Home (/) About (https://sparkbyexamples.com/about-sparkbyexamples/) | { One stop for all Spark Examples }

Spark by {Examples} (https://sparkbyexamples.com/)

Spark (https://sparkbyexamples.com/)
PySpark (https://sparkbyexamples.com/pyspark-tutorial/)
Hive (https://sparkbyexamples.com/apache-hive-tutorial/)
HBase (https://sparkbyexamples.com/apache-hbase-tutorial/)
Kafka (https://sparkbyexamples.com/apache-kafka-tutorials-with-examples/)
[FAQ's] (https://sparkbyexamples.com/spark-questions/)
More (https://sparkbyexamples.com/)

Spark Performance Tuning & Best Practices

NNK (https://sparkbyexamples.com/author/admin/) - Apache Spark (https://sparkbyexamples.com/category/spark/) / PySpark (https://sparkbyexamples.com/category/pyspark/)

Spark Performance tuning is a process to improve the performance of the Spark and PySpark applications by adjusting and optimizing system resources (CPU cores and memory), tuning some configurations, and following some framework guidelines and best practices.

Spark RDD – Read text file (https://sparkbyexamples.com/apache-spark-rdd/spark-read-multiple-text-files-into-a-single-rdd/)

Spark RDD – Read CSV (https://sparkbyexamples.com/apache-spark-rdd/spark-load-csv-file-into-rdd/)

Spark RDD – Create RDD (https://sparkbyexamples.com/apache-spark-rdd/different-ways-to-create-spark-rdd/)

Spark RDD – Create Empty RDD (https://sparkbyexamples.com/apache-spark-rdd/spark-how-to-create-an-empty-rdd/)

Spark RDD – Transformations (https://sparkbyexamples.com/apache-spark-rdd/spark-rdd-transformations/)

Spark RDD – Actions (https://sparkbyexamples.com/apache-spark-rdd/spark-rdd-actions/)

Spark RDD – Pair Functions (https://sparkbyexamples.com/apache-spark-rdd/spark-pair-rdd-functions/)

Spark RDD – Repartition and Coalesce (https://sparkbyexamples.com/spark/spark-repartition-vs-coalesce/)

Spark RDD – Shuffle Partitions (https://sparkbyexamples.com/spark/spark-shuffle-partitions/)

Spark RDD – Cache vs Persist (https://sparkbyexamples.com/spark/spark-difference-between-cache-and-persist/)

Spark RDD – Persistance Storage Levels (https://sparkbyexamples.com/spark/spark-persistence-storage-levels/)

Spark application performance can be improved in several ways. Below are the different articles I've written to cover these.

- Spark Guidelines and Best Practices (Covered in this article)
- Tuning System Resources (executors, CPU cores, memory) – In progress
- Tuning Spark Configurations (https://sparkbyexamples.com/spark/spark-sql-performance-tuning-configurations/) (AQE, Partitions e.t.c)

In this article, I have covered some of the framework guidelines and best practices to follow while developing Spark applications which ideally improves the performance of the application, most of these best practices would be the same for both Spark with Scala or PySpark (Python).

Spark Performance Tuning – Best Guidelines & Practices

Spark performance tuning and optimization is a bigger topic which consists of several techniques, and configurations (resources memory & cores), here I've covered some of the best guidelines I've used to improve my workloads and I will keep updating this as I come acrossnew ways.

[Spark RDD – Broadcast Variables (https://sparkbyexamples.com/spark/spark-broadcast-variables/)]

[Spark RDD – Accumulator Variables (https://sparkbyexamples.com/spark/spark-accumulators/)]

[Spark RDD – Convert RDD to DataFrame (https://sparkbyexamples.com/apache-spark-rdd/convert-spark-rdd-to-dataframe-dataset/)]

Spark SQL Tutorial

[Spark SQL – Create DataFrame (https://sparkbyexamples.com/spark/different-ways-to-create-a-spark-dataframe/)]

[Spark SQL – Select Columns (https://sparkbyexamples.com/spark/spark-select-columns-from-dataframe/)]

[Spark SQL – Add and Update Column (withColumn) (https://sparkbyexamples.com/spark/spark-dataframe-withcolumn/)]

[Spark SQL – Rename Nested Column (https://sparkbyexamples.com/spark/rename-a-column-on-spark-dataframes/)]

[Spark SQL – Drop column (https://sparkbyexamples.com/spark/spark-drop-column-from-dataframe-dataset/)]

[Spark SQL – Where | Filter (https://sparkbyexamples.com/spark/spark-dataframe-where-filter/)]

[Spark SQL – When Otherwise (https://sparkbyexamples.com/spark/spark-case-when-otherwise-example/)]

1. Use DataFrame/Dataset over RDD

For Spark jobs, prefer using Dataset/DataFrame over RDD as Dataset and DataFrame's includes several optimization modules to improve the performance of the Spark workloads. In PySpark use, DataFrame over RDD as Dataset's are not supported in PySpark applications.

