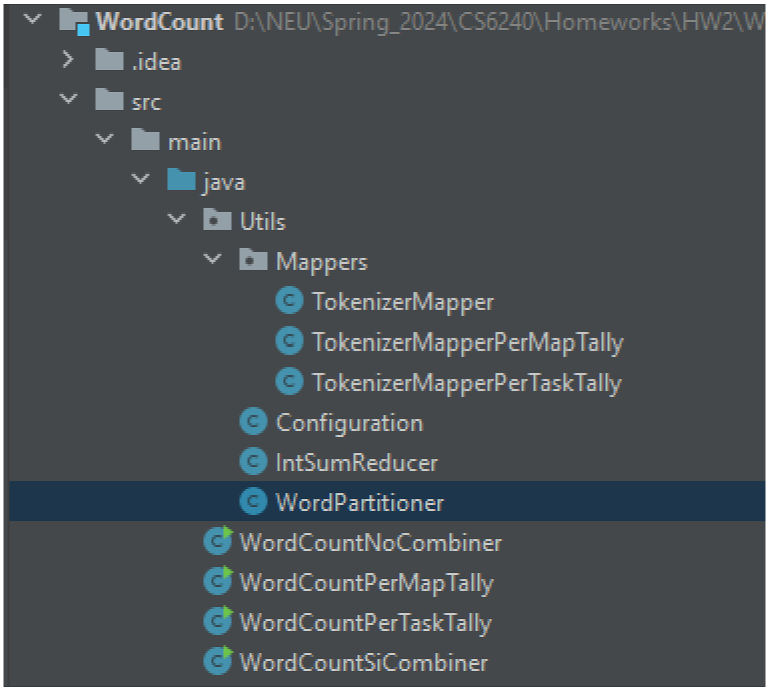
Name: Khoa Cu Dang (Cody) Cao

Course: CS6620 Spring 2024

Homework 2 Report

I. Souce Code

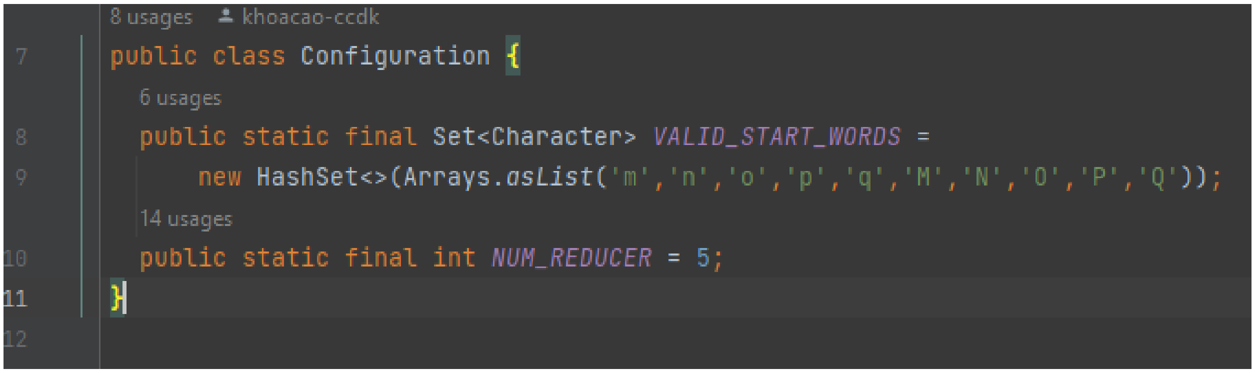
For code reuse purposes, I separated components into their own classes (Mapper, Reducer, Partitioner) then have different main classes utilize them as needed.



```
WordCount D:\NEU\Spring_2024\CS6240\Homeworks\HW2\W
  .idea
  src
    main
      java
        Utils
          Mappers
            TokenizerMapper
            TokenizerMapperPerMapTally
            TokenizerMapperPerTaskTally
          Configuration
          IntSumReducer
          WordPartitioner
        WordCountNoCombiner
        WordCountPerMapTally
        WordCountPerTaskTally
        WordCountSiCombiner
```

