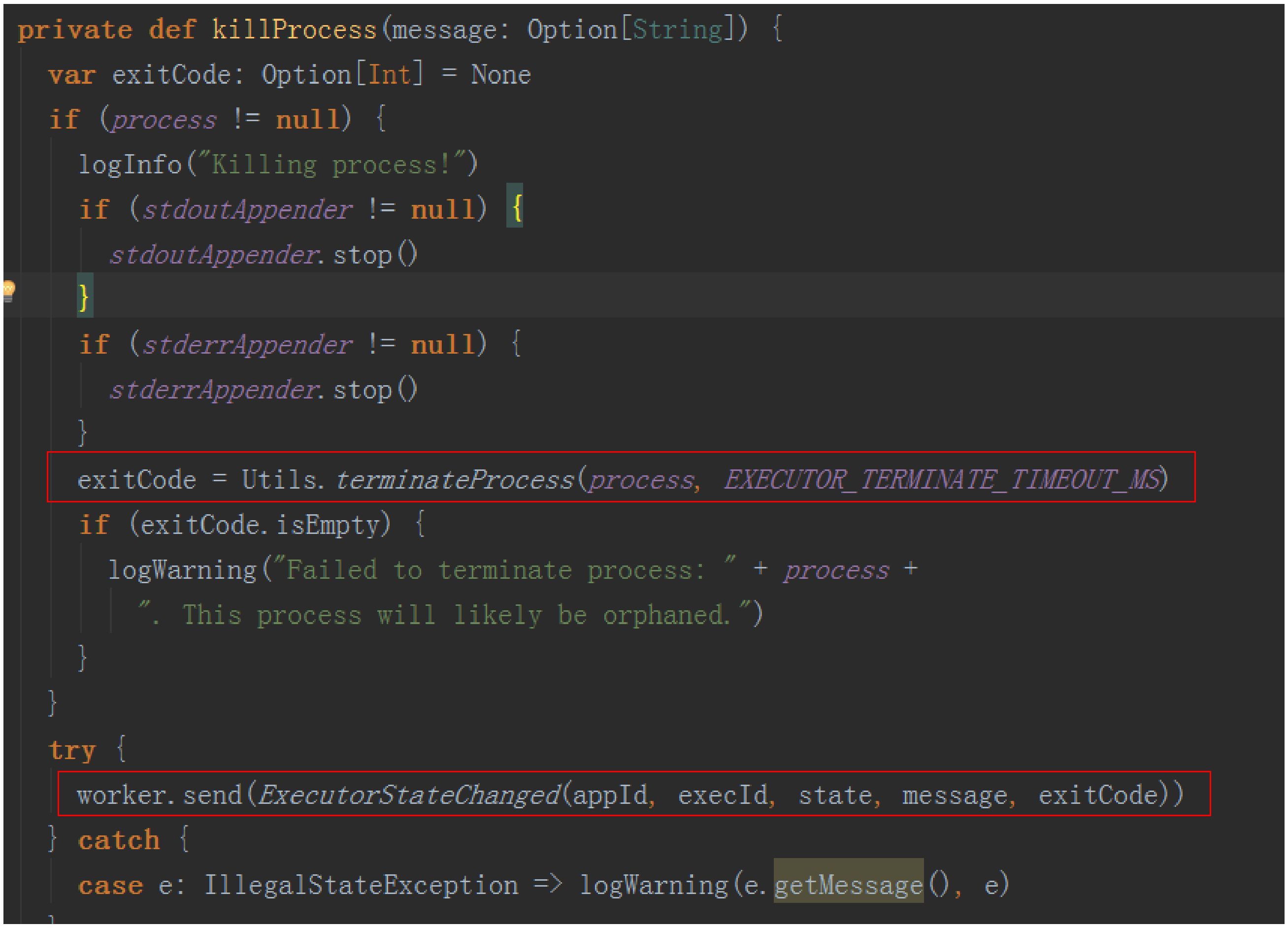
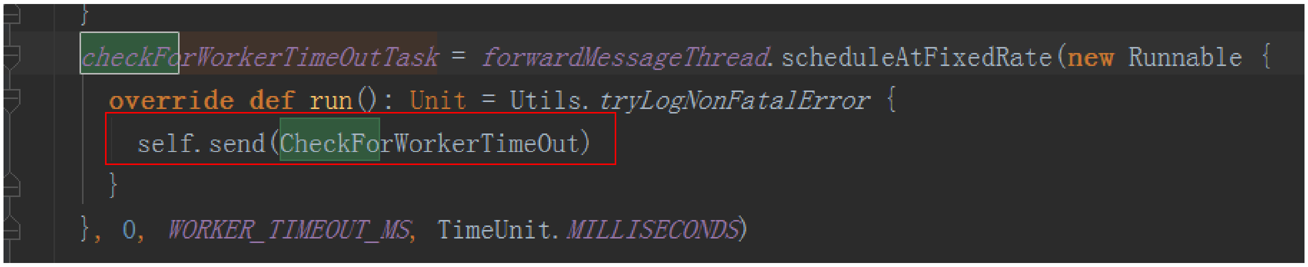
Worker异常退出