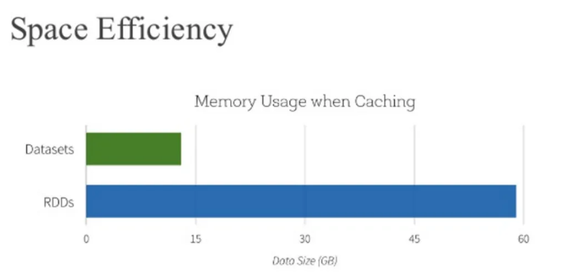
Spark RDD is a building block of Spark programming, even when we use DataFrame/Dataset, Spark internally uses RDD to execute operations/queries but the efficient and optimized way by analyzing your query and creating the execution plan thanks to Project Tungsten (https://databricks.com/blog/2015/04/28/

project-tungsten-bringing-spark-closer-to-bare-metal.html) and Catalyst optimizer (https://databricks.com/glossary/catalyst-optimizer).

Why RDD is slow?

Using RDD directly leads to performance issues as Spark doesn't know how to apply the optimization techniques and RDD serialize and de-serialize the data when it distributes across a cluster (repartition & shuffling).

Serialization and de-serialization are very expensive operations for Spark applications or any distributed systems, most of our time is spent only on serialization of data rather than executing the operations hence try to avoid using RDD.



Source: Databricks

Is DataFrame Faster

Since Spark DataFrame maintains the structure of the data and column types (like an RDMS table) it can handle the data better by storing and managing more efficiently.

The DataFrame API does two things that help to do this (through the Tungsten project). First, using off-heap storage for data in binary format. Second, generating encoder code on the fly to work with this binary format for your specific objects.

Spark SQL – Collect data to Driver (https://sparkbyexamples.com/spark/spark-dataframe-collect/)

Spark SQL – Distinct (https://sparkbyexamples.com/spark/spark-remove-duplicate-rows/)

Spark SQL- Pivot Table DataFrame (https://sparkbyexamples.com/spark/how-to-pivot-table-and-unpivot-a-spark-dataframe/)

Spark SQL – Data Types (https://sparkbyexamples.com/spark/spark-sql-dataframe-data-types/)

Spark SQL – StructType | StructField (https://sparkbyexamples.com/spark/spark-sql-structtype-on-dataframe/)

Spark SQL – Schema (https://sparkbyexamples.com/spark/spark-schema-explained-with-examples/)

Spark SQL – Groupby (https://sparkbyexamples.com/spark/using-groupby-on-dataframe/)

Spark SQL – Sort DataFrame (https://sparkbyexamples.com/spark/spark-how-to-sort-dataframe-column-explained/)

Spark SQL – Join Types (https://sparkbyexamples.com/spark/spark-sql-dataframe-join/)

Spark SQL – Union and UnionAll (https://sparkbyexamples.com/spark/spark-dataframe-union-and-union-all/)

Spark SQL – map() vs mapPartitions() (https://sparkbyexamples.com/spark/spark-map-vs-mappartitions-transformation/)

[Spark SQL – foreach() vs foreachPartition() (https://sparkbyexamples.com/spark/spark-foreachpartition-vs-foreach-explained/)]

[Spark SQL – map() vs flatMap() (https://sparkbyexamples.com/spark/spark-map-vs-flatmap-with-examples/)]

[Spark SQL – Persist and Cache (https://sparkbyexamples.com/spark/spark-dataframe-cache-and-persist-explained/)]

[Spark SQL – UDF (User Defined Functions) (https://sparkbyexamples.com/spark/spark-sql-udf/)]

[Spark SQL – Array (ArrayType) Column (https://sparkbyexamples.com/spark/spark-array-arraytype-dataframe-column/)]

[Spark SQL – Map (MapType) column (https://sparkbyexamples.com/spark/spark-dataframe-map-maptype-column/)]

[Spark SQL – Flatten Nested Struct Column (https://sparkbyexamples.com/spark/spark-flatten-nested-struct-column/)]

[Spark SQL – Flatten Nested Array Column (https://sparkbyexamples.com/spark/spark-flatten-nested-array-column-to-single-column/)]

[Spark SQL – Explode Array & Map Columns (https://sparkbyexamples.com/spark/explode-spark-array-and-map-dataframe-column/)]

[Spark SQL – Sampling (https://sparkbyexamples.com/spark/spark-sampling-with-examples/)]

Bachelor of Techn
B.Tech at #1 University in K
(New Universities Under 5 Years)
PES UNIVERSITY
AD
CL

Since Spark/PySpark DataFrame internally stores data in binary there is no need of Serialization and deserialization data when it distributes across a cluster hence you would see a performance improvement.

Project Tungsten

Spark Dataset/DataFrame includes Project Tungsten (https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html) which optimizes Spark jobs for **Memory and CPU efficiency**. Tungsten is a Spark SQL component that provides increased performance by rewriting Spark operations in bytecode, at runtime. Tungsten performance by focusing on jobs close to bare metal CPU and memory efficiency.