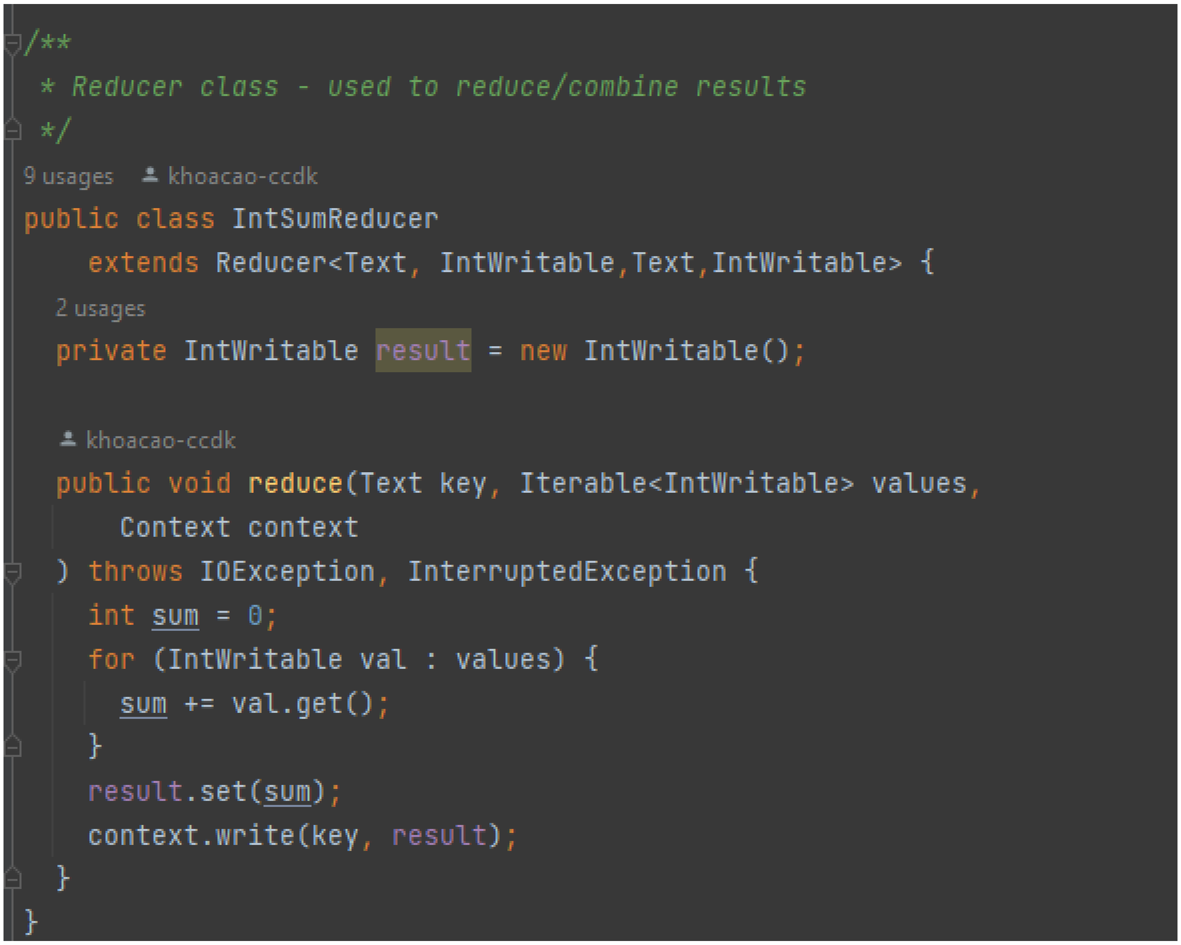
Common Utility Classes

Configuration: This class stores common configuration variables (list of acceptable starting characters, number of reducers being used)



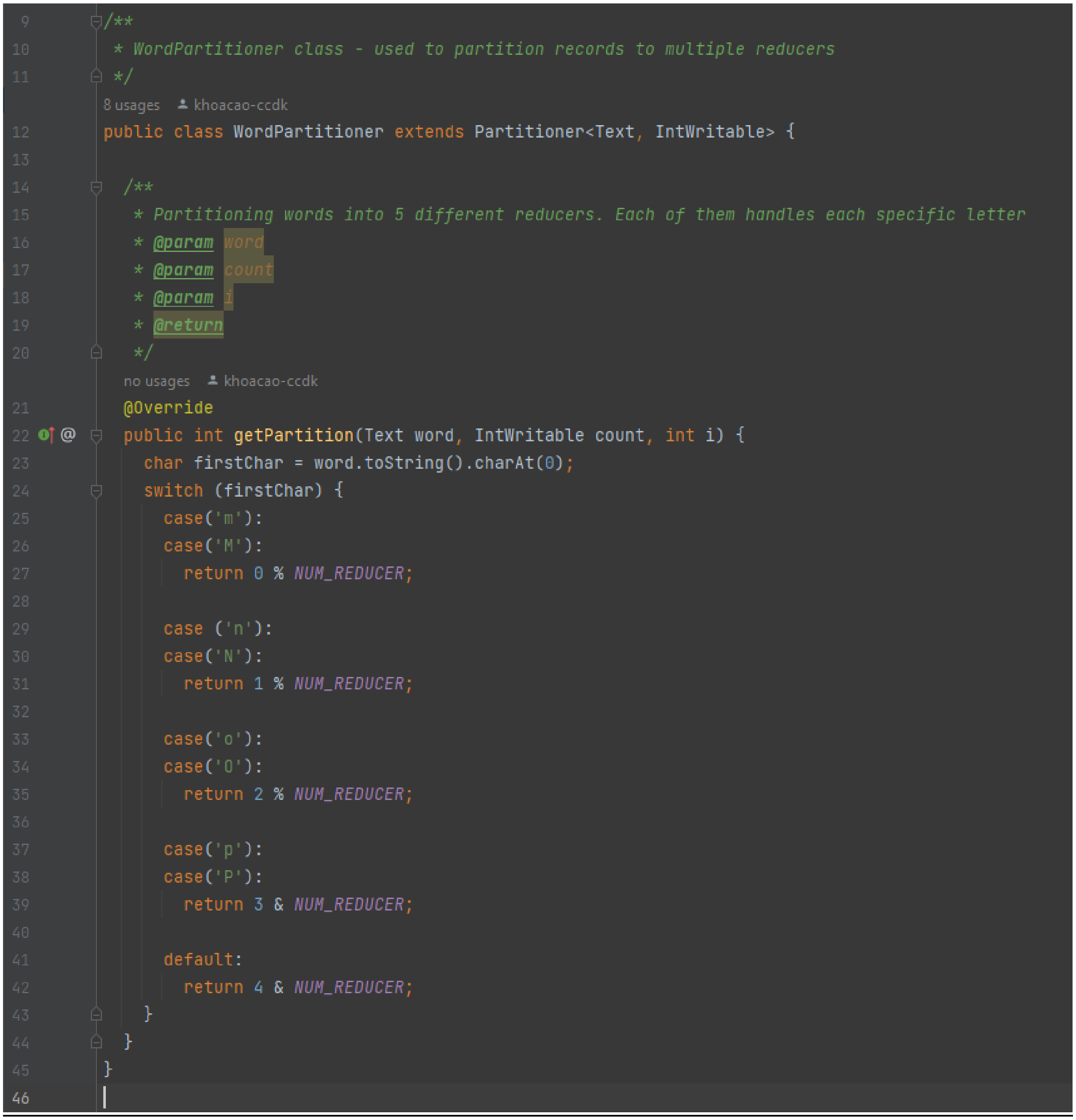
```
8 usages  khoacao-ccdk
public class Configuration {
  6 usages
  public static final Set<Character> VALID_START_WORDS =
      new HashSet<>(Arrays.asList('m','n','o','p','q','M','N','O','P','Q'));
  14 usages
  public static final int NUM_REDUCER = 5;
}
```