- Executor会调用shutdownHook，继而调用killProcess函数，这里会把进程的退出状态码exitCode作为ExecutorStateChanged的一部分发送给worker。



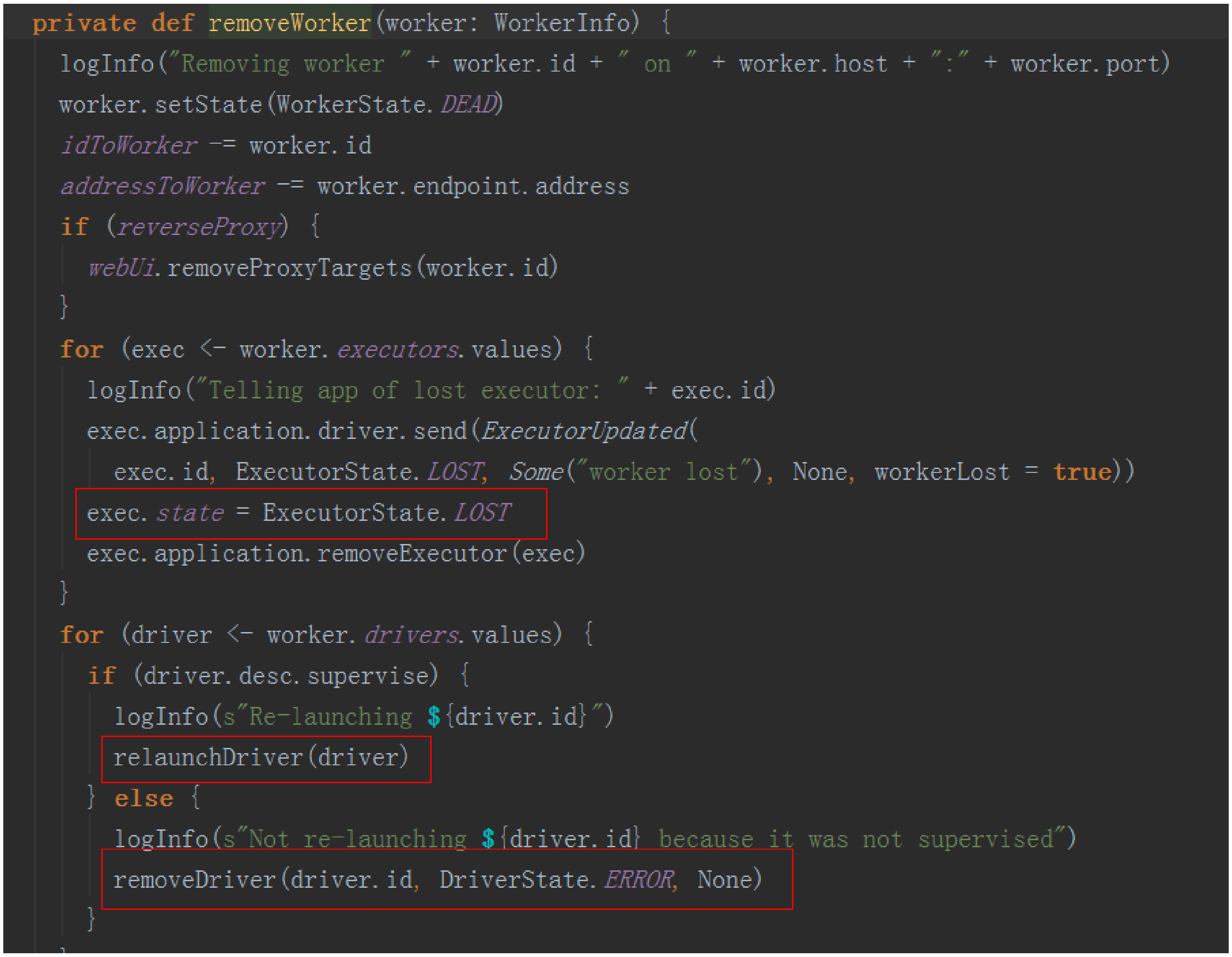
```
private def killProcess(message: Option[String]) {
  var exitCode: Option[Int] = None
  if (process != null) {
    logInfo("Killing process!")
    if (stdoutAppender != null) {
      stdoutAppender.stop()
    }
    if (stderrAppender != null) {
      stderrAppender.stop()
    }
    exitCode = Utils.terminateProcess(process, EXECUTOR_TERMINATE_TIMEOUT_MS)
    if (exitCode.isEmpty) {
      logWarning("Failed to terminate process: " + process +
        ". This process will likely be orphaned.")
    }
  }
  try {
    worker.send(ExecutorStateChanged(appId, execId, state, message, exitCode))
  } catch {
    case e: IllegalStateException => logWarning(e.getMessage(), e)
```