Since DataFrame is a column format that contains additional metadata, hence Spark can perform certain optimizations on a query. Before your query is run, a logical plan is created using Catalyst Optimizer (https://databricks.com/glossary/catalyst-optimizer) and then it's executed using the Tungsten execution engine.

What is Catalyst?

Catalyst Optimizer (https://databricks.com/glossary/catalyst-optimizer) is an integrated query optimizer and execution scheduler for Spark Datasets/DataFrame. Catalyst Optimizer is the place where Spark tends to improve the speed of your code execution by logically improving it.

Catalyst Optimizer can perform refactoring complex queries and decides the order of your query execution by creating a rule-based and code-based optimization.

Additionally, if you want type safety at compile time prefer using Dataset. For example, if you refer to a field that doesn't exist in your code, Dataset generates compile-time error whereas DataFrame compiles fine but returns an error during run-time.

2. Use coalesce() over repartition()

When you want to reduce the number of partitions prefer using `coalesce()` as it is an optimized or improved version of `repartition()` (https://sparkbyexamples.com/spark/spark-repartition-vs-coalesce/) where **the movement of the data across the partitions is lower using coalesce** which ideally performs better when you dealing with bigger datasets.

Note: Use repartition() when you wanted to increase the number of partitions.

Example repartition()

spark/spark-read-and-write-json-file/)

Spark – Read & Write Parquet file (https://sparkbyexamples.com/spark/spark-read-write-dataframe-parquet-example/)

Spark – Read & Write XML file (https://sparkbyexamples.com/spark/spark-read-write-xml/)

Spark – Read & Write Avro files (https://sparkbyexamples.com/spark/read-write-avro-file-spark-dataframe/)

Spark – Read & Write Avro files (Spark version 2.3.x or earlier) (https://sparkbyexamples.com/spark/using-avro-data-files-from-spark-sql-2-3-x/)

Spark – Read & Write HBase using "hbase-spark" Connector (https://sparkbyexamples.com/spark/spark-read-write-using-hbase-spark-connector/)

Spark – Read & Write from HBase using Hortonworks (https://sparkbyexamples.com/spark/create-spark-dataframe-from-hbase-using-hortonworks/)

Spark – Read & Write ORC file (https://sparkbyexamples.com/spark/spark-read-orc-file-into-dataframe/)

Spark – Read Binary File (https://sparkbyexamples.com/spark/spark-read-binary-file-into-dataframe/)

Spark Streaming & Kafka

Spark Streaming – OutputModes (https://sparkbyexamples.com/spark/spark-streaming-outputmode/)

```
val rdd1 = spark.sparkContext.pa
  println("parallelize : "+rdd1.

val rdd2 = rdd1.repartition(4)
  println("Repartition size : "+
  rdd2.saveAsTextFile("/tmp/re-p
```

This yields output `Repartition size : 4` and the repartition re-distributes the data(as shown below) from all partitions which is full shuffle leading to very expensive operation when dealing with billions and trillions of data. By tuning the partition size to optimal, you can improve the performance of the Spark application

```
Partition 1 : 1 6 10 15 19
Partition 2 : 2 3 7 11 16
Partition 3 : 4 8 12 13 17
Partition 4 : 0 5 9 14 18
```

Example coalesce()

```
val rdd3 = rdd1.coalesce(4)
  println("Repartition size : "+
  rdd3.saveAsTextFile("/tmp/coal
```

If you compared the below output with section 1, you will notice partition 3 has been moved to 2 and Partition 6 has moved to 5, resulting data movement from just 2 partitions.

```
Partition 1 : 0 1 2
Partition 2 : 3 4 5 6 7 8 9
Partition 4 : 10 11 12
Partition 5 : 13 14 15 16 17 18
```

3. Use mapPartitions() over map()

Spark `map()` and `mapPartitions()` transformation applies the function on each element/record/row of the DataFrame/Dataset and returns the new DataFrame/Dataset. mapPartitions() over map() prefovides performance improvement (https://sparkbyexamples.com/spark/spark-map-vs-mappartitions-transformation/) when you have havy initializations like initializing classes, database connections e.t.c

Spark `mapPartitions()` provides a facility to do heavy initializations (for example Database connection) once for each partition instead of doing it on every DataFrame row. This helps the performance of the Spark jobs when you dealing with heavy-weighted initialization on larger datasets.