<u>IntSumReducer:</u> This class is used for Reducer and in-mapper Combiner tasks.



```java
/**
 * Reducer class - used to reduce/combine results
 */
public class IntSumReducer
    extends Reducer<Text, IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
      Context context
  ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}
```

WordPartitioner: This class is used to distribute key-value pairs emitted by mappers to reducers based on the word's starting character. Each reducer will handle a set of words that start with a specific character (both lowercase and uppercase).



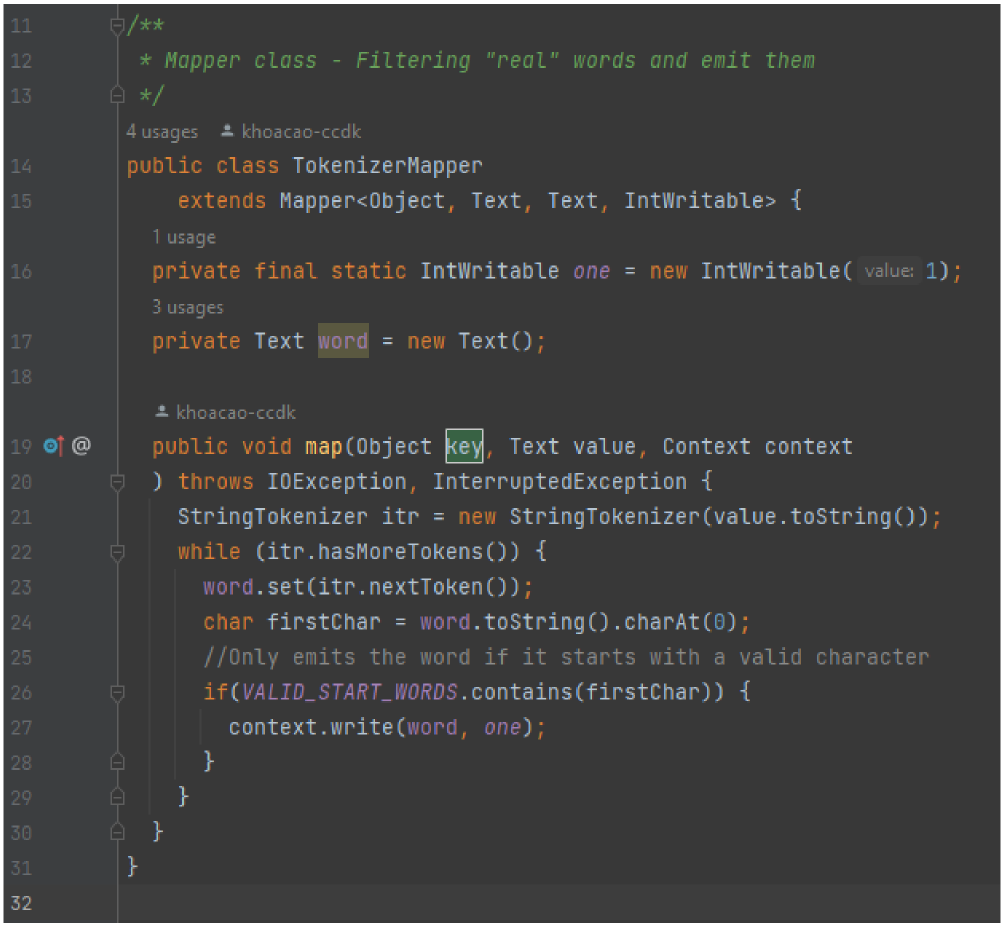
```
/**
 * WordPartitioner class - used to partition records to multiple reducers
 */
8 usages  khoacao-ccdk
public class WordPartitioner extends Partitioner<Text, IntWritable> {

  /**
   * Partitioning words into 5 different reducers. Each of them handles each specific letter
   * @param word
   * @param count
   * @param i
   * @return
   */
  no usages  khoacao-ccdk
  @Override
  public int getPartition(Text word, IntWritable count, int i) {
    char firstChar = word.toString().charAt(0);
    switch (firstChar) {
      case('m'):
      case('M'):
        return 0 % NUM_REDUCER;

      case ('n'):
      case('N'):
        return 1 % NUM_REDUCER;

      case('o'):
      case('O'):
        return 2 % NUM_REDUCER;

      case('p'):
      case('P'):
        return 3 & NUM_REDUCER;

      default:
        return 4 & NUM_REDUCER;
    }
  }
}
```

Mapper Classes

There are three main parameters for the map() function within Mapper classes:

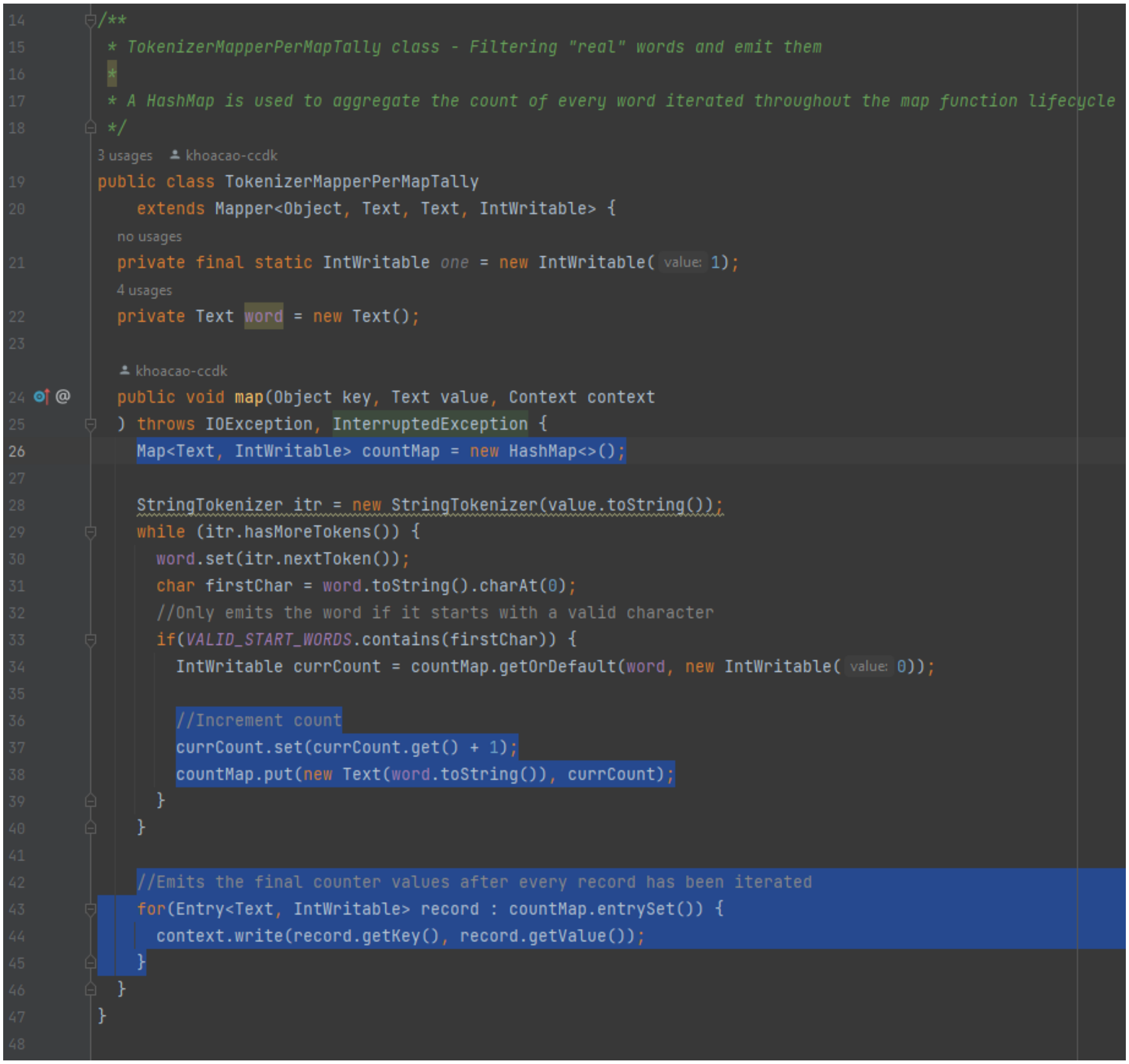
- Key – This parameter is often not used and/or set to null since in mapper phase, the focus is on processing the values. This variable is also not mentioned as being used by Hadoop documentation.
- Value – From the document of Hadoop, this Text value contains a line of the file. This then gets broken down further into separate words (separated by space) using StringTokenizer class. Thus, each call of the map() function will handle one line of the input file.
- Context: This object allows mappers to interact with the rest of the Hadoop system, as explained by a StackOverflow post. This explains the context.write() function being used to emit records.

TokenizerMapper: This is the default implementation of the WordCount example given by Hadoop. I went ahead and added an additional condition that the words being processed must starts with a valid alphabetic character in order to be emitted.



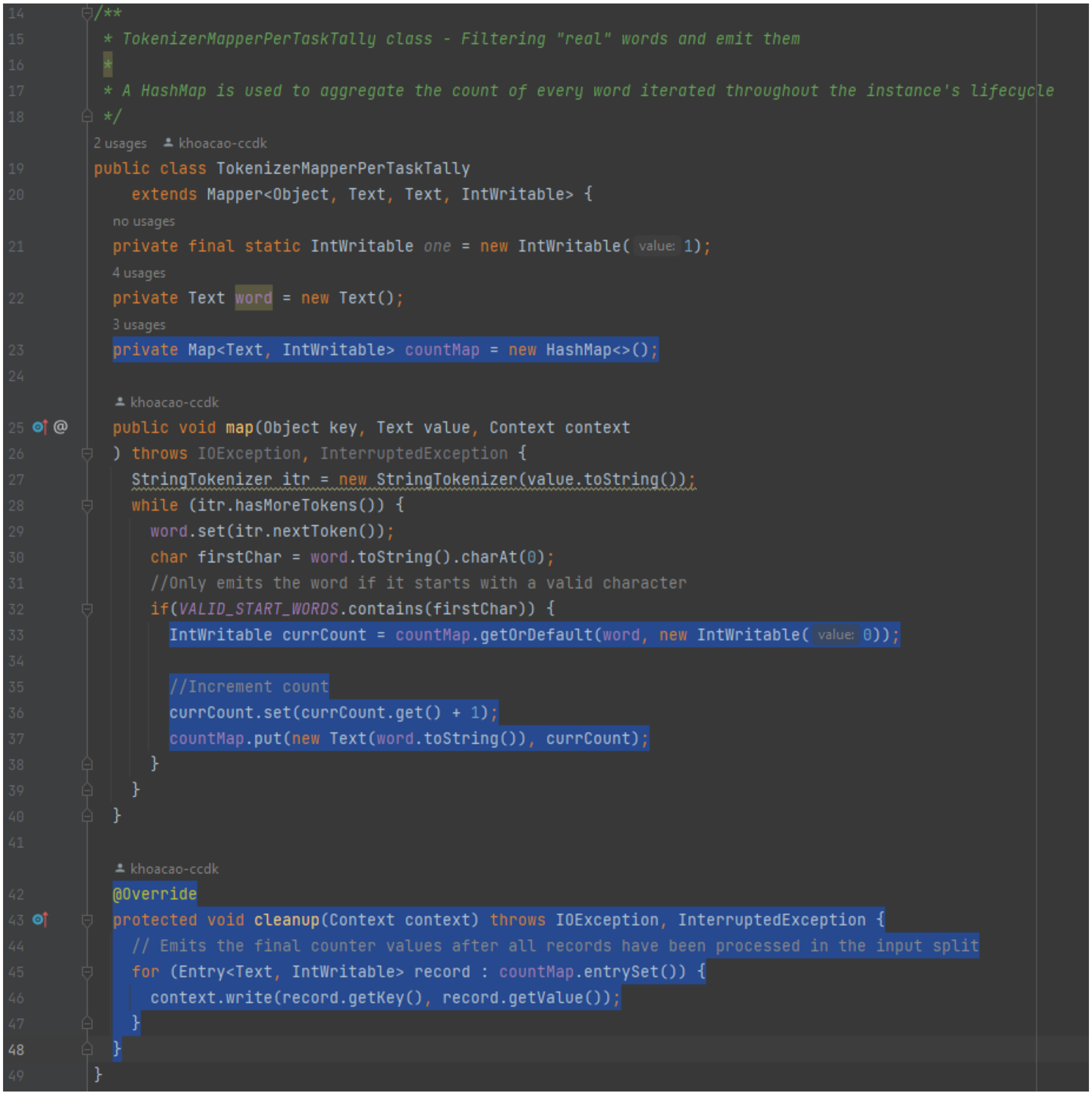
```java
/**
 * Mapper class - Filtering "real" words and emit them
 */
public class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(Object key, Text value, Context context
  ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      char firstChar = word.toString().charAt(0);
      //Only emits the word if it starts with a valid character
      if(VALID_START_WORDS.contains(firstChar)) {
        context.write(word, one);
      }
    }
  }
}
```

TokenizerMapperPerMapTally: With this per map tally implementation, the emitting record will be held off until the map has gone through every word in the input line. Aggregation is also performed using HashMap in case a word appears more than once in a single line.



```
/**
 * TokenizerMapperPerMapTally class - Filtering "real" words and emit them
 *
 * A HashMap is used to aggregate the count of every word iterated throughout the map function lifecycle
 */
3 usages  khoacao-ccdk
public class TokenizerMapperPerMapTally
    extends Mapper<Object, Text, Text, IntWritable> {
  no usages
  private final static IntWritable one = new IntWritable( value: 1);
  4 usages
  private Text word = new Text();

  khoacao-ccdk
  public void map(Object key, Text value, Context context
  ) throws IOException, InterruptedException {
    Map<Text, IntWritable> countMap = new HashMap<>();

    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      char firstChar = word.toString().charAt(0);
      //Only emits the word if it starts with a valid character
      if(VALID_START_WORDS.contains(firstChar)) {
        IntWritable currCount = countMap.getOrDefault(word, new IntWritable( value: 0));

        //Increment count
        currCount.set(currCount.get() + 1);
        countMap.put(new Text(word.toString()), currCount);
      }
    }

    //Emits the final counter values after every record has been iterated
    for(Entry<Text, IntWritable> record : countMap.entrySet()) {
      context.write(record.getKey(), record.getValue());
    }
  }
}
```

