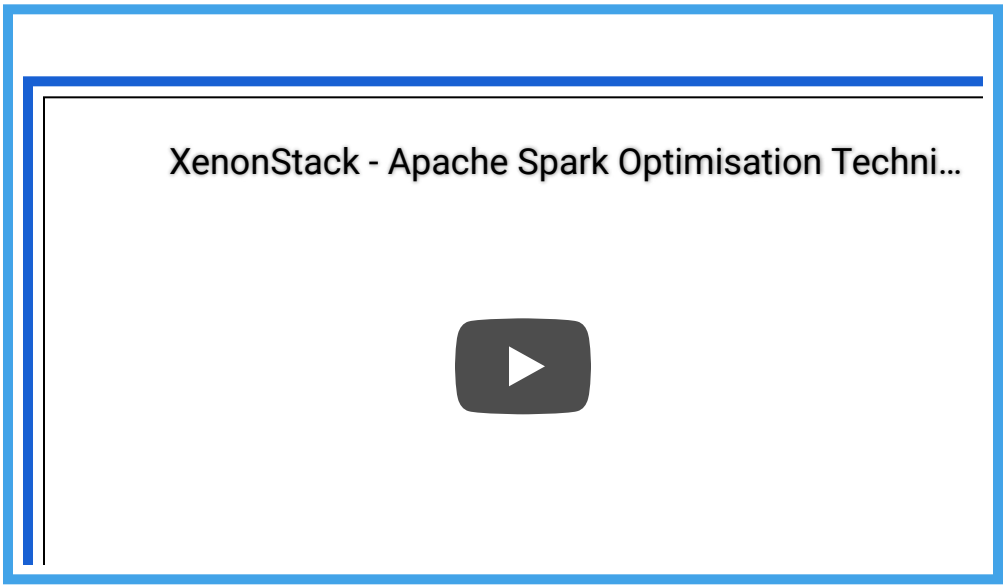
13. Data Serialization

Apache Spark optimization works on data that we need to process for

some use cases such as Analytics or just for movement of data. This movement of data or Analytics can be well performed if data is in some better-serialized format. Apache Spark supports Data serialization to manage the data formats needed at Source or Destination effectively. By Default, Apache Spark uses Java Serialization but also supports Kryo Serialization. By default, Spark uses Java's ObjectOutputStream to serialize the data. The implementation can be through the java.io.Serializable class. It encodes the objects into a stream of bytes. It provides lightweight persistence and flexible. But it becomes slow as it leads to huge serialized formats for each class it is used in. Spark supports Kryo Serialization

library (v4) for Serialization of objects nearly 10x faster than Java Serialization as it is more compact than Java.

Apache Spark Optimization: A Comprehensive Approach (Video)



Apache Spark, an open-source distributed computing engine, is currently the most popular framework for in-memory batch processing, which also supports real-time streaming. With its advanced query optimizer and execution engine, Apache Spark Optimisation Techniques can process and analyze large datasets very efficiently. However, running Apache Spark Join Optimization techniques without careful tuning can degrade performance. If you want to harness your Apache Spark Application power, then check out our Managed Apache Spark Services.

