Actor Bug: Concurrency - Workload

Sometimes an actor's task can be too computationally taxing and it won't complete in time or respond to messages it receives as fast as expected as a result. This can cause errors in a system that hasn't been designed to anticipate these kinds of problems. Such as getting multiple duplicate messages or even being terminated as a dead-locked actor while busy due to configured time out windows.

Spark had one such difficulty with increased work times in their executors. The problem would occur when an executor would be performing work that lasted past the set executorIdleTimeout time frame and it would be marked as idle and for termination. If the remaining computations would take this much time or longer than the job would continuously fail and never finish. To fix this, logic had to be added to how and when an executor is labeled as being idle. Support was added to prevent wrongfully adding busy tasks to the expired executors list.

This issue was solved by checking more thoroughly if a node is busy or if it's hung. First, it is more thoroughly checked for expiration in the schedule method, and if it is expired, the removeExecutor method is run upon it's executorId. To check for idle executors, a method, isExecutorIdle was added and is called whenever an Executor is added in the onExecutorAdded method.

Link: https://github.com/apache/spark/pull/3783

Code:

spark/ExecutorAllocationManager.scal

```
...
   private def schedule(): Unit = synchronized {
     val now = clock.getTimeMillis
     if (addTime != NOT_SET && now >= addTime) {
       addExecutors()
       logDebug(s"Starting timer to add more executors (to " +
         s"expire in $sustainedSchedulerBacklogTimeout seconds)")
       addTime += sustainedSchedulerBacklogTimeout * 1000
     }

     removeTimes.retain { case (executorId, expireTime) =>
-      now < expireTime || !removeExecutor(executorId)
+      val expired = now >= expireTime
+      if (expired) {
+        removeExecutor(executorId)
+      }
+      !expired
     }
   }
...
   private def onExecutorAdded(executorId: String): Unit = synchronized {
     if (!executorIds.contains(executorId)) {
       executorIds.add(executorId)
-      onExecutorIdle(executorId)
+      executorIds.filter(listener.isExecutorIdle).foreach(onExecutorIdle)
       logInfo(s"New executor $executorId has registered (new total is ${executorIds.size})")
       if (numExecutorsPending > 0) {
         numExecutorsPending -= 1
         logDebug(s"Decremented number of pending executors ($numExecutorsPending left)")
       }
     } else {
       logWarning(s"Duplicate executor $executorId has registered")
     }
}
...
+
+    /**
+     * Note: This is not thread-safe without the caller owning the `allocationManager` lock.
+     */
+    def isExecutorIdle(executorId: String): Boolean = {
+      !executorIdToTaskIds.contains(executorId)
+    }
```