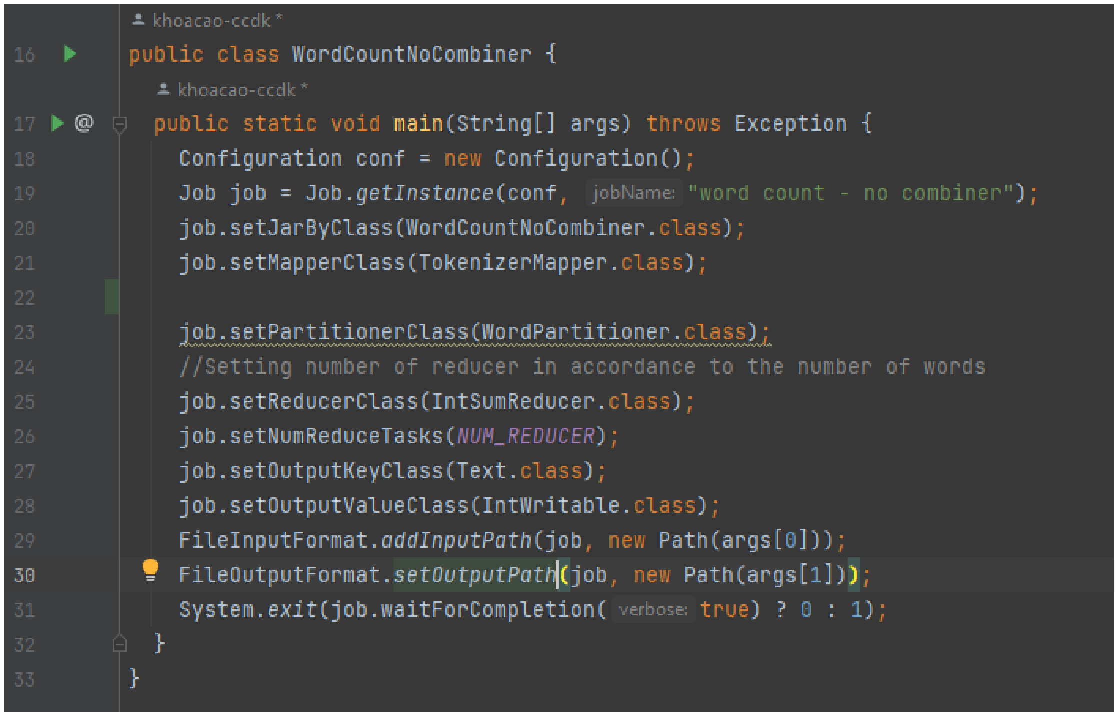
TokenizerMapperPerTaskTally: With this mapper implementation, I am having the aggregation at the mapper/input split level. Records are being held from emitting until the mapper has finished processing the whole input split. The records are aggregated using HashMap, then emitted using the cleanup() method of the mapper.



```java
/**
 * TokenizerMapperPerTaskTally class - Filtering "real" words and emit them
 *
 * A HashMap is used to aggregate the count of every word iterated throughout the instance's lifecycle
 */
public class TokenizerMapperPerTaskTally
    extends Mapper<Object, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();
  private Map<Text, IntWritable> countMap = new HashMap<>();

  public void map(Object key, Text value, Context context
  ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      char firstChar = word.toString().charAt(0);
      //Only emits the word if it starts with a valid character
      if(VALID_START_WORDS.contains(firstChar)) {
        IntWritable currCount = countMap.getOrDefault(word, new IntWritable(0));

        //Increment count
        currCount.set(currCount.get() + 1);
        countMap.put(new Text(word.toString()), currCount);
      }
    }
  }

  @Override
  protected void cleanup(Context context) throws IOException, InterruptedException {
    // Emits the final counter values after all records have been processed in the input split
    for (Entry<Text, IntWritable> record : countMap.entrySet()) {
      context.write(record.getKey(), record.getValue());
    }
  }
}
```

