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 Partitioning & Partition Understanding

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

Spark/PySpark partitioning is a way to split the data into multiple partitions so that you can execute transformations on multiple partitions in parallel which allows completing the job faster. You can also write partitioned data into a file system (multiple sub-directories) for faster reads by downstream systems.

Spark Tutorial

Spark – Installation on Windows (https://sparkbyexamples.com/spark/apache-spark-installation-on-windows/)

Spark – Installation on Linux | Ubuntu (https://sparkbyexamples.com/spark/spark-installation-on-linux-ubuntu/)

Spark – Cluster Setup with Hadoop Yarn (https://sparkbyexamples.com/spark/spark-setup-on-hadoop-yarn/)

Spark – Web/Application UI (https://sparkbyexamples.com/spark/spark-web-ui-understanding/)

Spark – Setup with Scala and IntelliJ (https://sparkbyexamples.com/spark/spark-setup-run-with-scala-intellij/)

Spark – How to Run Examples From this Site on IntelliJ IDEA (https://sparkbyexamples.com/spark/how-to-run-spark-examples-from-intellij/)

Spark – SparkSession (https://sparkbyexamples.com/spark/sparksession-explained-with-examples/)

Spark – SparkContext (https://sparkbyexamples.com/spark/spark-sparkcontext/)

Spark RDD Tutorial

Spark RDD – Parallelize (https://sparkbyexamples.com/apache-spark-rdd/how-to-create-an-rdd-using-parallelize/)

Spark has several partitioning methods to achieve parallelism, based on your need, you should choose which one to use.

SPARK DATAFRAME PARTITIONING METHODS (SCALA)	KEY POINTS
repartition(numPartitions : scala.Int)	Uses RoundRobinPartitioning
repartition(partitionExprs : Column*)	Uses HashPartitioner
repartition(numPartitions : scala.Int, partitionExprs : Column*)	partition = hash(partitionExprs) % numPartitions
coalesce(numPartitions : scala.Int)	Use only to reduce the number of partitions.
repartitionByRange(partitionExprs : Column*)	Uses rangepartitioning. Ideal to use on numeric columns.
repartitionByRange(numPartitions : scala.Int, partitionExprs : Column*)	
partitionBy(colNames : _root_.scala.Predef.String*)	Use to write the data into sub-folder

Note: `partitionBy()` is a method from `DataFrameWriter` class, all others are from `DataFrame`.

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. Understanding Spark Partitioning

- By default, Spark/PySpark creates partitions that are equal to the number of CPU cores in the machine.
- Data of each partition resides in a single machine.
- Spark/PySpark creates a task for each partition.
- Spark Shuffle operations (https://sparkbyexamples.com/spark/spark-shuffle-partitions/) move the data from one partition to other partitions.
- Partitioning is an expensive operation as it creates a data shuffle (Data could move between the nodes)
- By default, DataFrame shuffle operations create 200 partitions.

Spark/PySpark supports partitioning in memory (RDD/DataFrame) and partitioning on the disk (File system).

Partition in memory: You can partition or repartition the DataFrame by calling `repartition()` or `coalesce()` (https://sparkbyexamples.com/pyspark/pyspark-repartition-vs-coalesce/) transformations.

Partition on disk: While writing the PySpark DataFrame back to disk, you can choose how to partition the data based on columns by using `partitionBy()` of `pyspark.sql.DataFrameWriter`. This is similar to [Hives partitions (https://sparkbyexamples.com/apache-hive/hive-partitions-explained-with-examples/)](https://sparkbyexamples.com/apache-hive/hive-partitions-explained-with-examples/).

2. Spark Partitioning Advantages

As you are aware Spark is designed to process large datasets 100x faster than traditional processing, this wouldn't have been possible without partitions. Below are some of the advantages of using Spark partitions on memory or on disk.

- Fast accessed to the data.
- Provides the ability to perform an operation on a smaller dataset.

Partitioning at rest (disk) is a feature of many databases and data processing frameworks and it is key to make reads faster.

3. Default Spark Partitions & Configurations

Spark by default partitions data based on a number of factors, and the factors differ were you running your job on and what mode.

3.1 Local mode

When you running on local in standalone mode, Spark partitions data into the number of CPU cores you have on your system or the value you specify at the time of creating `SparkSession` object

[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/)]

```
import pyspark
from pyspark.sql import SparkSes
spark = SparkSession.builder.app
        .master("local[5]").getO
```

The above example provides `local[5]` as an argument to `master()` method meaning to run the job locally with 5 partitions. Though if you have just 2 cores on your system, it still creates 5 partition tasks.

```
df = spark.range(0,20)
print(df.rdd.getNumPartitions())
```

Above example yields output as 5 partitions.