Example map()

```
import spark.implicits._
  val df3 = df2.map(row=>{
    val util = new Util() // Ini
    val fullName = util.combine(
    (fullName, row.getString(3),
  })
  val df3Map =  df3.toDF("fullNa
```

Example mapPartitions()

[PySpark – Install on Windows (https://sparkbyexamples.com/pyspark-tutorial/#pyspark-installation)]
[PySpark – Web/Application UI (https://sparkbyexamples.com/spark/spark-web-ui-understanding/)]
[PySpark – SparkSession (https://sparkbyexamples.com/pyspark/pyspark-what-is-sparksession/)]
[PySpark – RDD (https://sparkbyexamples.com/pyspark-rdd)]
[PySpark – Parallelize (https://sparkbyexamples.com/pyspark/pyspark-parallelize-create-rdd/)]
[PySpark – repartition() vs coalesce() (https://sparkbyexamples.com/pyspark/pyspark-repartition-vs-coalesce/)]
[PySpark – Broadcast Variables (https://sparkbyexamples.com/pyspark/pyspark-broadcast-variables/)]
[PySpark – Accumulator (https://sparkbyexamples.com/pyspark/pyspark-accumulator-with-example/)]

PySpark DataFrame

[PySpark – Create a DataFrame (https://sparkbyexamples.com/pyspark/different-ways-to-create-dataframe-in-pyspark/)]
[PySpark – Create an empty DataFrame (https://sparkbyexamples.com/pyspark/pyspark-create-an-empty-dataframe/)]
[PySpark – Convert RDD to DataFrame (https://sparkbyexamples.com/pyspark/convert-pyspark-rdd-to-dataframe/)]

```
  val df4 = df2.mapPartitions(it
    val util = new Util()
    val res = iterator.map(row=>
      val fullName = util.combin
      (fullName, row.getString(3
    })
    res
  })
  val df4part = df4.toDF("fullNa
```

Note: One key point to remember is these both transformations returns the `Dataset[U]` but not the `DataFrame` (In Spark 2.0, `DataFrame = Dataset[Row]`).

4. Use Serialized data format's

Most of the Spark jobs run as a pipeline where one Spark job writes data into a File and another Spark jobs read the data, process it, and writes to another file for another Spark job to pick up. When you have such use case, prefer writing an intermediate file in Serialized and optimized formats like Avro, Kryo, Parquet e.t.c, any transformations on these formats performs better than text, CSV (https://sparkbyexamples.com/spark/spark-read-csv-file-into-dataframe/), and JSON (https://sparkbyexamples.com/spark/spark-read-and-write-json-file/).

What is Parquet

Apache Parquet is a columnar file format that provides optimizations (https://sparkbyexamples.com/spark/spark-read-write-dataframe-parquet-example/) to speed up queries and is a

PySpark – Convert DataFrame to Pandas (https://sparkbyexamples.com/pyspark/convert-pyspark-dataframe-to-pandas/)

PySpark – show() (https://sparkbyexamples.com/pyspark/pyspark-show-display-dataframe-contents-in-table/)

PySpark – StructType & StructField (https://sparkbyexamples.com/pyspark/pyspark-structtype-and-structfield/)

PySpark – Row Class (https://sparkbyexamples.com/pyspark/pyspark-row-using-rdd-dataframe/)

PySpark – Column Class (https://sparkbyexamples.com/pyspark/pyspark-column-functions/)

PySpark – select() (https://sparkbyexamples.com/pyspark/select-columns-from-pyspark-dataframe/)

PySpark – collect() (https://sparkbyexamples.com/pyspark/pyspark-collect/)

PySpark – withColumn() (https://sparkbyexamples.com/pyspark/pyspark-withcolumn/)

PySpark – withColumnRenamed() (https://sparkbyexamples.com/pyspark/pyspark-rename-dataframe-column/)

PySpark – where() & filter() (https://sparkbyexamples.com/pyspark/pyspark-where-filter/)

PySpark – drop() & dropDuplicates() (https://sparkbyexamples.com/pyspark/pyspark-distinct-to-drop-duplicates/)

PySpark – orderBy() and sort() (https://sparkbyexamples.com/

far more efficient file format than CSV or JSON, supported by many data processing systems.

It is compatible with most of the data processing frameworks in the Hadoop (https://en.wikipedia.org/wiki/Hadoop) echo systems. It provides efficient data compression and encoding schemes with enhanced performance to handle complex data in bulk.

val dF = spark.read.parquet("/tmp/output/people.parquet") //Read Parquet file
df.write.parquet("/tmp/output/people-new.parquet")//Writing parquet file

What is Avro

Apache Avro (https://sparkbyexamples.com/spark/read-write-avro-file-spark-dataframe/) is an open-source, row-based, data serialization and data exchange framework for Hadoop projects, originally developed by databricks as an open-source library that supports reading and writing data in Avro file format. it is mostly used in Apache Spark especially for Kafka-based data pipelines. When Avro data is stored in a file, its schema is stored with it, so that files may be processed later by any program.