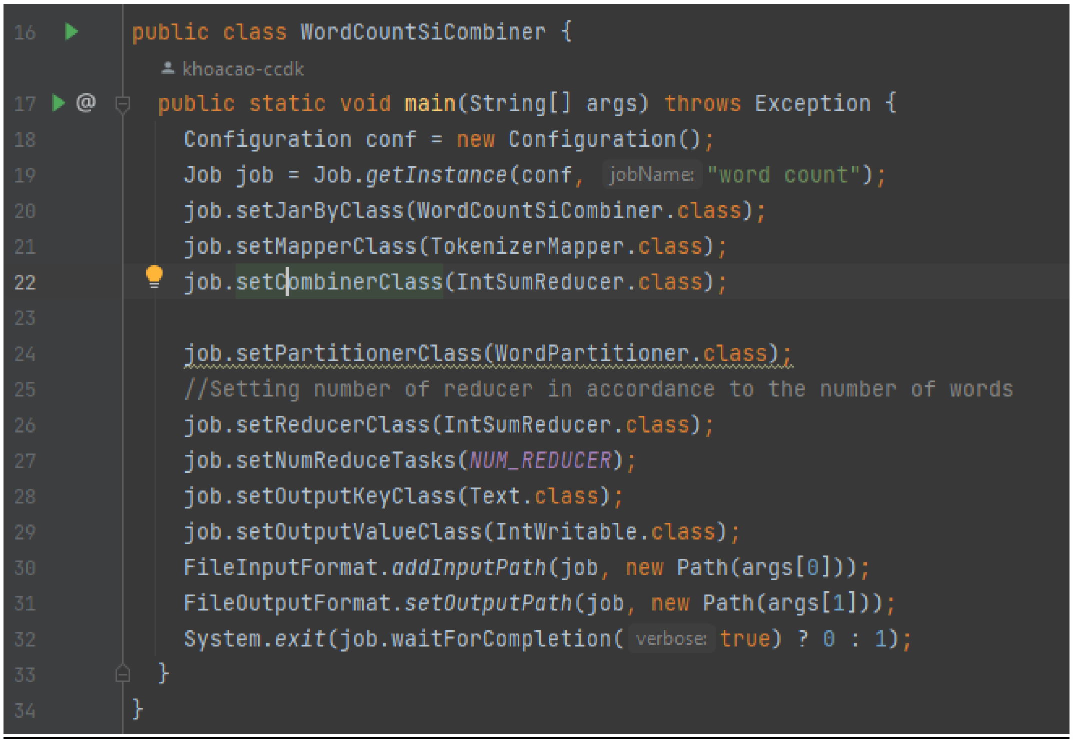
Job Classes

<u>WordCountNocombiner</u>



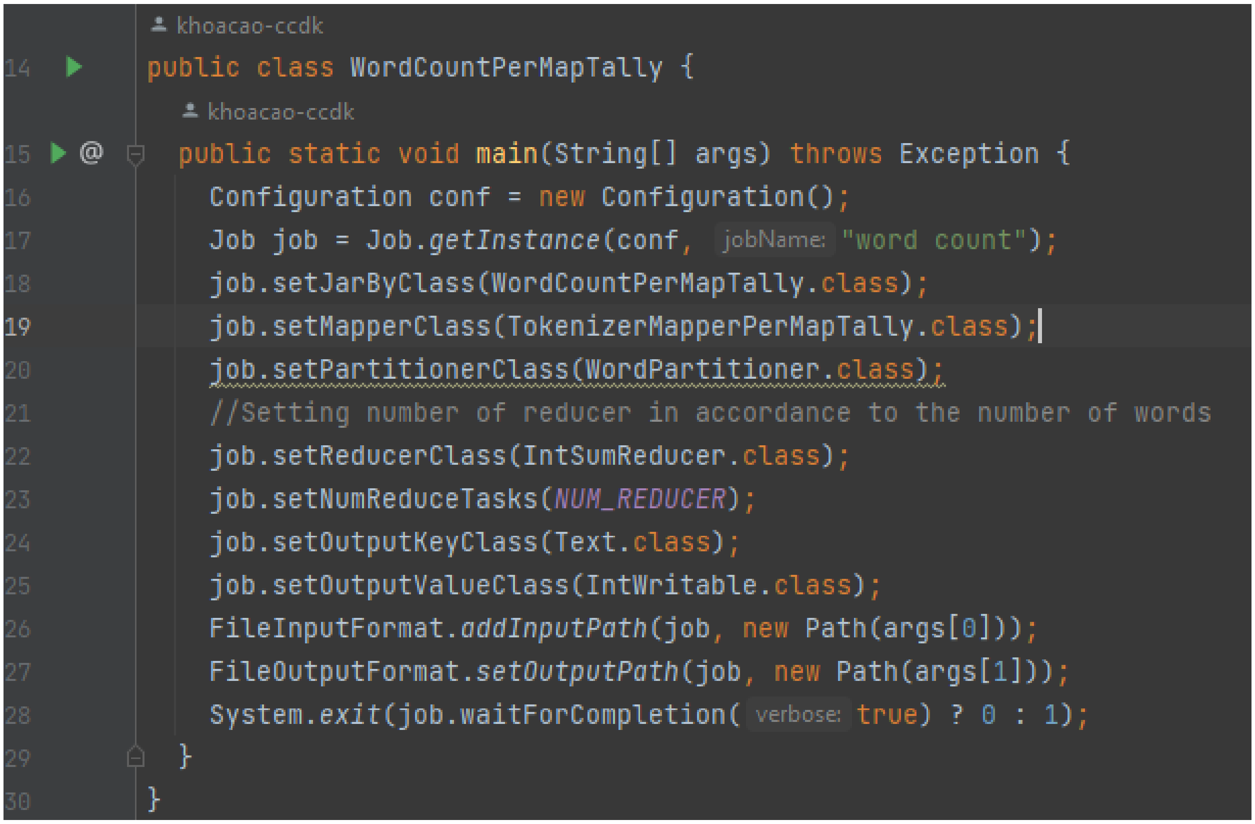
```
khoacao-ccdk *
public class WordCountNoCombiner {
    khoacao-ccdk *
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, jobName: "word count - no combiner");
        job.setJarByClass(WordCountNoCombiner.class);
        job.setMapperClass(TokenizerMapper.class);

        job.setPartitionerClass(WordPartitioner.class);
        //Setting number of reducer in accordance to the number of words
        job.setReducerClass(IntSumReducer.class);
        job.setNumReduceTasks(NUM_REDUCER);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion( verbose: true) ? 0 : 1);
    }
}
```

<u>WordCountSiCombiner</u>



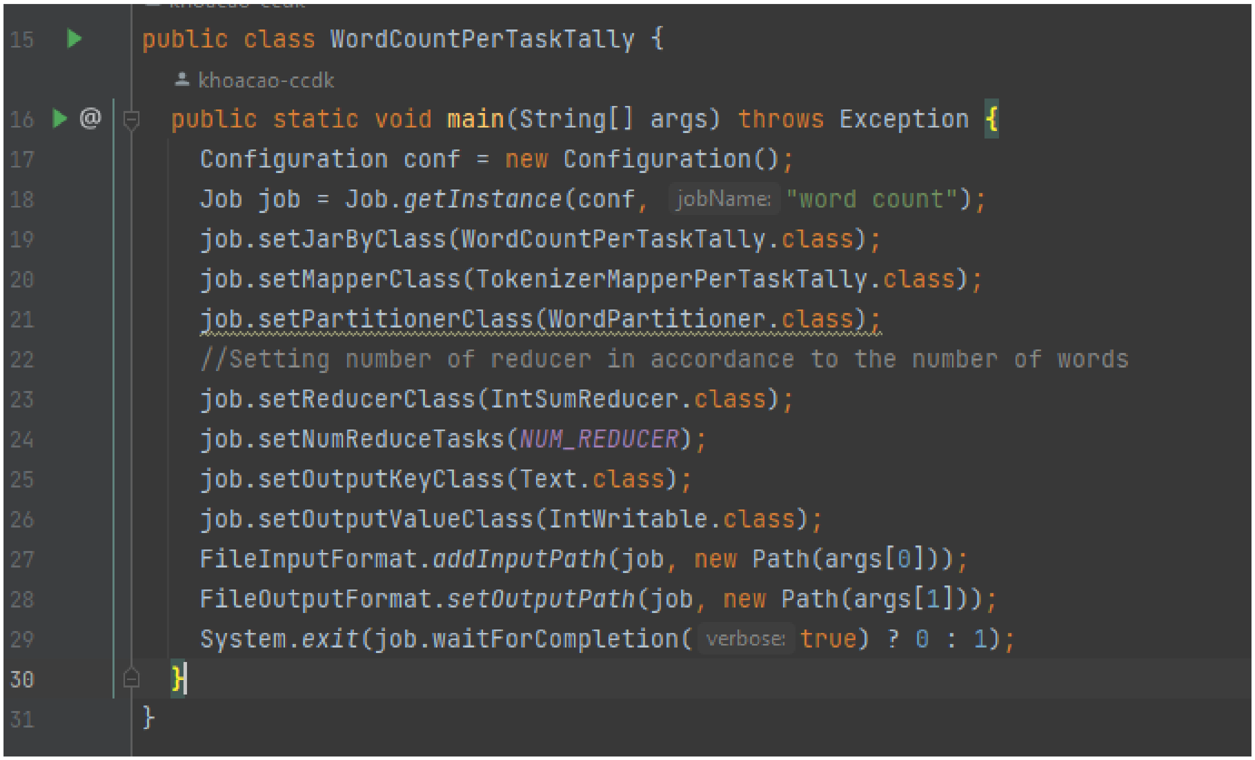
```
public class WordCountSiCombiner {
    khoacao-ccdk
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, jobName: "word count");
        job.setJarByClass(WordCountSiCombiner.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);

        job.setPartitionerClass(WordPartitioner.class);
        //Setting number of reducer in accordance to the number of words
        job.setReducerClass(IntSumReducer.class);
        job.setNumReduceTasks(NUM_REDUCER);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion( verbose: true) ? 0 : 1);
    }
}
```