3.2 HDFS cluster mode

When you running Spark jobs on the Hadoop cluster the default number of partitions is based on the following.

- On the HDFS cluster, by default, Spark creates one Partition for each block of the file.
- In Version 1 Hadoop the HDFS block size is 64 MB and in Version 2 Hadoop the HDFS block size is 128 MB

[Spark SQL – Partitioning (https://sparkbyexamples.com/spark/spark-partitioning-understanding/)](https://sparkbyexamples.com/spark/spark-partitioning-understanding/)

Spark SQL Functions

[Spark SQL String Functions (https://sparkbyexamples.com/spark/usage-of-spark-sql-string-functions/)](https://sparkbyexamples.com/spark/usage-of-spark-sql-string-functions/)

[Spark SQL Date and Timestamp Functions (https://sparkbyexamples.com/spark/spark-sql-date-and-time-functions/)](https://sparkbyexamples.com/spark/spark-sql-date-and-time-functions/)

[Spark SQL Array Functions (https://sparkbyexamples.com/spark/spark-sql-array-functions/)](https://sparkbyexamples.com/spark/spark-sql-array-functions/)

[Spark SQL Map Functions (https://sparkbyexamples.com/spark/spark-sql-map-functions/)](https://sparkbyexamples.com/spark/spark-sql-map-functions/)

[Spark SQL Sort Functions (https://sparkbyexamples.com/spark/spark-sql-sort-functions/)](https://sparkbyexamples.com/spark/spark-sql-sort-functions/)

[Spark SQL Aggregate Functions (https://sparkbyexamples.com/spark/spark-sql-aggregate-functions/)](https://sparkbyexamples.com/spark/spark-sql-aggregate-functions/)

[Spark SQL Window Functions (https://sparkbyexamples.com/spark/spark-sql-window-functions/)](https://sparkbyexamples.com/spark/spark-sql-window-functions/)

[Spark SQL JSON Functions (https://sparkbyexamples.com/spark/spark-most-used-json-functions-with-examples/)](https://sparkbyexamples.com/spark/spark-most-used-json-functions-with-examples/)

Spark Data Source API

[Spark – Read & Write CSV file (https://sparkbyexamples.com/spark/spark-read-csv-file-into-dataframe/)](https://sparkbyexamples.com/spark/spark-read-csv-file-into-dataframe/)

[Spark – Read and Write JSON file (https://sparkbyexamples.com/](https://sparkbyexamples.com/)

- Total number of cores on all executor nodes in a cluster or 2, whichever is larger

For example if you have 640 MB file and running it on Hadoop version 2, creates 5 partitions with each consists on 128 MB blocks (5 blocks * 128 MB = 640 MB). If you repartition to 10 then it creates 2 partitions for each block.

3.3 Spark configuration

- `spark.default.parallelism` configuration default value set to the number of all cores on all nodes in a cluster, on local it is set to a number of cores on your system.
- `spark.sql.shuffle.partitions` configuration default value is set to **200** and it is used when you call shuffle operations like `<a href="https://sparkbyexamples.com/spark/spark-dataframe-union-and-union-all/">union()</a>`, `<a href="https://sparkbyexamples.com/spark/using-groupby-on-dataframe/">groupBy()</a>`, `<a href="https://sparkbyexamples.com/spark/spark-sql-dataframe-join/">join()</a>` and many more. This property is available only in DataFrame API but not in RDD.

You can change the values of these properties through programmatically using the below statement.

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

You can also set the partition value of these configurations using `spark-submit` command.

Data Science

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/)

```
./bin/spark-submit --conf spark.
```

4. Dynamically Changing Spark Partitions

When you create an RDD/DataFrame from a file/table, based on certain parameters Spark creates them with a certain number of partitions and it also provides a way to change the partitions runtime in memory and options to partition based on one or multiple columns while writing to disk.

4.1 repartition() & coalesce()

While working with partition data we often need to increase or decrease the partitions based on data distribution. Methods repartition() and coalesce() helps us to repartition.

You can find the dataset explained in this article at GitHub zipcodes.csv file (https://github.com/spark-examples/pyspark-examples/blob/master/resources/simple-zipcodes.csv)

[Spark Streaming – Reading Files From Directory (https://sparkbyexamples.com/spark/spark-streaming-read-json-files-from-directory/)](https://sparkbyexamples.com/spark/spark-streaming-read-json-files-from-directory/)

[Spark Streaming – Reading Data From TCP Socket (https://sparkbyexamples.com/spark/spark-streaming-from-tcp-socket/)](https://sparkbyexamples.com/spark/spark-streaming-from-tcp-socket/)

[Spark Streaming – Processing Kafka Messages in JSON Format (https://sparkbyexamples.com/spark/spark-streaming-with-kafka/)](https://sparkbyexamples.com/spark/spark-streaming-with-kafka/)

[Spark Streaming – Processing Kafka messages in AVRO Format (https://sparkbyexamples.com/spark/spark-streaming-consume-and-produce-kafka-messages-in-avro-format/)](https://sparkbyexamples.com/spark/spark-streaming-consume-and-produce-kafka-messages-in-avro-format/)

[Spark SQL Batch – Consume & Produce Kafka Message (https://sparkbyexamples.com/spark/spark-batch-processing-produce-consume-kafka-topic/)](https://sparkbyexamples.com/spark/spark-batch-processing-produce-consume-kafka-topic/)

```
val rdd = spark.sparkContext.par
print("From local[5]"+rdd.getNum

val rdd1 = spark.sparkContext.pa
print("parallelize : "+rdd1.getN

#DataFrame repartition()
val df=spark.read.option("header
        .csv("/tmp/resources/sim
println(df.rdd.getNumPartitions(

#Change DataFrame partitions to
newDF=df.repartition(10)
println(newDF.rdd.getNumPartitio
```

Note: When you want to reduce the number of partitions, It is recommended to use [PySpark coalesce() over repartition() (https://sparkbyexamples.com/pyspark/pyspark-repartition-vs-coalesce/)](https://sparkbyexamples.com/pyspark/pyspark-repartition-vs-coalesce/) as it uses fewer resources due to less number of shuffles it takes.

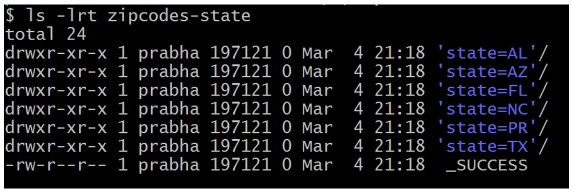
4.2 partitionBy()

Spark partitionBy() is a function of `pyspark.sql.DataFrameWriter` class which is used to partition based on one or multiple column values while writing DataFrame to Disk/File system.

When you write Spark DataFrame to disk by calling `partitionBy()`, PySpark splits the records based on the partition column and stores each partition data into a sub-directory.

```
#partitionBy()
df.write.option("header",True) \
        .partitionBy("state") \
        .mode("overwrite") \
        .csv("/tmp/zipcodes-stat
```

On our DataFrame, we have a total of 6 different states hence, it creates 6 directories as shown below. The name of the sub-directory would be the partition column and its value (partition column=value).



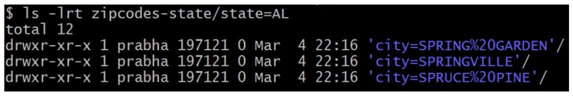
```
$ ls -lrt zipcodes-state
total 24
drwxr-xr-x 1 prabha 197121 0 Mar  4 21:18 'state=AL'/
drwxr-xr-x 1 prabha 197121 0 Mar  4 21:18 'state=AZ'/
drwxr-xr-x 1 prabha 197121 0 Mar  4 21:18 'state=FL'/
drwxr-xr-x 1 prabha 197121 0 Mar  4 21:18 'state=NC'/
drwxr-xr-x 1 prabha 197121 0 Mar  4 21:18 'state=PR'/
drwxr-xr-x 1 prabha 197121 0 Mar  4 21:18 'state=TX'/
-rw-r--r-- 1 prabha 197121 0 Mar  4 21:18  _SUCCESS
```

4.3 partitionBy() Multiple Columns

You can also create partitions on multiple columns using Spark `partitionBy()`. Just pass columns you want to partition as arguments to this method.

```
#partitionBy() multiple columns
df.write.option("header",True) \
        .partitionBy("state","ci
        .mode("overwrite") \
        .csv("/tmp/zipcodes-stat
```

