Spark 的 checkpoint 源码讲解

Checkpoint 相关源码分四个步部分

1，Checkpoint 的基本使用:core 和 Streaming。

2，初始化的源码。

3，Checkpoint 的 job 生成及执行的过程。

4，读 Checkpoint 的过程。

一，Checkpoint 的基本使用

Checkpoint 可以还原药水。辅助 Spark 应用从故障中恢复。SparkStreaming 宕机恢复，适合调度器有自动重试功能的。对于 SparkCore 则适合那些计算链条超级长或者计算耗时的关键点进行 Checkpoint，便于故障恢复。

Checkpoint 和 persist 从跟不上不一样：

1,Cache or persist

Cache or persist 保存了 RDD 的血统关系，假如有部分 cache 的数据丢失可以根据血缘关系重新生成。

2,Checkpoint

会将 RDD 数据写到 hdfs 这种安全的文件系统里，并且抛弃了 RDD 血缘关系的记录。即使 persist 存储到了磁盘里面，在 driver 停掉之后会被删除，而 checkpoint 可以被下次启动使用。

Checkpoint 基本使用

对于 SparkStreaming 任务，请参考源码例子 RecoverableNetworkWordCount

对 SparkCore：

DoCheckpoint

```
val sc = new SparkContext(confspark)
sc.setCheckpointDir("/test/checkpoint")
val textrdd=sc.textFile("/agent/test.txt")
textrdd.checkpoint()

val count = textrdd.count()
val file = textrdd.getCheckpointFile
println(file)
```

Recover

```
val count =
sc.checkpointFile("/test/checkpoint/7ce1511a-c25c-4ba7-9846-ee702e5c470d/rdd-1")
.count()
println(count)
```

二，Checkpoint 的初始化源码

1，设置 Checkpoint 目录

```
sc.setCheckpointDir("checkpointDirectory")
```

2，调用 Checkpoint 方法，构建 checkpointData

```
mapped.checkpoint()
checkpointData = Some(new ReliableRDDCheckpointData(this))
```

三，DoCheckpoint 源码

在 SparkContext 的 runJob 方法中

```
rdd.doCheckpoint()
```

进入之后

```
if (checkpointData.isDefined) {
  checkpointData.get.checkpoint()
} else {
  dependencies.foreach(_.rdd.doCheckpoint())
}
```

RDDCheckpointData 中真正做 Checkpoint 返回一个新的 RDD 并清除掉依赖关系

```
val newRDD = doCheckpoint()

// Update our state and truncate the RDD lineage
RDDCheckpointData.synchronized {
  cpRDD = Some(newRDD)
  cpState = Checkpointed
  rdd.markCheckpointed()
}
```

ReliableRDDCheckpointData 中真正进行 Checkpoint 操作

```
val newRDD = ReliableCheckpointRDD.writeRDDToCheckpointDirectory(rdd, cpDir)
```

在该方法中

1，获取 sc

```
val sc = originalRDD.sparkContext
```

2，创建输出目录

```
val checkpointDirPath = new Path(checkpointDir)
val fs = checkpointDirPath.getFileSystem(sc.hadoopConfiguration)
```

3，以 Job 的方式进行 Checkpoint 操作

```
val broadcastedConf = sc.broadcast(
  new SerializableConfiguration(sc.hadoopConfiguration))
// TODO: This is expensive because it computes the RDD again unnecessarily
(SPARK-8582)
sc.runJob(originalRDD,
  writePartitionToCheckpointFile[T](checkpointDirPath.toString, broadcastedConf)
_)
```

4，将分区策略写入 Checkpoint 目录

```
if (originalRDD.partitioner.nonEmpty) {
  writePartitionerToCheckpointDir(sc, originalRDD.partitioner.get,
checkpointDirPath)
}
```

四，读取 Checkpoint 数据

三种读法：

1，同一个 Spark 任务，共用了 Checkpoint 的 RDD，在该 RDD 的 iterator 方法中

```
final def iterator(split: Partition, context: TaskContext): Iterator[T] = {
  if (storageLevel != StorageLevel.NONE) {
    SparkEnv.get.cacheManager.getOrCompute(this, split, context, storageLevel)
  } else {
    computeOrReadCheckpoint(split, context)
  }
}
```

进入 computeOrReadCheckpoint

```
 private[spark] def computeOrReadCheckpoint(split: Partition, context:
TaskContext): Iterator[T] =
  {
    if (isCheckpointedAndMaterialized) {
      firstParent[T].iterator(split, context)
    } else {
      compute(split, context)
    }
  }
```

如果进行了 Checkpoint，条件为真

firstParent[T].iterator(split, context)其中，firstParent 为

```
/** Returns the first parent RDD */
protected[spark] def firstParent[U: ClassTag]: RDD[U] = {
  dependencies.head.rdd.asInstanceOf[RDD[U]]
}
```

接着是获取依赖

```
final def dependencies: Seq[Dependency[_]] = {
  checkpointRDD.map(r => List(new OneToOneDependency(r))).getOrElse {
    if (dependencies_ == null) {
      dependencies_ = getDependencies
    }
    dependencies_
  }
}
```

假如进行了 Checkpoint，那么 CheckpointRDD 就是存在

```
private def checkpointRDD: Option[CheckpointRDD[T]] =
checkpointData.flatMap(_.checkpointRDD)
```

在讲初始化 Checkpoint 的时候，我们已经初始化了 CheckpointData 了。

2，RDD 的计算链条失败，主动去读 Checkpoint 文件的过程

```
sc.checkpointFile("CheckpointRDD")
```

这个要求我们的入口类在

```
package org.apache.spark
```

3，SparkStreaming 的故障恢复

```
val checkpointOption = CheckpointReader.read(
  checkpointPath, new SparkConf(), hadoopConf, createOnError)
```

```
checkpointOption.map(new StreamingContext(null, _,
null)).getOrElse(creatingFunc())
```

首先，看一下 StreamingContext 的需要

```
StreamingContext private[streaming] (
    sc_ : SparkContext,
    cp_ : Checkpoint,
    batchDur_ : Duration
  ) extends Logging {
```

然后去读取 Checkpoint

```
val checkpointOption = CheckpointReader.read(
  checkpointPath, new SparkConf(), hadoopConf, createOnError)
```

分两个步骤

A,获取最新的 Checkpoint 目录

```
// Try to find the checkpoint files
val checkpointFiles = Checkpoint.getCheckpointFiles(checkpointDir,
Some(fs)).reverse
if (checkpointFiles.isEmpty) {
  return None
}
```

B,迭代找到最新的 Checkpoint 就返回

```
checkpointFiles.foreach(file => {
  logInfo("Attempting to load checkpoint from file " + file)
  try {
    val fis = fs.open(file)
    val cp = Checkpoint.deserialize(fis, conf)
    logInfo("Checkpoint successfully loaded from file " + file)
    logInfo("Checkpoint was generated at time " + cp.checkpointTime)
    return Some(cp)
  } catch {
    case e: Exception =>
      readError = e
      logWarning("Error reading checkpoint from file " + file, e)
  }
})
```

最后就是使用获取的 Checkpoint 去构建 ssc

```
checkpointOption.map(new StreamingContext(null, _,
null)).getOrElse(creatingFunc())
```

主要是做了一下动作

```
private[streaming] val isCheckpointPresent = (cp_ != null)

private[streaming] val sc: SparkContext = {
  if (sc_ != null) {
    sc_
  } else if (isCheckpointPresent) {
```

```
    SparkContext.getOrCreate(cp_.createSparkConf())
  } else {
    throw new SparkException("Cannot create StreamingContext without a
SparkContext")
  }
}
```

```
private[streaming] val graph: DStreamGraph = {
  if (isCheckpointPresent) {
    cp_.graph.setContext(this)
    cp_.graph.restoreCheckpointData()
    cp_.graph
  } else {
    require(batchDur_ != null, "Batch duration for StreamingContext cannot be null")
    val newGraph = new DStreamGraph()
    newGraph.setBatchDuration(batchDur_)
    newGraph
  }
}
```

```
private[streaming] var checkpointDir: String = {
  if (isCheckpointPresent) {
    sc.setCheckpointDir(cp_.checkpointDir)
    cp_.checkpointDir
  } else {
    null
  }
}
```

```
private[streaming] val checkpointDuration: Duration = {
  if (isCheckpointPresent) cp_.checkpointDuration else graph.batchDuration
}
```

```
private[streaming] def initialCheckpoint: Checkpoint = {
  if (isCheckpointPresent) cp_ else null
}
```