Spark Cache vs Persist

Using `cache()` and `persist()` methods, Spark provides an optimization mechanism to store the intermediate computation of an RDD, DataFrame, and Dataset so they can be reused in subsequent actions(reusing the RDD, Dataframe, and Dataset computation result's).

Both caching and persisting are used to save the Spark RDD, Dataframe, and Dataset's. But, the difference is, RDD cache() method default saves it to memory (MEMORY_ONLY) whereas persist() method is used to store it to the user-defined storage level.

When you persist a dataset, each node stores its 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.

Advantages for Caching and Persistence

Below are the advantages of using Spark Cache and Persist methods.

Cost efficient – Spark computations are very expensive hence reusing the computations are used to save cost.

Time efficient – Reusing the repeated computations saves lots of time.

Execution time – Saves execution time of the job and we can perform more jobs on the same cluster.

Below I will explain how to use Spark Cache and Persist with DataFrame or Dataset.

Spark Cache Syntax and Example

Spark DataFrame or Dataset caching by default saves it to storage level `` `MEMORY_AND_DISK` `` because recomputing the in-memory columnar representation of the underlying table is expensive. Note that this is different from the default cache level of `` `RDD.cache()` `` which is '`MEMORY_ONLY`'.

Syntax

```
cache() : Dataset.this.type
```

Spark `cache()` method in Dataset class internally calls `persist()` method which in turn uses `sparkSession.sharedState.cacheManager.cacheQuery` to cache the result set of DataFrame or Dataset. Let's look at an example.

Example

```
val spark:SparkSession = SparkSession.builder()
   .master("local[1]")
   .appName("SparkByExamples.com")
   .getOrCreate()
 import spark.implicits._
```

```
  val columns = Seq("Seqno","Quote")
  val data = Seq(("1", "Be the change that you wish to see in the world"),
    ("2", "Everyone thinks of changing the world, but no one thinks of changing
himself."),
    ("3", "The purpose of our lives is to be happy."))
  val df = data.toDF(columns:_*)

  val dfCache = df.cache()
  dfCache.show(false)
```

Spark Persist Syntax and Example

Spark persist has two signature first signature doesn't take any argument which by default saves it to `MEMORY_AND_DISK` storage level and the second signature which takes `StorageLevel` as an argument to store it to different storage levels.

Syntax

```
1) persist() : Dataset.this.type
2) persist(newLevel : org.apache.spark.storage.StorageLevel) : Dataset.this.type
```

Example

```
  val dfPersist = df.persist()
  dfPersist.show(false)
```

Using the second signature you can save DataFrame/Dataset to One of the storage levels `MEMORY_ONLY`,`MEMORY_AND_DISK`, `MEMORY_ONLY_SER`, `MEMORY_AND_DISK_SER`, `DISK_ONLY`, `MEMORY_ONLY_2`,`MEMORY_AND_DISK_2`

```
  val dfPersist = df.persist(StorageLevel.MEMORY_ONLY)
  dfPersist.show(false)
```

This stores DataFrame/Dataset into Memory.

Unpersist syntax and Example

We can also unpersist the persistence DataFrame or Dataset to remove from the memory or storage.

Syntax

```
unpersist() : Dataset.this.type
unpersist(blocking : scala.Boolean) : Dataset.this.type
```

Example

```
  dfPersist = dfPersist.unpersist()
  dfPersist.show(false)
```

unpersist(Boolean) with boolean as argument blocks until all blocks are deleted.