- 但是由于worker异常退出了，所以不会有HeartBeat消息发送给Master，但是Master会定时调用CheckForWorkerTimeOut检查worker的时间戳。



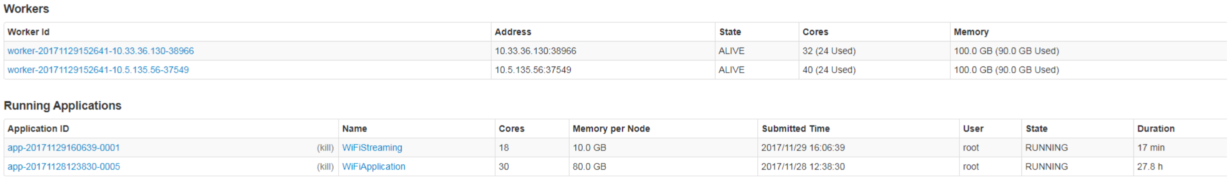
```
  }
  checkForWorkerTimeOutTask = forwardMessageThread.scheduleAtFixedRate(new Runnable {
    override def run(): Unit = Utils.tryLogNonFatalError {
      self.send(CheckForWorkerTimeOut)
    }
  }, 0, WORKER_TIMEOUT_MS, TimeUnit.MILLISECONDS)
```

- 那么就发现worker无法更新Master最后接收到的心跳的时间戳，那么Master就会调用removeWorker删除长期失联的worker的workInfo信息，并且将此worker的所有的Executor以LOST状态同步更新到Driver application，并且重新调度分配到其他的worker上。



```scala
private def removeWorker(worker: WorkerInfo) {
  logInfo("Removing worker " + worker.id + " on " + worker.host + ":" + worker.port)
  worker.setState(WorkerState.DEAD)
  idToWorker -= worker.id
  addressToWorker -= worker.endpoint.address
  if (reverseProxy) {
    webUi.removeProxyTargets(worker.id)
  }
  for (exec <- worker.executors.values) {
    logInfo("Telling app of lost executor: " + exec.id)
    exec.application.driver.send(ExecutorUpdated(
      exec.id, ExecutorState.LOST, Some("worker lost"), None, workerLost = true))
    exec.state = ExecutorState.LOST
    exec.application.removeExecutor(exec)
  }
  for (driver <- worker.drivers.values) {
    if (driver.desc.supervise) {
      logInfo(s"Re-launching ${driver.id}")
      relaunchDriver(driver)
    } else {
      logInfo(s"Not re-launching ${driver.id} because it was not supervised")
      removeDriver(driver.id, DriverState.ERROR, None)
    }
  }
```

- 重新调度的时候调用Master的schedule函数，这里会调用startExecutorsOnWorkers简单描述为：1、根据worker的剩余内存和剩余核数选择出可用的worker，如下图，worker-20171129152641-10.33.36.130-38966可用内存10G，可用核数24个，worker-20171129152641-10.5.135.56-37549可用内存10G，可用核数24个，UDE的启动参数是：--driver-memory 5g --executor-memory 10g --total-executor-cores 12。所以这两个都是可用worker



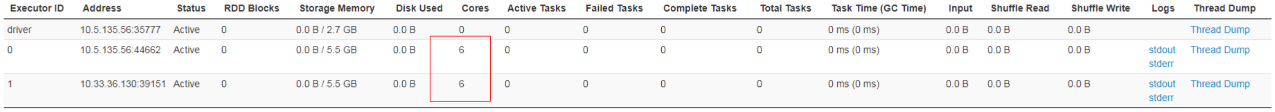
Workers

Worker Id	Address	State	Cores	Memory
worker-20171129152641-10.33.36.130-38966	10.33.36.130:38966	ALIVE	32 (24 Used)	100.0 GB (90.0 GB Used)
worker-20171129152641-10.5.135.56-37549	10.5.135.56:37549	ALIVE	40 (24 Used)	100.0 GB (90.0 GB Used)

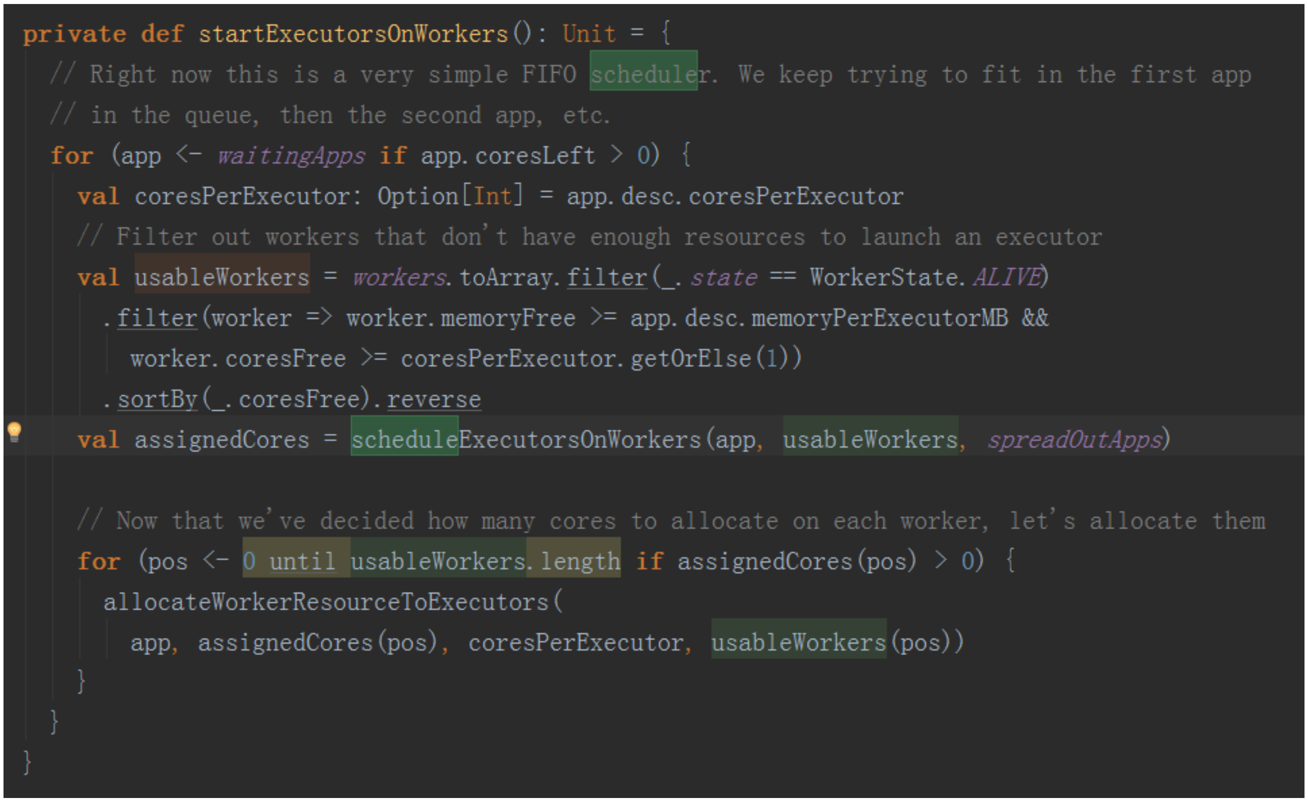
Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20171129160639-0001 (kill)	WiFiStreaming	18	10.0 GB	2017/11/29 16:06:39	root	RUNNING	17 min
app-20171128123830-0005 (kill)	WiFiApplication	30	80.0 GB	2017/11/28 12:38:30	root	RUNNING	27.8 h