It has build to serialize and exchange big data between different Hadoop based projects. It serializes data in a compact binary format and schema is in JSON format that defines the field names and data types.

pyspark/pyspark-orderby-and-sort-explained/)

PySpark – groupBy() (https://sparkbyexamples.com/pyspark/pyspark-groupby-explained-with-example/)

PySpark – join() (https://sparkbyexamples.com/pyspark/pyspark-join-explained-with-examples/)

PySpark – union() & unionAll() (https://sparkbyexamples.com/pyspark/pyspark-union-and-unionall/)

PySpark – unionByName() (https://sparkbyexamples.com/spark/spark-merge-two-dataframes-with-different-columns/)

PySpark – UDF (User Defined Function) (https://sparkbyexamples.com/pyspark/pyspark-udf-user-defined-function/)

PySpark – map() (https://sparkbyexamples.com/pyspark/pyspark-map-transformation/)

PySpark – flatMap() (https://sparkbyexamples.com/pyspark/pyspark-flatmap-transformation/)

pyspark – foreach() (https://sparkbyexamples.com/pyspark/pyspark-loop-iterate-through-rows-in-dataframe/#use-foreach-loop-through-dataframe)

PySpark – sample() vs sampleBy() (https://sparkbyexamples.com/pyspark/pyspark-sampling-example/)

PySpark – fillna() & fill() (https://sparkbyexamples.com/pyspark/pyspark-fillna-fill-replace-null-values/)

```
val df = spark.read.format("avro
df.write.format("avro").save("pe
//Avro Spark SQL
spark.sqlContext.sql("CREATE TEM
    OPTIONS (path \"person.avro\
spark.sqlContext.sql("SELECT * F
```

5. Avoid UDF's (User Defined Functions)

Try to avoid Spark/PySpark UDF's (https://sparkbyexamples.com/spark/spark-sql-udf/) at any cost and use when existing Spark built-in functions are not available for use. UDF's are a black box to Spark hence it can't apply optimization and you will lose all the optimization Spark does on Dataframe/Dataset. When possible you should use Spark SQL built-in functions (https://sparkbyexamples.com/spark/spark-sql-functions-understanding/) as these functions provide optimization

Before you create any UDF, do your research to check if the similar function you wanted is already available in Spark SQL Functions (https://sparkbyexamples.com/spark/spark-sql-functions-understanding/). Spark SQL provides several predefined common functions and many more new functions are added with every release. hence, It is best to check before you reinventing the wheel.

6. Persisting & Caching data in memory

Spark persisting/caching is one of the best techniques to improve the performance of the Spark workloads. Spark *Cache* and *Persist* are

PySpark – pivot() (Row to Column) (https://sparkbyexamples.com/pyspark/pyspark-pivot-and-unpivot-dataframe/)

PySpark – partitionBy() (https://sparkbyexamples.com/pyspark/pyspark-partitionby-example/)

PySpark – ArrayType Column (Array) (https://sparkbyexamples.com/pyspark/pyspark-arraytype-column-with-examples/)

PySpark – MapType (Map/Dict) (https://sparkbyexamples.com/pyspark/pyspark-maptype-dict-examples/)

PySpark SQL Functions

PySpark – Aggregate Functions (https://sparkbyexamples.com/pyspark/pyspark-aggregate-functions/)

PySpark – Window Functions (https://sparkbyexamples.com/pyspark/pyspark-window-functions/)

PySpark – Date and Timestamp Functions (https://sparkbyexamples.com/pyspark/pyspark-sql-date-and-timestamp-functions/)

PySpark – JSON Functions (https://sparkbyexamples.com/pyspark/pyspark-json-functions-with-examples/)

PySpark Datasources

PySpark – Read & Write CSV File (https://sparkbyexamples.com/pyspark/pyspark-read-csv-file-into-dataframe/)

PySpark – Read & Write Parquet File (https://sparkbyexamples.com/pyspark/pyspark-read-and-write-parquet-file/)

optimization techniques in DataFrame / Dataset (https://sparkbyexamples.com/spark/spark-dataframe-cache-and-persist-explained/) for iterative and interactive Spark applications to improve the performance of Jobs.

Using `cache()` and `persist()` methods, Spark provides an optimization mechanism to store the intermediate computation of a Spark DataFrame so they can be reused in subsequent actions.

When you persist a dataset, each node stores it's partitioned data in memory and reuses them in other actions on that dataset. And Spark's persisted data on nodes are fault-tolerant meaning if any partition of a Dataset is lost, it will automatically be recomputed using the original transformations that created it.

```
df.where(col("State") === "PR").
```