It creates a folder hierarchy for each partition; we have mentioned the first partition as `state` followed by `city` hence, it creates a `city` folder inside the `state` folder (one folder for each `city` in a `state`).



```
$ ls -lrt zipcodes-state/state=AL
total 12
drwxr-xr-x 1 prabha 197121 0 Mar  4 22:16 'city=SPRING%20GARDEN'/
drwxr-xr-x 1 prabha 197121 0 Mar  4 22:16 'city=SPRINGVILLE'/
drwxr-xr-x 1 prabha 197121 0 Mar  4 22:16 'city=SPRUCE%20PINE'/
```

4.4 repartitionByRange() – Range Partition

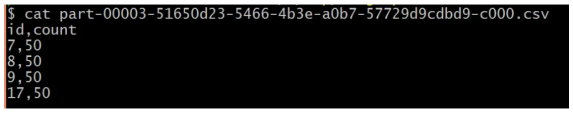
Below is a range partition example using `repartitionByRange()` transformation.

```
val data = Seq((1,10),(2,20),(3,
    (6,30),(7,50),(8,50),(9,50),
    (11,10),(12,10),(13,40),(14,
    (16,40),(17,50),(18,10),(19,
)

import spark.sqlContext.implicit
val dfRange = data.toDF("id","co
              .repartitionByRange

dfRange.write.option("header",tr
```

Let’s check data for one partition.



```
$ cat part-00003-51650d23-5466-4b3e-a0b7-57729d9cdbd9-c000.csv
id,count
7,50
8,50
9,50
17,50
```

Above, all data for `count=50` are in one partition.

5. How to Choose Spark Partition Column?

When using partitionBy(), you have to be very cautious with the number of partitions it creates, as having too many partitions creates too many sub-directories in a directory which brings unnecessarily and overhead to NameNode (if you are using Hadoop) since it must keep all metadata for the file system in memory.

Let’s assume you have a US census table that contains zip code, city, state, and other columns. Creating a partition on the state, splits the table into around 50 partitions, when searching for a zipcode within a state (state=’CA’ and zipCode =’92704′) results in faster as it needs to scan only in a **state=CA** partition directory.

Partition on zipcode may not be a good option as you might end up with too many partitions.

Another good example of partition is on the Date column. Ideally, you should partition on Year/Month but not on a date.

6. Too Many Partitions Good?

- If you are a beginner, you would think too many partitions will boost the Spark Job Performance (https://sparkbyexamples.com/spark/spark-performance-tuning/) actually, it won't and it's overkill.
- Spark has to create one task per partition and most of the time goes into creating, scheduling, and managing the tasks then executing.

7. Too Few Partitions Good?

- Too few partitions are not good as well, as you may not fully utilize your cluster resources.
- Less parallelism
- Applications may run longer as each partition takes more time to complete.

Conclusion

In this article, you have learned what is Spark/PySpark partitioning, different ways to do the partitioning, how to create dynamic partitions, and examples of how to do partitions.

Hope you like it. Happy Learning !!

Related Articles

- PySpark partitionBy() Explained with Examples (https://sparkbyexamples.com/pyspark/pyspark-partitionby-example/)
- PySpark repartition() vs coalesce() differences (https://sparkbyexamples.com/pyspark/pyspark-repartition-vs-coalesce/)
- PySpark Parallelize | Create RDD (https://sparkbyexamples.com/pyspark/pyspark-parallelize-create-rdd/)
- Spark SQL Shuffle Partitions (https://sparkbyexamples.com/spark/spark-shuffle-partitions/)

References

- https://mungingdata.com/apache-spark/partitionby/ (https://mungingdata.com/apache-spark/partitionby/)

TAGS: PYSPARK PARTITION (HTTPS://SPARKBYEXAMPLES.COM/TAG/PYSPARK-PARTITION/), PYSPARK PARTITIONING (HTTPS://SPARKBYEXAMPLES.COM/TAG/PYSPARK-PARTITIONING/), SPARK PARTITION (HTTPS://SPARKBYEXAMPLES.COM/TAG/SPARK-

PARTITION/), SPARK PARTITIONING (HTTPS://SPARKBYEXAMPLES.COM/TAG/SPARK-PARTITIONING/)

(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/)

Leave a Reply

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/)

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/)

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/)

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/)

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

(https: (https:

//www. //www.

(https: facebo linkedi (https:

//twitte ok.co n.com/ //githu

r.com/ m/spar in/n- b.com/

sparkb kbyex nk- spark-

yexam ample b860a examp

ples) s/) 8193/) les/)