First Name*

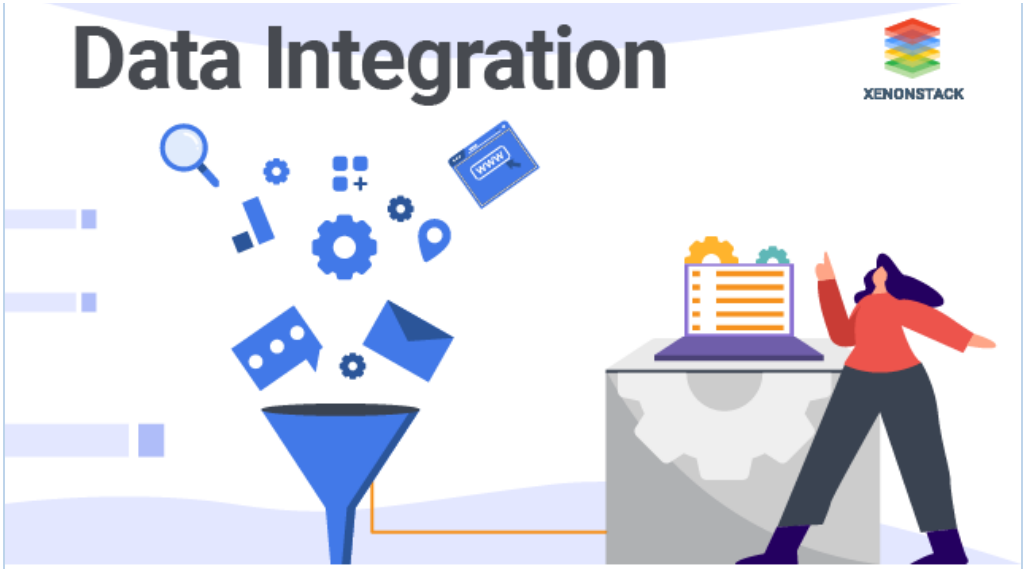
Last Name

Email*

Comment*

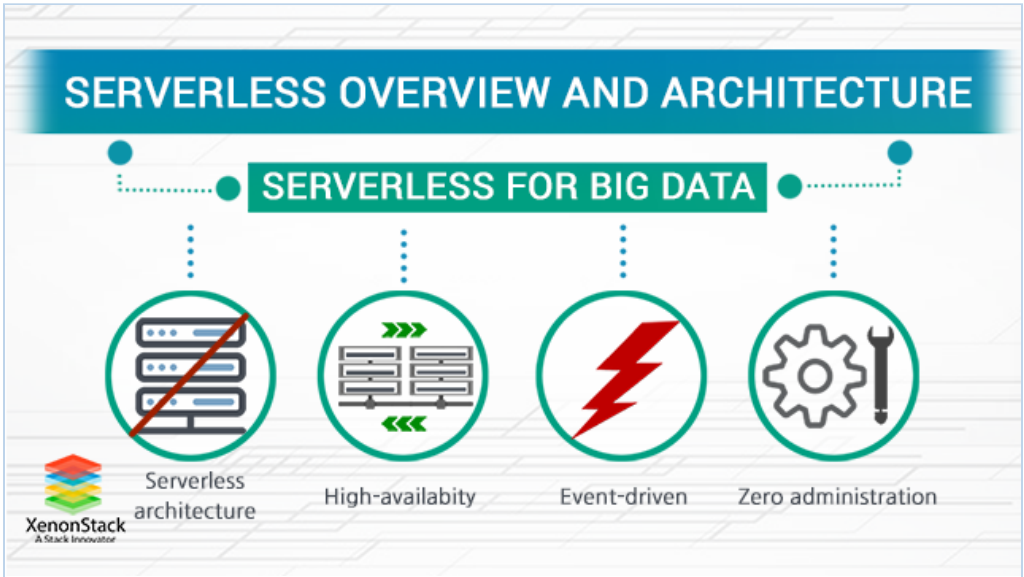
Submit Comment

Related blogs and Articles



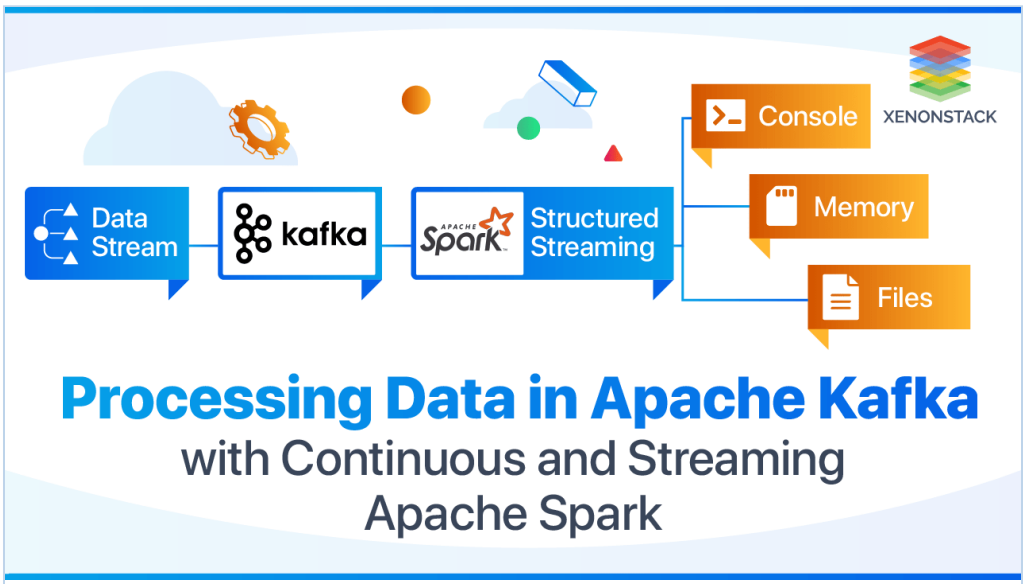
BIG DATA ENGINEERING

What is Data Integration ? Benefits | Tools | Challenges



BIG DATA ENGINEERING

Serverless Solutions and Architecture for Big Data, and Data Lake



BIG DATA ENGINEERING

Real Time Streaming Application with Apache Spark

Apache Spark Overview Apache Spark is a fast, in-memory data processing engine with expressive development APIs to allow data workers to execute streaming conveniently. With Spark running on Apache Hadoop YARN, developers everywhere can now create applications to exploit Spark's power, derive insights, and enrich their data science workloads within a single, shared dataset in Apache Hadoop. In...

23 May 2021

Fresh news directly to your mail box

Email*

Submit

Be A Neural Company

Know More

We are members of :

Partnerships & Certifications :

XENONSTACK

Why Xenonstack

About us

Services

Solutions

Careers

Neural Enablers

Human Interaction

Modular Driven

API Driven

Platform Strategy

Data and AI

Technology Consulting

Cloud Strategy

Big Data Strategy

Devops

AI

Solutions

GitOps with Continuous Delivery

Knowledge Graph and Graph Analytics

Data Catalog and Discovery Platform

Augmented Data Quality and Management

Serverless Data Mesh Platform

Video Analytics

Time Series Analytics

Reach Us

Managed Analytics Services

SRE Managed Services

Managed AI Cloud

On-Premises AI Cluster

Connect

Products

ElixirData

Nexastack

Akirastack

Akira AI