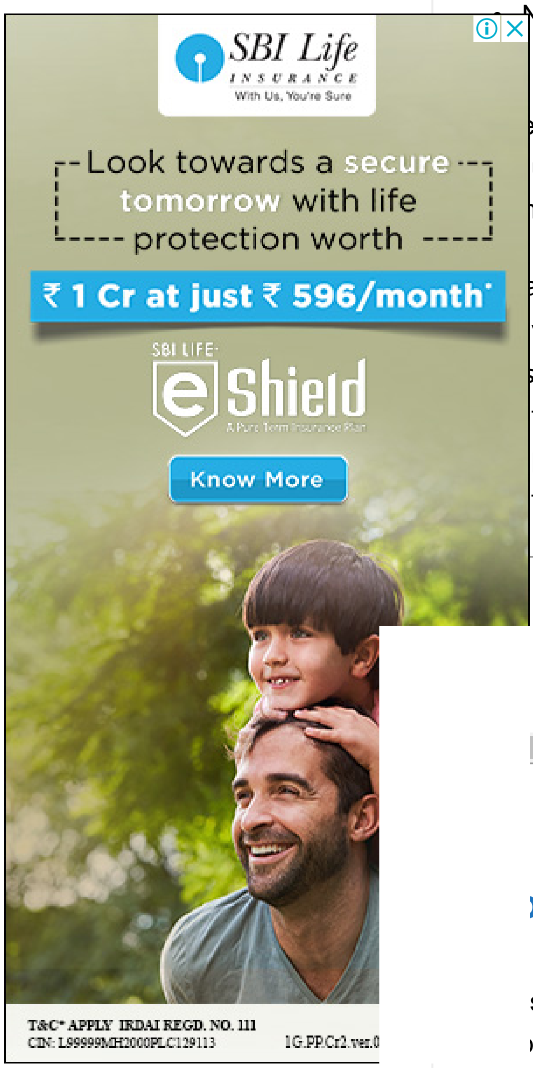
When caching use in-memory columnar format, By tuning the batchSize property you can also improve Spark performance.

```
spark.conf.set("spark.sql.inMemo
spark.conf.set("spark.sql.inMemo
```

Spark provides several storage levels to store the cached data (https://sparkbyexamples.com/spark/spark-persistence-storage-levels/), use the once which suits your cluster.

7. Reduce expensive Shuffle operations

PySpark – Read & Write JSON file (https://sparkbyexamples.com/pyspark/pyspark-read-json-file-into-dataframe/)


SBI Life INSURANCE
With Us, You're Sure
Look towards a secure tomorrow with life protection worth
₹ 1 Cr at just ₹ 596/month*
SBI LIFE eShield
A Pure Term Insurance Plan
Know More
T&C* APPLY IRDAI REGD. NO. 111
CIN: L99999MH2000PLC129113
1G.PP.Cr2.ver.0

PySpark Built-In Functions

PySpark – when() (https://sparkbyexamples.com/pyspark/pyspark-when-otherwise/)

Shuffling is a mechanism Spark uses to redistribute the data (https://sparkbyexamples.com/spark/spark-repartition-vs-coalesce/) across different executors and even across machines. Spark shuffling triggers when
perform certain transformation
rations
gropByKey(), reducebyKey(), jo
) on RDD and DataFrame.

ark Shuffle is an expensive operation
ce it involves the following

Disk I/O
nvolves data serialization and deserialization
Network I/O

cannot completely avoid shuffle
rations in but when possible try to
uce the number of shuffle operations
noved any unused operations.

ark
vides spark.sql.shuffle.partiti
configurations to control the
titions of the shuffle, By tuning this
perty you can improve Spark
formance.

```
park.conf.set("spark.sql.shuffl
qlContext.setConf("spark.sql.sh
```

Disable DEBUG & INFO
gging

s is one of the simple ways to
rove the performance of Spark Jobs and can be easily avoided by following good coding principles. During the development phase of Spark/PySpark application, we usually write debug/info

[PySpark – expr() (https://sparkbyexamples.com/pyspark/pyspark-sql-expr-expression-function/)](https://sparkbyexamples.com/pyspark/pyspark-sql-expr-expression-function/)

[PySpark – lit() (https://sparkbyexamples.com/pyspark/pyspark-lit-add-literal-constant/)](https://sparkbyexamples.com/pyspark/pyspark-lit-add-literal-constant/)

[PySpark – split() (https://sparkbyexamples.com/pyspark/pyspark-convert-string-to-array-column/)](https://sparkbyexamples.com/pyspark/pyspark-convert-string-to-array-column/)

[PySpark – concat_ws() (https://sparkbyexamples.com/pyspark/pyspark-convert-array-column-to-string-column/)](https://sparkbyexamples.com/pyspark/pyspark-convert-array-column-to-string-column/)

[Pyspark – substring() (https://sparkbyexamples.com/pyspark/pyspark-substring-from-a-column/)](https://sparkbyexamples.com/pyspark/pyspark-substring-from-a-column/)

[PySpark – translate() (https://sparkbyexamples.com/pyspark/pyspark-replace-column-values/#translate-replace-character-by-character)](https://sparkbyexamples.com/pyspark/pyspark-replace-column-values/#translate-replace-character-by-character)

[PySpark – regexp_replace() (https://sparkbyexamples.com/pyspark/pyspark-replace-column-values/#regexp_replace-replace-string-columns)](https://sparkbyexamples.com/pyspark/pyspark-replace-column-values/#regexp_replace-replace-string-columns)

[PySpark – overlay() (https://sparkbyexamples.com/pyspark/pyspark-replace-column-values/#overlay-function)](https://sparkbyexamples.com/pyspark/pyspark-replace-column-values/#overlay-function)

[PySpark – to_timestamp() (https://sparkbyexamples.com/spark/pyspark-to_timestamp-convert-string-to-timestamp-type/)](https://sparkbyexamples.com/spark/pyspark-to_timestamp-convert-string-to-timestamp-type/)