- 然后不断循环遍历这两个worker，每次分配一个核，直到分配完全，也就是说核的最小分配粒度是1个，这样分配后如下所示：



Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
driver	10.5.135.56:35777	Active	0	0.0 B / 2.7 GB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump
0	10.5.135.56:44662	Active	0	0.0 B / 5.5 GB	0.0 B	6	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
1	10.33.36.130:39151	Active	0	0.0 B / 5.5 GB	0.0 B	6	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump

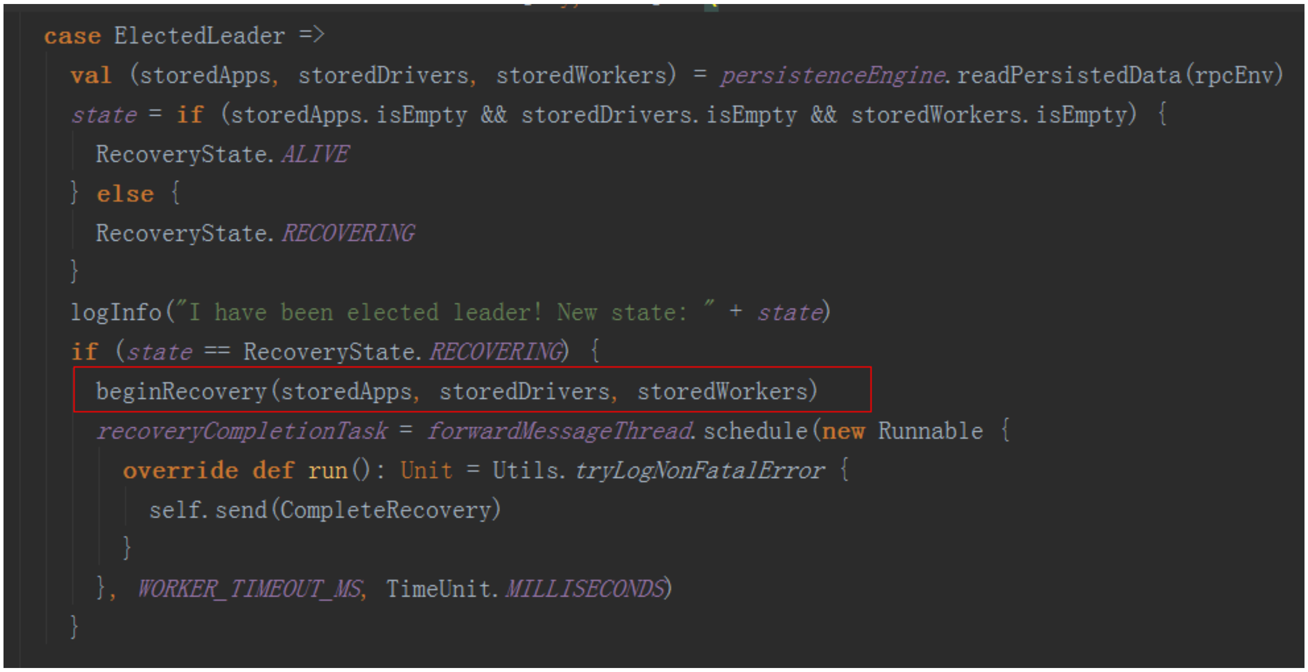


```
private def startExecutorsOnWorkers(): Unit = {
  // Right now this is a very simple FIFO scheduler. We keep trying to fit in the first app
  // in the queue, then the second app, etc.
  for (app <- waitingApps if app.coresLeft > 0) {
    val coresPerExecutor: Option[Int] = app.desc.coresPerExecutor
    // Filter out workers that don't have enough resources to launch an executor
    val usableWorkers = workers.toArray.filter(_.state == WorkerState.ALIVE)
      .filter(worker => worker.memoryFree >= app.desc.memoryPerExecutorMB &&
        worker.coresFree >= coresPerExecutor.getOrElse(1))
      .sortBy(_.coresFree).reverse
    val assignedCores = scheduleExecutorsOnWorkers(app, usableWorkers, spreadOutApps)

    // Now that we've decided how many cores to allocate on each worker, let's allocate them
    for (pos <- 0 until usableWorkers.length if assignedCores(pos) > 0) {
      allocateWorkerResourceToExecutors(
        app, assignedCores(pos), coresPerExecutor, usableWorkers(pos))
    }
  }
}
```

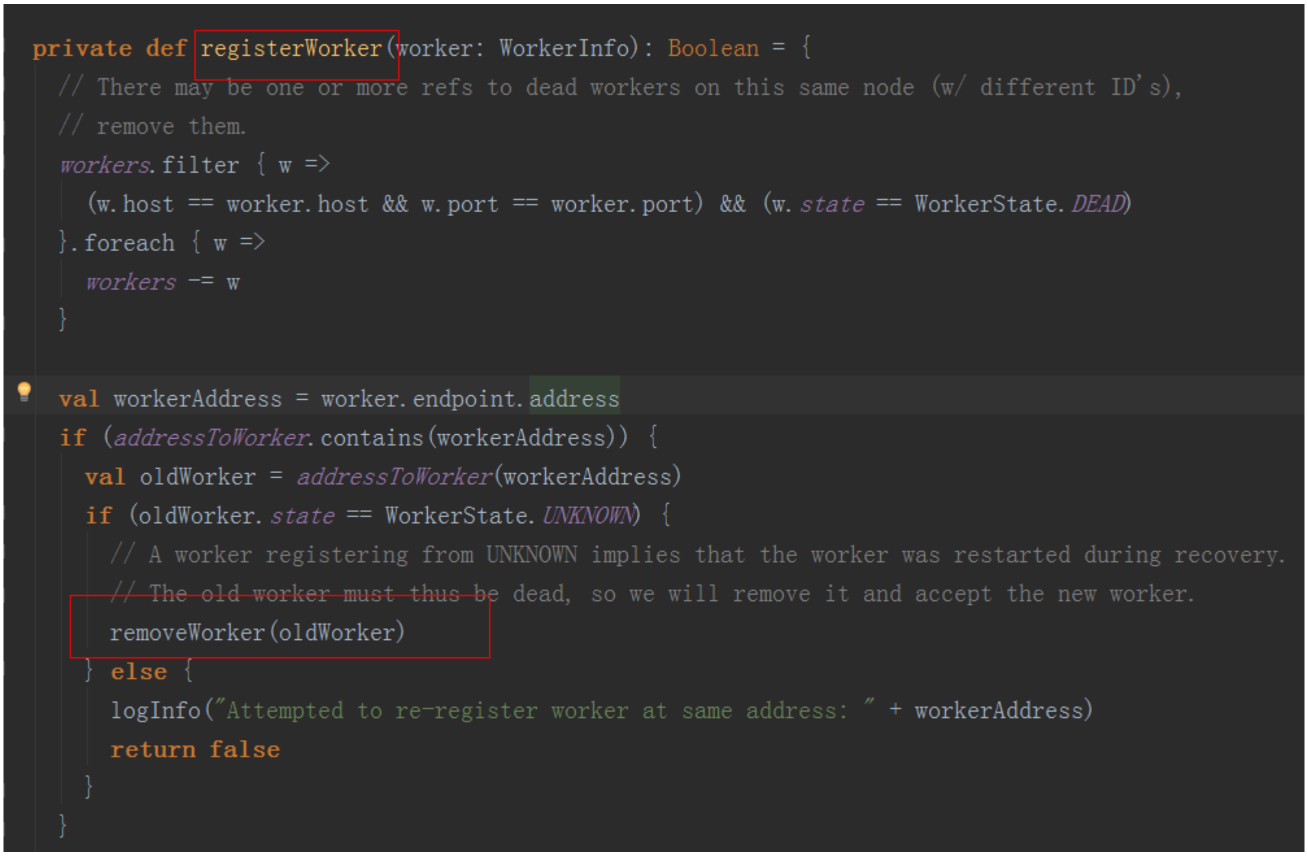
Master异常退出

- Master退出后会进行选举，当选举出新的Master后，就需要恢复之前的信息，主要有：storedApps、storedDrivers、storedWorkers



```
case ElectedLeader =>
  val (storedApps, storedDrivers, storedWorkers) = persistenceEngine.readPersistedData(rpcEnv)
  state = if (storedApps.isEmpty && storedDrivers.isEmpty && storedWorkers.isEmpty) {
    RecoveryState.ALIVE
  } else {
    RecoveryState.RECOVERING
  }
  logInfo("I have been elected leader! New state: " + state)
  if (state == RecoveryState.RECOVERING) {
    beginRecovery(storedApps, storedDrivers, storedWorkers)
    recoveryCompletionTask = forwardMessageThread.schedule(new Runnable {
      override def run(): Unit = Utils.tryLogNonFatalError {
        self.send(CompleteRecovery)
      }
    }, WORKER_TIMEOUT_MS, TimeUnit.MILLISECONDS)
  }
```

- 在恢复的时候就会注册新worker信息，并删除以前的worker信息，并重新调度，那么重复上面的调度流程。



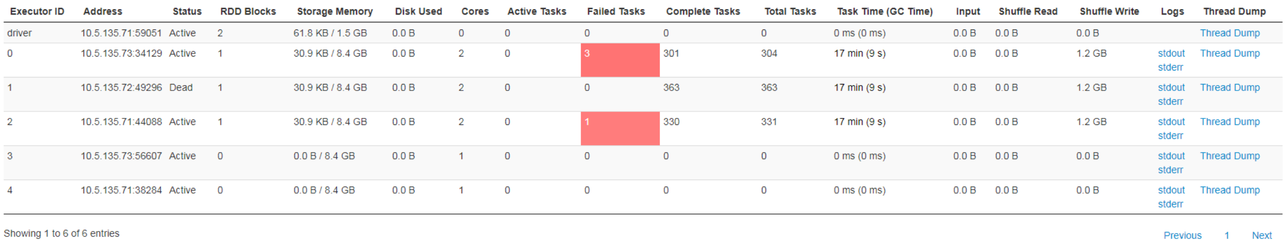
```scala
private def registerWorker(worker: WorkerInfo): Boolean = {
  // There may be one or more refs to dead workers on this same node (w/ different ID's),
  // remove them.
  workers.filter { w =>
    (w.host == worker.host && w.port == worker.port) && (w.state == WorkerState.DEAD)
  }.foreach { w =>
    workers -= w
  }

  val workerAddress = worker.endpoint.address
  if (addressToWorker.contains(workerAddress)) {
    val oldWorker = addressToWorker(workerAddress)
    if (oldWorker.state == WorkerState.UNKNOWN) {
      // A worker registering from UNKNOWN implies that the worker was restarted during recovery.
      // The old worker must thus be dead, so we will remove it and accept the new worker.
      removeWorker(oldWorker)
    } else {
      logInfo("Attempted to re-register worker at same address: " + workerAddress)
      return false
    }
  }
```

结论

总结：只要是ude的driver挂掉，那么就不会出现分裂，否则就会出现分裂。

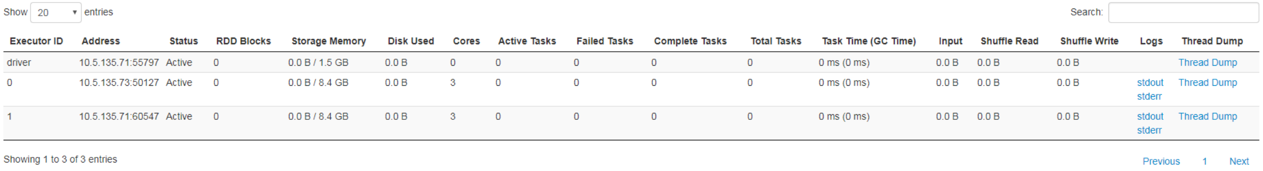
- 所以如果worker异常退出，并且不是UDE的driver，那么就会出现分裂，如下所示：



Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
driver	10.5.135.71:59051	Active	2	61.8 KB / 1.5 GB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump
0	10.5.135.73:34129	Active	1	30.9 KB / 8.4 GB	0.0 B	2	0	3	301	304	17 min (9 s)	0.0 B	0.0 B	1.2 GB	stdout stderr	Thread Dump
1	10.5.135.72:49296	Dead	1	30.9 KB / 8.4 GB	0.0 B	2	0	0	363	363	17 min (9 s)	0.0 B	0.0 B	1.2 GB	stdout stderr	Thread Dump
2	10.5.135.71:44088	Active	1	30.9 KB / 8.4 GB	0.0 B	2	0	1	330	331	17 min (9 s)	0.0 B	0.0 B	1.2 GB	stdout stderr	Thread Dump
3	10.5.135.73:56607	Active	0	0.0 B / 8.4 GB	0.0 B	1	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
4	10.5.135.71:38284	Active	0	0.0 B / 8.4 GB	0.0 B	1	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump

Showing 1 to 6 of 6 entries

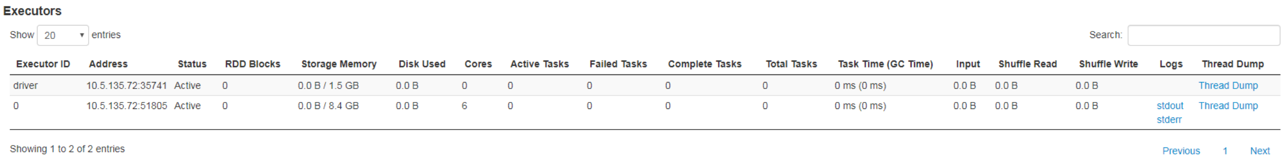
- 所以如果worker异常退出，并且是UDE的driver，那么就不会出现分裂，如下所示：



Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
driver	10.5.135.71:55797	Active	0	0.0 B / 1.5 GB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump
0	10.5.135.73:50127	Active	0	0.0 B / 8.4 GB	0.0 B	3	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
1	10.5.135.71:60547	Active	0	0.0 B / 8.4 GB	0.0 B	3	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump

Showing 1 to 3 of 3 entries

- 如果是master异常退出，并且是UDE的dirver，那么就不会出现分裂，如下所示：

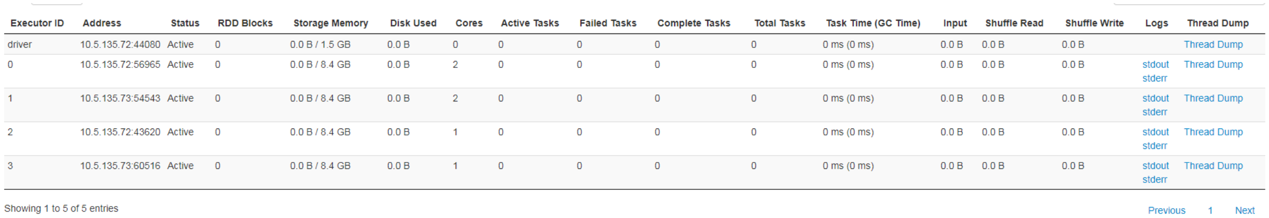


Executors

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
driver	10.5.135.72:35741	Active	0	0.0 B / 1.5 GB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump
0	10.5.135.72:51805	Active	0	0.0 B / 8.4 GB	0.0 B	6	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump

Showing 1 to 2 of 2 entries

- 如果是master异常退出，并且不是UDE的dirver，那么就会出现分裂，如下所示：



Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
driver	10.5.135.72:44080	Active	0	0.0 B / 1.5 GB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump
0	10.5.135.72:56965	Active	0	0.0 B / 8.4 GB	0.0 B	2	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
1	10.5.135.73:54543	Active	0	0.0 B / 8.4 GB	0.0 B	2	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
2	10.5.135.72:43620	Active	0	0.0 B / 8.4 GB	0.0 B	1	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
3	10.5.135.73:60516	Active	0	0.0 B / 8.4 GB	0.0 B	1	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump

Showing 1 to 5 of 5 entries

可改进方法

- 加这个参数能解决问题， --executor-cores 4
- 修改Spark调度的代码，比如按worker的可用资源排序，并且设定一个资源分配的粒度，比如4，那么举例说明如下：比如有3个worker1、2、3，可用资源分别是8、7、6，需要分配的资源是3，那么就直接在1号worker上起1个Executor就可以了。现有的方案是1、2、3号worker分别起1个Executor，每个Executor1个核。