a3.scala(Code):

```
package streaming

import org.apache.spark.SparkConf
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.dstream.DStream

/**
 * Author: Dhruvi Trivedi (s4146514), Maitreya Milind Kadam(s4087536)
 * RMIT University -      COSC 2637/2633  Big Data Processing
 *
 * Assignment 3 – HDFS Monitoring via Spark Streaming
 */

object a3 {

  def main(args: Array[String]): Unit = {

    if (args.length != 2) {
      System.err.println(
        "Usage: spark-submit --class streaming.a3 --master yarn --deploy-mode client
a3.jar hdfs:///input hdfs:///output"
      )
      System.exit(1)
    }

    val inputDir = args(0)
    val outputDir = args(1)
    val checkpointDir = "/s4146514/checkpoint"

    val conf = new SparkConf().setAppName("A3-Streaming")
    val ssc = new StreamingContext(conf, Seconds(3))
    ssc.checkpoint(checkpointDir)

    val lines = ssc.textFileStream(inputDir)

    // Atomic counters to generate unique output folder suffixes
    val task1Seq = new java.util.concurrent.atomic.AtomicInteger(1)
    val task2Seq = new java.util.concurrent.atomic.AtomicInteger(1)
    val task3Seq = new java.util.concurrent.atomic.AtomicInteger(1)

    // Cleans and filters words in each line
    def preprocess(line: String): Array[String] = {
      if (line == null) return Array.empty[String]
      line.split(" ")
        .map(_.trim.toLowerCase)
```

```
    .filter(word => word.matches("^[A-Za-z]+$") && word.length >= 3)
}

// Task 1: count word frequency for each batch
val wordsStream: DStream[String] = lines.flatMap(preprocess)
wordsStream.foreachRDD { rdd =>
  if (!rdd.isEmpty()) {
    val counts = rdd.map(w => (w, 1)).reduceByKey(_ + _)
    val output = counts.map { case (w, c) => s"$w\t$c" }
    val seq = task1Seq.getAndIncrement()
    val path = f"$outputDir/task1-$seq%03d"
    println(s"Saving Task-1 output to $path")
    output.saveAsTextFile(path)
  }
}

// Task 2: count co-occurrence frequency per batch
val pairStream: DStream[((String, String), Int)] = lines.flatMap { line =>
  val words = preprocess(line)
  val n = words.length
  if (n <= 1) Seq.empty
  else {
    for {
      i <- 0 until n
      j <- 0 until n
      if i != j
    } yield ((words(i), words(j)), 1) // key = (w1, w2), value = 1
  }
}

pairStream.foreachRDD { rdd =>
  if (!rdd.isEmpty()) {
    val counts = rdd.reduceByKey(_ + _) // count co-occurrences
    val seq = task2Seq.getAndIncrement()
    val path = f"$outputDir/task2-$seq%03d"
    println(s"Saving Task-2 output to $path")
    // save as ((word1, word2), count)
    counts.saveAsTextFile(path)
  }
}


// Task 3:accumulate co-occurrence frequency across all batches
val updateFunc = (newValues: Seq[Int], prevState: Option[Int]) => {
  val newSum = newValues.sum + prevState.getOrElse(0)
  Some(newSum)
}
```

```
    val stateStream = pairStream.updateStateByKey[Int](updateFunc)

    stateStream.foreachRDD { rdd =>
      if (!rdd.isEmpty()) {
        val output = rdd.map { case (pair, c) => s"$pair\t$c" }
        val seq = task3Seq.getAndIncrement()
        val path = f"$outputDir/task3-$seq%03d"
        println(s"Saving Task-3 output to $path")
        output.saveAsTextFile(path)
      }
    }

    println("Streaming job started... waiting for new data in HDFS input directory.")
    ssc.start()
    ssc.awaitTermination()
  }
}
```