[PySpark – to_date() (https://sparkbyexamples.com/pyspark/pyspark-to_date-convert-timestamp-to-date/)](https://sparkbyexamples.com/pyspark/pyspark-to_date-convert-timestamp-to-date/)

[PySpark – date_format() (https://sparkbyexamples.com/pyspark/pyspark-date_format-convert-date-to-string-format/)](https://sparkbyexamples.com/pyspark/pyspark-date_format-convert-date-to-string-format/)

messages to console using `println()` and logging to a file using some logging framework (log4j);

These both methods results I/O operations hence cause performance issues when you run Spark jobs with greater workloads. Before promoting your jobs to production make sure you review your code and take care of the following.

Remove or convert all println() statements to log4j info/debug.

```
   logger.debug("Debug logging
   logger.info("Info logging me
```

Disable DEBUG/INFO by enabling ERROR/WARN/FATAL logging

If you are using log4j.properties use the following or use appropriate configuration based on your logging framework and configuration method (XML vs properties vs yaml)

```
log4j.rootLogger=warn, stdout
```

Personally I've seen this in my project where our team written 5 log statements in a [map() (https://sparkbyexamples.com/spark/spark-map-vs-mappartitions-transformation/)](https://sparkbyexamples.com/spark/spark-map-vs-mappartitions-transformation/) transformation; When we are processing 2 million records which resulted 10 million I/O operations and caused my job running for hrs. After disabling DEBUG & INFO logging I've witnessed jobs running in few mins.

Note: Spark workloads are increasingly bottlenecked by CPU and memory use rather than I/O and network, but still avoiding I/O operations are always a good practice.

Spark Performance Tuning Conclusion

Spark with Scala or Python (pyspark) jobs run on huge dataset's, when not following good coding principles and optimization techniques you will pay the price with performance bottlenecks, by following the topics I've covered in this article you will achieve improvement programmatically however there are other ways to improve the performance and tuning Spark jobs (by config & increasing resources) which I will cover in my next article.

References

- https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html (https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html)
- https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html (https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html)

Hope you like this article, leave me a comment if you like it or have any questions.

Happy Learning !!

Share this:

(https://sparkbyexamples.com/spark/spark-performance-tuning/?share=facebook&nb=1)

(https://sparkbyexamples.com/spark/spark-performance-tuning/?share=reddit&nb=1)

PySpark – datediff() (https://sparkbyexamples.com/pyspark/pyspark-difference-between-two-dates-days-months-years/#datediff)

PySpark – months_between() (https://sparkbyexamples.com/pyspark/pyspark-difference-between-two-dates-days-months-years/#months_between())

PySpark – explode() (https://sparkbyexamples.com/pyspark/pyspark-explode-nested-array-into-rows/)

PySpark – array_contains() (https://sparkbyexamples.com/pyspark/pyspark-arraytype-column-with-examples/#array_contains)

PySpark – array() (https://sparkbyexamples.com/pyspark/pyspark-arraytype-column-with-examples/#array)

PySpark – collect_list() (https://sparkbyexamples.com/pyspark/pyspark-aggregate-functions/#collect-list)

PySpark – collect_set() (https://sparkbyexamples.com/pyspark/pyspark-aggregate-functions/#collect-set)

PySpark – create_map() (https://sparkbyexamples.com/pyspark/pyspark-convert-dataframe-columns-to-maptype-dict/)

PySpark – map_keys() (https://sparkbyexamples.com/pyspark/pyspark-maptype-dict-examples/#map_keys)

PySpark – map_values() (https://sparkbyexamples.com/pyspark/pyspark-maptype-dict-examples/#map_values)

PySpark – struct() (https://sparkbyexamples.com/pyspark/pyspark-structtype-and-structfield/#update-struct-function)

[PySpark – countDistinct() (https://sparkbyexamples.com/pyspark/pyspark-count-distinct-from-dataframe/)]

[PySpark – sum(), avg() (https://sparkbyexamples.com/pyspark/pyspark-dataframe-groupby-and-sort-by-descending-order/)]

[PySpark – row_number() (https://sparkbyexamples.com/pyspark/pyspark-window-functions/#row_number)]

[PySpark – rank() (https://sparkbyexamples.com/pyspark/pyspark-window-functions/#rank)]

[PySpark – dense_rank() (https://sparkbyexamples.com/pyspark/pyspark-window-functions/#dense_rank)]

[PySpark – percent_rank() (https://sparkbyexamples.com/pyspark/pyspark-window-functions/#percent_rank)]

[PySpark – typedLit() (https://sparkbyexamples.com/pyspark/pyspark-lit-add-literal-constant/#typedlit)]

[PySpark – from_json() (https://sparkbyexamples.com/pyspark/pyspark-json-functions-with-examples/#from_json)]

[PySpark – to_json() (https://sparkbyexamples.com/pyspark/pyspark-json-functions-with-examples/#to_json)]

