CACHÉ

El almacenamiento en caché permite a Spark conservar los datos y las operaciones. De hecho, esta es una de las técnicas más importantes de Spark para acelerar los cálculos, especialmente cuando se trata de cálculos iterativos.

El almacenamiento en caché funciona almacenando el RDD tanto como sea posible en la memoria. Si no hay suficiente memoria, entonces los datos actuales en el almacenamiento se expulsan, según la directiva LRU. Si los datos que se piden que se almacenen en caché son mayores que la memoria disponible, el rendimiento se reducirá porque se usará Disco en lugar de memoria.

Puede marcar un RDD como almacenado en caché utilizando o persist() o cache().

Nota

cache() es simplemente un sinónimo de persistir(MEMORY_ONLY)

persist() puede utilizar memoria o disco o ambos:

```
persist(newLevel: StorageLevel)
```

Los siguientes son los posibles para el nivel de almacenamiento:

Nivel de almacenamiento	Significado
MEMORY_ONLY	Almacena RDD como objetos Java deserializados en la JVM. Si el RDD no cabe en la memoria, algunas particiones no se almacenarán en caché y se volverán a calcular sobre la marcha cada vez que se necesiten. Este es el nivel predeterminado.
MEMORY_AND_DISK	Almacena RDD como objetos Java deserializados en la JVM. Si el RDD no cabe en la memoria, almacene las particiones que no caben en el disco y léalas desde allí cuando se necesiten.
MEMORY_ONLY_SER (Java y Scala)	Almacena RDD como objetos Java serializados (una matriz de bytes por partición). Por lo general, esto es más eficiente en el espacio que los objetos deserializados, especialmente cuando se usa un serializador rápido, pero que consume más CPU para leer.
MEMORY_AND_DISK_SER (Java y Scala)	Similar a , pero derrame las particiones que no caben en la memoria en el disco en lugar de volver a calcularlas sobre la marcha cada vez que se necesitan.MEMORY_ONLY_SER
DISK_ONLY	Almacene las particiones RDD solo en el disco.
MEMORY_ONLY_2, , y así sucesivamente.MEMORY_AND_DISK_2	Igual que los niveles anteriores, pero replique cada partición en dos nodos de clúster.
OFF_HEAP (experimental)	Similar a , pero almacene los datos en memoria fuera del montón. Esto requiere que se habilite la memoria fuera del montón.MEMORY_ONLY_SER

El nivel de almacenamiento a elegir depende de la situación

- Si los RDD caben en la memoria, use como la opción más rápida para el rendimiento de ejecuciónMEMORY_ONLY
- Try es que hay objetos serializables que se utilizan con el fin de hacer los objetos más pequeñosMEMORY_ONLY_SER
- DISK no debe utilizarse a menos que sus cálculos sean caros.
- Utilice el almacenamiento replicado para obtener la mejor tolerancia a errores si puede ahorrar la memoria adicional necesaria. Esto evitará el cálculo de particiones perdidas para una mejor disponibilidad.

Nota

unpersist() simplemente libera el contenido almacenado en caché.

Los siguientes son ejemplos de cómo llamar a la función utilizando diferentes tipos de almacenamiento (memoria o disco):persist()

```
scala> import org.apache.spark.storage.StorageLevel

import org.apache.spark.storage.StorageLevel

scala> rdd_one.persist(StorageLevel.MEMORY_ONLY)

res37: rdd_one.type = ParallelCollectionRDD[26] at
parallelize at <console>:24

scala> rdd_one.unpersist()

res39: rdd_one.type = ParallelCollectionRDD[26] at
parallelize at <console>:24

scala> rdd_one.persist(StorageLevel.DISK_ONLY)

res40: rdd_one.type = ParallelCollectionRDD[26] at
parallelize at <console>:24

scala> rdd_one.unpersist()

res41: rdd_one.type = ParallelCollectionRDD[26] at
parallelize at <console>:24
```

La siguiente es una ilustración de la mejora del rendimiento que obtenemos al almacenar en caché.

En primer lugar, ejecutaremos el código:

```
scala> val rdd_one = sc.parallelize(Seq(1,2,3,4,5,6))

rdd_one: org.apache.spark.rdd.RDD[Int] =
ParallelCollectionRDD[0] at parallelize at <console>:24

scala> rdd_one.count

res0: Long = 6
```

```
scala> rdd_one.cache

res1: rdd_one.type = ParallelCollectionRDD[0] at
parallelize at <console>:24

scala> rdd_one.count

res2: Long = 6
```

Puede utilizar la WebUI para ver la mejora lograda como se muestra en las siguientes capturas de pantalla: