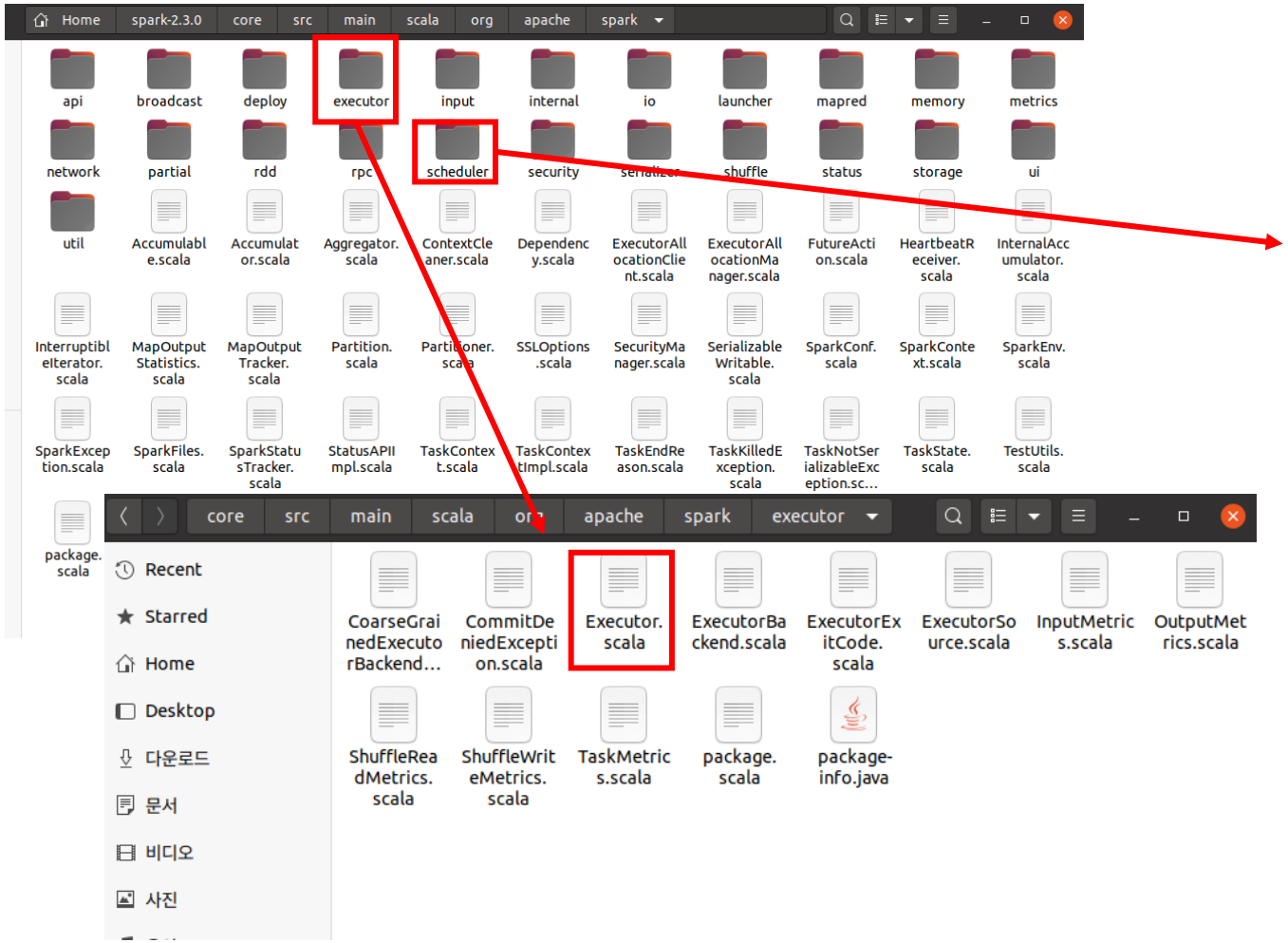
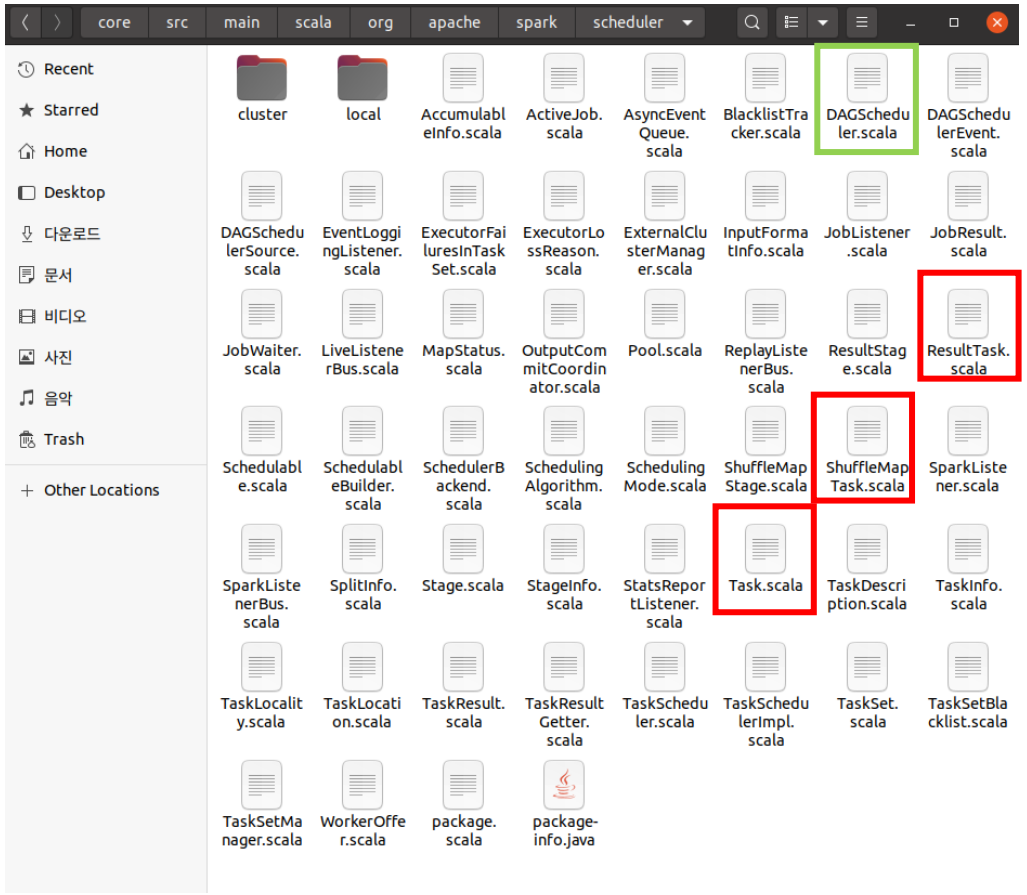
스파크 코드 분석

오현주

숙명여자대학교
SOOKMYUNG WOMEN'S UNIVERSITY

시작점 찾기


Home
spark-2.3.0
core
src
main
scala
org
apache
spark
api
broadcast
deploy
executor
input
internal
io
launcher
mapred
memory
metrics
network
partial
rdd
rpc
scheduler
security
shuffle
status
storage
ui
util
Accumulable.scala
Accumulator.scala
Aggregator.scala
ContextCleaner.scala
Dependency.scala
ExecutorAllocationClient.scala
ExecutorAllocationManager.scala
FutureAction.scala
HeartbeatReceiver.scala
InternalAccumulator.scala
InterruptibleIterator.scala
MapOutputStatistics.scala
MapOutputTracker.scala
Partition.scala
Partitioner.scala
SSLOptions.scala
SecurityManager.scala
SerializableWritable.scala
SparkConf.scala
SparkContext.scala
SparkEnv.scala
SparkException.scala
SparkFiles.scala
SparkStatusTracker.scala
StatusAPIImpl.scala
TaskContext.scala
TaskContextImpl.scala
TaskEndReason.scala
TaskKilledException.scala
TaskNotSerializableException.sc...
TaskState.scala
TestUtils.scala
package.scala
core
src
main
scala
org
apache
spark
executor
Recent
Starred
Home
Desktop
다운로드
문서
비디오
사진
CoarseGrainedExecutorBackend...
CommitDeniedException.scala
Executor.scala
ExecutorBackend.scala
ExecutorExitCode.scala
ExecutorSource.scala
InputMetrics.scala
OutputMetrics.scala
ShuffleReadMetrics.scala
ShuffleWriteMetrics.scala
TaskMetrics.scala
package.scala
package-info.java


core
src
main
scala
org
apache
spark
scheduler
Recent
Starred
Home
Desktop
다운로드
문서
비디오
사진
음악
Trash
Other Locations
cluster
local
AccumulableInfo.scala
ActiveJob.scala
AsyncEventQueue.scala
BlacklistTracker.scala
DAGScheduler.scala
DAGSchedulerEvent.scala
DAGSchedulerSource.scala
EventLoggingListener.scala
ExecutorFailuresInTaskSet.scala
ExecutorLossReason.scala
ExternalClusterManager.scala
InputFormatInfo.scala
JobListener.scala
JobResult.scala
JobWaiter.scala
LiveListenerBus.scala
MapStatus.scala
OutputCommitCoordinator.scala
Pool.scala
ReplayListenerBus.scala
ResultStage.scala
ResultTask.scala
Schedulable.scala
SchedulableBuilder.scala
SchedulerBackend.scala
SchedulingAlgorithm.scala
SchedulingMode.scala
ShuffleMapStage.scala
ShuffleMapTask.scala
SparkListener.scala
SparkListenerBus.scala
SplitInfo.scala
Stage.scala
StageInfo.scala
StatsReportListener.scala
Task.scala
TaskDescription.scala
TaskInfo.scala
TaskLocality.scala
TaskLocation.scala
TaskResult.scala
TaskResultGetter.scala
TaskScheduler.scala
TaskSchedulerImpl.scala
TaskSet.scala
TaskSetBlacklist.scala
TaskSetManager.scala
WorkerOffer.scala
package.scala
package-info.java

시작점 찾기

※ REPL(Read, Evaluate, Print, Loop) 에서 변수 선언이 이루어지면 인터프리터에서 해당 변수의 type과 value를 리턴해줌

SparkContext.scala | WordCountScalaHDFSScript.scala | MapOutputTracker.scala | TaskContext.scala

```
1 import java.util.ArrayList
2 import java.util.Random
3 val start = System.currentTimeMillis()
4
5 val text = sc.textFile("hdfs://localhost:9000/user/ubuntu/wordcount/input/500mb.csv")
6 val counts = text.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(_+_)
7 counts.saveAsTextFile("hdfs://localhost:9000/user/ubuntu/wordcount/output")
8 System.out.println("Spark Wordcount Complete")
9
10 val end = System.currentTimeMillis()
11 System.out.println("time:"+(end-start)/1000.0+"sec")
```

※ action 시작

```
user@user-MS-7B23:~/spark-2.3.0$ ./bin/spark-shell -i /home/user/WordCountScalaHDFSScript.scala
21/03/29 01:01:02 WARN Utils: Your hostname, user-MS-7B23 resolves to a loopback address: 127.0.1.1; using 203.153.147.216 instead (on interface eno1)
21/03/29 01:01:02 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
21/03/29 01:01:03 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://203.153.147.216:4040
Spark context available as 'sc' (master = local[*], app id = local-1616947266183).
Spark session available as 'spark'.
Loading /home/user/WordCountScalaHDFSScript.scala...
import java.util.ArrayList
import java.util.Random
start: Long = 1616947267445
text: org.apache.spark.rdd.RDD[String] = hdfs://localhost:9000/user/ubuntu/wordcount/input/500mb.csv MapPartitionsRDD[1] at textFile at <console>:26
counts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <console>:27
DAGScheduler.scala-runJob start
DAGScheduler.scala-submitJob() start
DAGScheduler.scala-submitJob() finish 0
```

DAG scheduler

Action의 결과를 얻어내기 위해 해야하는 일

- Job을 위한 execution DAG(Directed Acyclic Graph) 만듦
  - 즉, 각 Job을 위한 DAG stage들을 만듦
  - RDD Lineage를 stage DAG로 변환
    - Stage는 shuffle dependency들에 의해 나뉘어짐
    - 하나의 Stage 처리를 위해 하나 이상의 Task 필요

      → DAG scheduler는 생성된 stage들을 TaskSet들로 만들어서 TaskScheduler에게 submit함
- 고려해야할 부분 5가지
  - **Jobs, Stages, Tasks**, Cache tracking, Preferred locations, Cleanup

DAG scheduler

```scala
def runJob[T, U](
    rdd: RDD[T],
    func: (TaskContext, Iterator[T]) => U,
    partitions: Seq[Int],
    callSite: CallSite,
    resultHandler: (Int, U) => Unit,
    properties: Properties): Unit = {
  var startRunJob = System.currentTimeMillis() //kahu
  println("DAGScheduler.scala-runJob start")  //kahu
  val start = System.nanoTime

  println("DAGScheduler.scala-submitJob() start")  //kahu
  var startSubmitJob = System.currentTimeMillis() //kahu
  val waiter = submitJob(rdd, func, partitions, callSite, resultHandler, properties)
  var finishSubmitJob = System.currentTimeMillis() //kahu
  println("DAGScheduler.scala-submitJob() finish "+(finishSubmitJob-startSubmitJob)/1000)  //kahu

  ThreadUtils.awaitReady(waiter.completionFuture, Duration.Inf)
  waiter.completionFuture.value.get match {
    case scala.util.Success(_) =>
      logInfo("Job %d finished: %s, took %f s".format
        (waiter.jobId, callSite.shortForm, (System.nanoTime - start) / 1e9))
    case scala.util.Failure(exception) =>
      logInfo("Job %d failed: %s, took %f s".format
        (waiter.jobId, callSite.shortForm, (System.nanoTime - start) / 1e9))
      // SPARK-8644: Include user stack trace in exceptions coming from DAGScheduler.
      val callerStackTrace = Thread.currentThread().getStackTrace.tail
      exception.setStackTrace(exception.getStackTrace ++ callerStackTrace)
      throw exception
  }

  var finishRunJob = System.currentTimeMillis() //kahu
  println("DAGScheduler.scala-runJob finish "+(finishRunJob-startRunJob)/1000)  //kahu
}
```

```scala
/**
 * Submit an action job to the scheduler.
 *
 * @param rdd target RDD to run tasks on
 * @param func a function to run on each partition of the RDD
 * @param partitions set of partitions to run on; some jobs may not want to compute on all
 *   partitions of the target RDD, e.g. for operations like first()
 * @param callSite where in the user program this job was called
 * @param resultHandler callback to pass each result to
 * @param properties scheduler properties to attach to this job, e.g. fair scheduler pool name
 *
 * @return a JobWaiter object that can be used to block until the job finishes executing
 *         or can be used to cancel the job.
 *
 * @throws IllegalArgumentException when partitions ids are illegal
 */
def submitJob[T, U](
    rdd: RDD[T],
    func: (TaskContext, Iterator[T]) => U,
    partitions: Seq[Int],
    callSite: CallSite,
    resultHandler: (Int, U) => Unit,
    properties: Properties): JobWaiter[U] = {

  // Check to make sure we are not launching a task on a partition that does not exist.
  val maxPartitions = rdd.partitions.length
  partitions.find(p => p >= maxPartitions || p < 0).foreach { p =>
    throw new IllegalArgumentException(
      "Attempting to access a non-existent partition: " + p + ". " +
        "Total number of partitions: " + maxPartitions)
  }

  val jobId = nextJobId.getAndIncrement()
  if (partitions.size == 0) {
    // Return immediately if the job is running 0 tasks
    return new JobWaiter[U](this, jobId, 0, resultHandler)
  }

  assert(partitions.size > 0)
  val func2 = func.asInstanceOf[(TaskContext, Iterator[_]) => _]
  val waiter = new JobWaiter(this, jobId, partitions.size, resultHandler)
  eventProcessLoop.post(JobSubmitted(
    jobId, rdd, func2, partitions.toArray, callSite, waiter,
    SerializationUtils.clone(properties)))
  waiter
}
```

Executor.scala

- Spark executor, backed by a threadpool to run tasks
- Can be used with Mesos, YARN, and the standalone scheduler
- Executor.scala 내부의 TaskRunner 클래스

```scala
class TaskRunner(
    execBackend: ExecutorBackend,
    private val taskDescription: TaskDescription)
  extends Runnable {

  val taskId = taskDescription.taskId
  val threadName = s"Executor task launch worker for task $taskId"
  private val taskName = taskDescription.name

  /** If specified, this task has been killed and this option contains the reason. */
  @volatile private var reasonIfKilled: Option[String] = None

  @volatile private var threadId: Long = -1

  def getThreadId: Long = threadId

  /** Whether this task has been finished. */
  @GuardedBy("TaskRunner.this")
  private var finished = false

  def isFinished: Boolean = synchronized { finished }

  /** How much the JVM process has spent in GC when the task starts to run. */
  @volatile var startGCTime: Long = _

  /**
   * The task to run. This will be set in run() by deserializing the task binary coming
   * from the driver. Once it is set, it will never be changed.
   */
  @volatile var task: Task[Any] = _
```

Executor.scala

- TaskRunner 클래스 내의 run 함수
  - run 함수 내의 실제 task 실행 부분

```
override def run(): Unit = {
  var runJobStart = System.currentTimeMillis() //kahu
  println(executorId+" Executor.scala-Taskrunner.class-run()-start") //kahu

  threadId = Thread.currentThread.getId
  Thread.currentThread.setName(threadName)
  val threadMXBean = ManagementFactory.getThreadMXBean
  val taskMemoryManager = new TaskMemoryManager(env.memoryManager, taskId)
  val deserializeStartTime = System.currentTimeMillis()
  val deserializeStartCpuTime = if (threadMXBean.isCurrentThreadCpuTimeSupported) {
    threadMXBean.getCurrentThreadCpuTime
  } else 0L
  Thread.currentThread.setContextClassLoader(replClassLoader)
  val ser = env.closureSerializer.newInstance()
  logInfo(s"Running $taskName (TID $taskId)")
  execBackend.statusUpdate(taskId, TaskState.RUNNING, EMPTY_BYTE_BUFFER)
  var taskStart: Long = 0
  var taskStartCpu: Long = 0
  startGCTime = computeTotalGcTime()
```

```
// Run the actual task and measure its runtime.
println("real task start")  //kahu
taskStart = System.currentTimeMillis()
taskStartCpu = if (threadMXBean.isCurrentThreadCpuTimeSupported) {
  threadMXBean.getCurrentThreadCpuTime
} else 0L
var threwException = true
val value = try {
  val res = task.run(
    taskAttemptId = taskId,
    attemptNumber = taskDescription.attemptNumber,
    metricsSystem = env.metricsSystem)
  threwException = false
  res
} finally {
  val releasedLocks = env.blockManager.releaseAllLocksForTask(taskId)
  val freedMemory = taskMemoryManager.cleanUpAllAllocatedMemory()

  if (freedMemory > 0 && !threwException) {
    val errMsg = s"Managed memory leak detected; size = $freedMemory bytes, TID = $taskId"
    if (conf.getBoolean("spark.unsafe.exceptionOnMemoryLeak", false)) {
      throw new SparkException(errMsg)
    } else {
      logWarning(errMsg)
    }
  }

  if (releasedLocks.nonEmpty && !threwException) {
    val errMsg =
      s"${releasedLocks.size} block locks were not released by TID = $taskId:\n" +
        releasedLocks.mkString("[", ", ", "]")
    if (conf.getBoolean("spark.storage.exceptionOnPinLeak", false)) {
      throw new SparkException(errMsg)
    } else {
      logInfo(errMsg)
    }
  }
}
task.context.fetchFailed.foreach { fetchFailure =>
  // uh-oh.  it appears the user code has caught the fetch-failure without throwing any
  // other exceptions.  Its *possible* this is what the user meant to do (though highly
  // unlikely).  So we will log an error and keep going.
  logError(s"TID ${taskId} completed successfully though internally it encountered " +
    s"unrecoverable fetch failures!  Most likely this means user code is incorrectly " +
    s"swallowing Spark's internal ${classOf[FetchFailedException]}", fetchFailure)
}
val taskFinish = System.currentTimeMillis()
val taskFinishCpu = if (threadMXBean.isCurrentThreadCpuTimeSupported) {
  threadMXBean.getCurrentThreadCpuTime
} else 0L

println(executorId+": real task time "+(taskFinish-taskStart)/1000);  //kahu
```

Task.scala

- A unit of execution
- Two kinds of Task's in Spark
  - ShuffleMapTask.scala
    - 마지막 stage 이전의 stage들은 ShuffleMapTask들을 가지고 있음
    - A ShuffleMapTask executes the task and divides the task output to multiple buckets(based on the task's partitioner)
  - ResultTask.scala
    - Job에서 가장 마지막 stage는 여러 개의 ResultTask들을 가지고 있음
    - A ResultTask executes the task and sends the task output back to the driver application

ShuffleMapTask.scala

- Divides the elements of an RDD into multiple buckets
- ShuffleMapTask.scala에서 runTask 함수

```scala
override def runTask(context: TaskContext): MapStatus = {

  println("ShuffleMapTask.scala-runTask() start")  //kahu
  // Deserialize the RDD using the broadcast variable.
  val threadMXBean = ManagementFactory.getThreadMXBean
  val deserializeStartTime = System.currentTimeMillis()
  val deserializeStartCpuTime = if (threadMXBean.isCurrentThreadCpuTimeSupported) {
    threadMXBean.getCurrentThreadCpuTime
  } else 0L
  val ser = SparkEnv.get.closureSerializer.newInstance()
  val (rdd, dep) = ser.deserialize[(RDD[_], ShuffleDependency[_, _, _])](
    ByteBuffer.wrap(taskBinary.value), Thread.currentThread.getContextClassLoader)
  _executorDeserializeTime = System.currentTimeMillis() - deserializeStartTime
  _executorDeserializeCpuTime = if (threadMXBean.isCurrentThreadCpuTimeSupported) {
    threadMXBean.getCurrentThreadCpuTime - deserializeStartCpuTime
  } else 0L

  var writer: ShuffleWriter[Any, Any] = null
  try {
    val manager = SparkEnv.get.shuffleManager
    writer = manager.getWriter[Any, Any](dep.shuffleHandle, partitionId, context)
    writer.write(rdd.iterator(partition, context).asInstanceOf[Iterator[_ <: Product2[Any, Any]]])
    writer.stop(success = true).get
  } catch {
    case e: Exception =>
      try {
        if (writer != null) {
          writer.stop(success = false)
        }
      } catch {
        case e: Exception =>
          log.debug("Could not stop writer", e)
      }
      throw e
  }
}
```

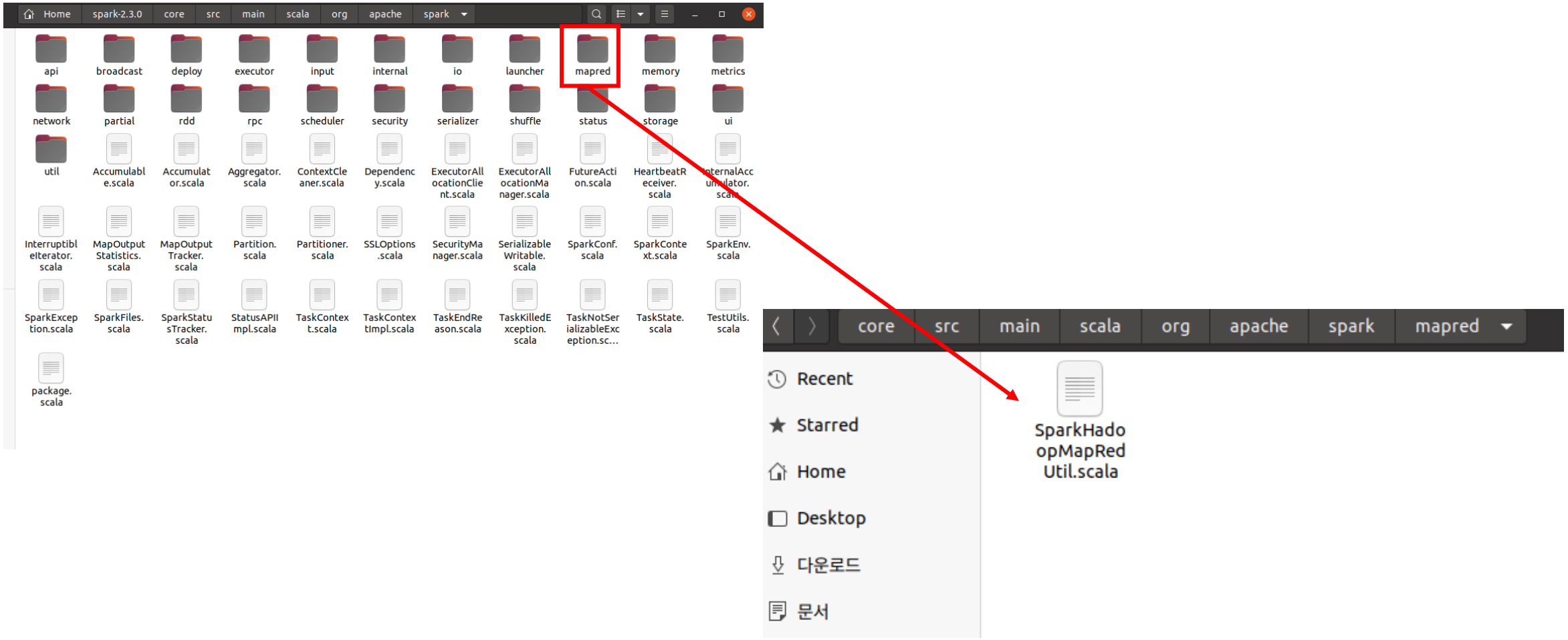
ResultTask.scala

- A task that sends back the output to the driver application
- ResultTask.scala의 runTask 메소드

```scala
override def runTask(context: TaskContext): U = {
  println("ResultTask.scala start")  //kahu
  // Deserialize the RDD and the func using the broadcast variables.
  val threadMXBean = ManagementFactory.getThreadMXBean
  val deserializeStartTime = System.currentTimeMillis()
  val deserializeStartCpuTime = if (threadMXBean.isCurrentThreadCpuTimeSupported) {
    threadMXBean.getCurrentThreadCpuTime
  } else 0L
  val ser = SparkEnv.get.closureSerializer.newInstance()
  val (rdd, func) = ser.deserialize[(RDD[T], (TaskContext, Iterator[T]) => U)](
    ByteBuffer.wrap(taskBinary.value), Thread.currentThread.getContextClassLoader)
  _executorDeserializeTime = System.currentTimeMillis() - deserializeStartTime
  _executorDeserializeCpuTime = if (threadMXBean.isCurrentThreadCpuTimeSupported) {
    threadMXBean.getCurrentThreadCpuTime - deserializeStartCpuTime
  } else 0L

  func(context, rdd.iterator(partition, context))
}
```

SparkHadoopMapRedUtil.scala



SparkHadoopMapRedUtil.scala

- Commits a task output
  - Coordinates with the driver in order to determine whether this attempt can commit
    - Some other task attempt might be racing to commit the same output partition
  - SparkHadoopMapRedUtil.scala 내의 commitTask 함수

```scala
def commitTask(
    committer: MapReduceOutputCommitter,
    mrTaskContext: MapReduceTaskAttemptContext,
    jobId: Int,
    splitId: Int): Unit = {

  println("SparkHadoopMapRedUtil.scala-commitTask() start")  //kahu

  val mrTaskAttemptID = mrTaskContext.getTaskAttemptID

  // Called after we have decided to commit
  def performCommit(): Unit = {
    try {
      committer.commitTask(mrTaskContext)
      logInfo(s"$mrTaskAttemptID: Committed")
    } catch {
      case cause: IOException =>
        logError(s"Error committing the output of task: $mrTaskAttemptID", cause)
        committer.abortTask(mrTaskContext)
        throw cause
    }
  }

  // First, check whether the task's output has already been committed by some other attempt
  if (committer.needsTaskCommit(mrTaskContext)) {
    val shouldCoordinateWithDriver: Boolean = {
      val sparkConf = SparkEnv.get.conf
      // We only need to coordinate with the driver if there are concurrent task attempts.
      // Note that this could happen even when speculation is not enabled (e.g. see SPARK-8029).
      // This (undocumented) setting is an escape-hatch in case the commit code introduces bugs.
      sparkConf.getBoolean("spark.hadoop.outputCommitCoordination.enabled", defaultValue = true)
    }

    if (shouldCoordinateWithDriver) {
      val outputCommitCoordinator = SparkEnv.get.outputCommitCoordinator
      val taskAttemptNumber = TaskContext.get().attemptNumber()
      val stageId = TaskContext.get().stageId()
      val canCommit = outputCommitCoordinator.canCommit(stageId, splitId, taskAttemptNumber)

      if (canCommit) {
        performCommit()
      } else {
        val message =
          s"$mrTaskAttemptID: Not committed because the driver did not authorize commit"
        logInfo(message)
        // We need to abort the task so that the driver can reschedule new attempts, if necessary
        committer.abortTask(mrTaskContext)
        throw new CommitDeniedException(message, stageId, splitId, taskAttemptNumber)
      }
    } else {
      // Speculation is disabled or a user has chosen to manually bypass the commit coordination
      performCommit()
    }
  } else {
    // Some other attempt committed the output, so we do nothing and signal success
    logInfo(s"No need to commit output of task because needsTaskCommit=false: $mrTaskAttemptID")
  }

  println("SparkHadoopMapRedUtil.scala-commitTask()-Finish")  //kahu
}
```

```
DAGScheduler.scala-runJob start
DAGScheduler.scala-submitJob() start
DAGScheduler.scala-submitJob() finish 0
driver Executor.scala-Taskrunner.class-run()-start
driver Executor.scala-Taskrunner.class-run()-start
driver Executor.scala-Taskrunner.class-run()-start
driver Executor.scala-Taskrunner.class-run()-start
driver Executor.scala-Taskrunner.class-run()-start
Executor.scala-Taskrunner.class-run()-actualTaskrun-start
Executor.scala-Taskrunner.class-run()-actualTaskrun-start
Executor.scala-Taskrunner.class-run()-actualTaskrun-start
Executor.scala-Taskrunner.class-run()-actualTaskrun-start
Executor.scala-Taskrunner.class-run()-actualTaskrun-start
ShuffleMapTask.scala-runTask() start
ShuffleMapTask.scala-runTask() start
ShuffleMapTask.scala-runTask() start
ShuffleMapTask.scala-runTask() start
ShuffleMapTask.scala-runTask() start
[Stage 0:>                                                          (0 + 5) / 5]driver Executor.scala-Taskrunner.class-run()-actualTaskrun-finish 2
driver Executor.scala-run()-finish 2
[Stage 0:===========>                                               (1 + 4) / 5]driver Executor.scala-Taskrunner.class-run()-actualTaskrun-finish 3
driver Executor.scala-run()-finish 3
driver Executor.scala-Taskrunner.class-run()-actualTaskrun-finish 3
driver Executor.scala-run()-finish 3
driver Executor.scala-Taskrunner.class-run()-actualTaskrun-finish 3
driver Executor.scala-run()-finish 3
driver Executor.scala-Taskrunner.class-run()-actualTaskrun-finish 3
driver Executor.scala-run()-finish 3
driver Executor.scala-Taskrunner.class-run()-start
driver Executor.scala-Taskrunner.class-run()-start
driver Executor.scala-Taskrunner.class-run()-start
driver Executor.scala-Taskrunner.class-run()-start
driver Executor.scala-Taskrunner.class-run()-start
Executor.scala-Taskrunner.class-run()-actualTaskrun-start
Executor.scala-Taskrunner.class-run()-actualTaskrun-start
Executor.scala-Taskrunner.class-run()-actualTaskrun-start
Executor.scala-Taskrunner.class-run()-actualTaskrun-start
Executor.scala-Taskrunner.class-run()-actualTaskrun-start
ResultTask.scala start
ResultTask.scala start
ResultTask.scala start
ResultTask.scala start
ResultTask.scala start
[Stage 1:>                                                          (0 + 5) / 5]SparkHadoopMapRedUtil.scala-commitTask() start
SparkHadoopMapRedUtil.scala-commitTask() start
SparkHadoopMapRedUtil.scala-commitTask() start
SparkHadoopMapRedUtil.scala-commitTask() start
SparkHadoopMapRedUtil.scala-commitTask() start
SparkHadoopMapRedUtil.scala-commitTask()-Finish
SparkHadoopMapRedUtil.scala-commitTask()-Finish
SparkHadoopMapRedUtil.scala-commitTask()-Finish
SparkHadoopMapRedUtil.scala-commitTask()-Finish
SparkHadoopMapRedUtil.scala-commitTask()-Finish
driver Executor.scala-Taskrunner.class-run()-actualTaskrun-finish 0
driver Executor.scala-Taskrunner.class-run()-actualTaskrun-finish 0
driver Executor.scala-Taskrunner.class-run()-actualTaskrun-finish 0
driver Executor.scala-Taskrunner.class-run()-actualTaskrun-finish 0
driver Executor.scala-run()-finish 0
driver Executor.scala-run()-finish 0
driver Executor.scala-Taskrunner.class-run()-actualTaskrun-finish 0
driver Executor.scala-run()-finish 0
driver Executor.scala-run()-finish 0
driver Executor.scala-run()-finish 0
DAGScheduler.scala-runJob finish 5
Spark WordCount Complete
end: Long = 1616997291629
time:6.838sec
```