[PySpark – json_tuple() (https://sparkbyexamples.com/pyspark/pyspark-json-functions-with-examples/#json_tuple)]

[PySpark – get_json_object() (https://sparkbyexamples.com/pyspark/pyspark-json-functions-with-examples/#get_json_object)]

[PySpark – schema_of_json() (https://sparkbyexamples.com/py

(https://sparkbyexamples.com/spark/spark-performance-tuning/?share=pinterest&nb=1)

(https://sparkbyexamples.com/spark/spark-performance-tuning/?share=tumblr&nb=1)

(https://sparkbyexamples.com/spark/spark-performance-tuning/?share=pocket&nb=1)

(https://sparkbyexamples.com/spark/spark-performance-tuning/?share=linkedin&nb=1)

(https://sparkbyexamples.com/spark/spark-performance-tuning/?share=twitter&nb=1)

TAGS: PYSPARK OPTIMIZATION (HTTPS://SPARKBYEXAMPLES.COM/TAG/PYSPARK-OPTIMIZATION/), PYSPARK PERFORMANCE (HTTPS://SPARKBYEXAMPLES.COM/TAG/PYSPARK-PERFORMANCE/), PYSPARK TUNING (HTTPS://SPARKBYEXAMPLES.COM/TAG/PYSPARK-TUNING/), SPARK OPTIMIZATION (HTTPS://SPARKBYEXAMPLES.COM/TAG/SPARK-OPTIMIZATION/), SPARK PERFORMANCE (HTTPS://SPARKBYEXAMPLES.COM/TAG/SPARK-PERFORMANCE/)

(https://sparkbyexamples.com/author/admin/)

NNK (Https://Sparkbyexamples.Com/Author/Admin/)

SparkByExamples.com is a Big Data and Spark examples community page, all examples are simple and easy to understand and well tested in our development environment Read more .. (https://sparkbyexamples.com/about-sparkbyexamples/)

THIS POST HAS 4 COMMENTS

Sudhindra Kulkarni

23 JUN 2021 REPLY

Very nice explanation with good examples. Please keep the articles moving.

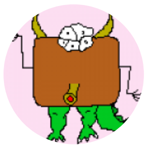
NNK 6 JUL 2021 REPLY

Thanks Sudhindra.

spark/pyspark-json-functions-with-examples/#schema_of_json)

Naveen 26 MAR 2021 REPLY

Hi..
Please Post the Performance tuning the spark code to load oracle table..



Anonymous

13 JAN 2021 REPLY

Very Well Explained in Simple way

Leave a Reply

Enter your comment here...

Categories

Apache Hadoop (https://sparkbyexamples.com/category/hadoop/)

Apache Spark (https://sparkbyexamples.com/category/spark/)

Apache Spark Streaming (https://sparkbyexamples.com/category/spark/apache-spark-streaming/)

Recent Posts

Spark regexp_replace() – Replace String Value (https://sparkbyexamples.com/spark/spark-regexp_replace-replace-string-value/)

How to Run a PySpark Script from Python? (https://sparkbyexamples.com/pyspark/run-pyspark-script-from-python-subprocess/)

About SparkByExamples.Com

SparkByExamples.com is a Big Data and Spark examples community page, all examples are simple and easy to understand, and well tested in our development environment Read more .. (https://sparkbyexamples.com/about-sparkbyexamples/)

Follow Us

Apache Kafka (https://sparkbyexamples.com/category/kafka/)

Apache HBase (https://sparkbyexamples.com/category/hbase/)

Apache Cassandra (https://sparkbyexamples.com/category/cassandra/)

Snowflake Database (https://sparkbyexamples.com/category/snowflake/)

H2O Sparkling Water (https://sparkbyexamples.com/category/h2o-sparkling-water/)

PySpark (https://sparkbyexamples.com/category/pyspark/)

Spark SQL like() Using Wildcard Example (https://sparkbyexamples.com/spark/spark-sql-like-using-wildcard-example/)

Spark isin() & IS NOT IN Operator Example (https://sparkbyexamples.com/spark/spark-isin-is-not-in-operator-example/)

Spark – Get Size/Length of Array & Map Column (https://sparkbyexamples.com/spark/spark-get-size-length-of-array-map-column/)

Spark Using Length/Size Of a DataFrame Column (https://sparkbyexamples.com/spark/spark-using-length-size-of-a-dataframe-column/)

Spark rlike() Working with Regex Matching Examples (https://sparkbyexamples.com/spark/spark-rlike-regex-matching-examples/)

Spark Check String Column Has Numeric Values (https://sparkbyexamples.com/spark/spark-check-string-column-has-numeric-values/)

Spark Check Column Data Type is Integer or String (https://sparkbyexamples.com/spark/spark-check-column-data-type-is-integer-or-string/)

(https://www.facebook.com/sparkbyexamples/) (https://www.linkedin.com/in/n-nk-b860a8193/) (https://twitter.com/sparkbyexamples) (https://github.com/spark-examples/)