WordCountPerMapTally



```
khoacao-ccdk
public class WordCountPerMapTally {
  khoacao-ccdk
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, jobName: "word count");
    job.setJarByClass(WordCountPerMapTally.class);
    job.setMapperClass(TokenizerMapperPerMapTally.class);
    job.setPartitionerClass(WordPartitioner.class);
    //Setting number of reducer in accordance to the number of words
    job.setReducerClass(IntSumReducer.class);
    job.setNumReduceTasks(NUM_REDUCER);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion( verbose: true) ? 0 : 1);
  }
}
```

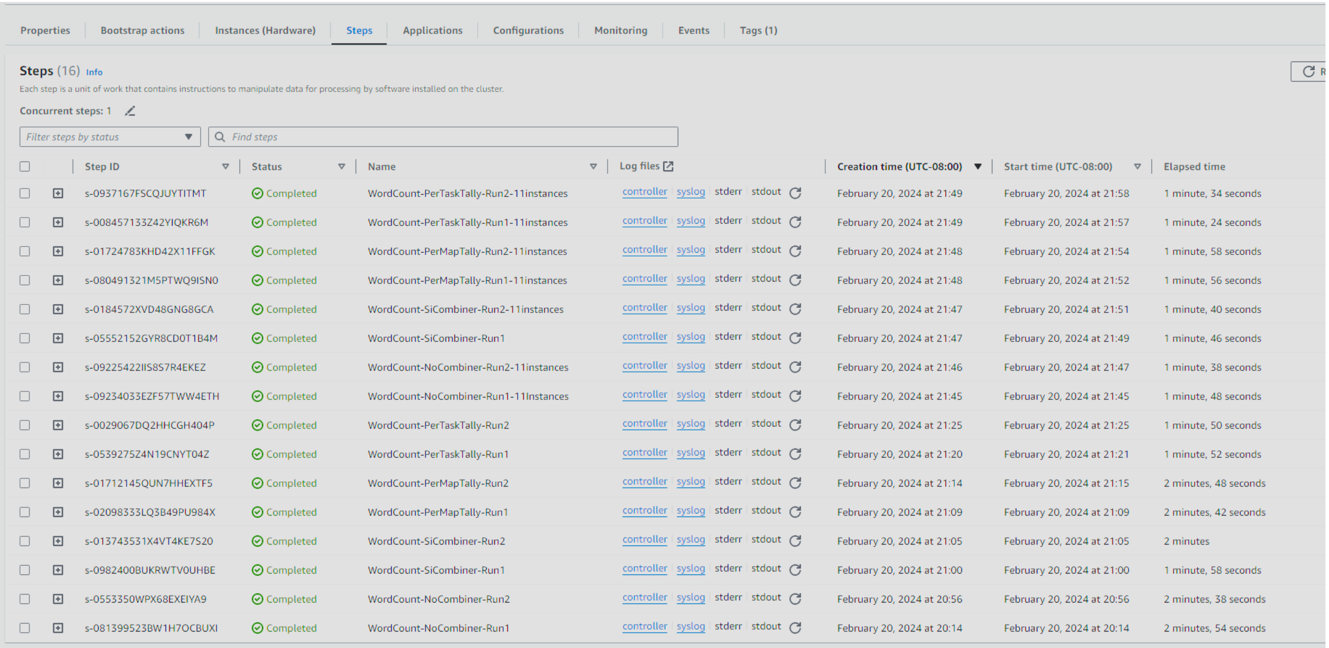
WordCountPerTaskTally



```
public class WordCountPerTaskTally {
  khoacao-ccdk
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, jobName: "word count");
    job.setJarByClass(WordCountPerTaskTally.class);
    job.setMapperClass(TokenizerMapperPerTaskTally.class);
    job.setPartitionerClass(WordPartitioner.class);
    //Setting number of reducer in accordance to the number of words
    job.setReducerClass(IntSumReducer.class);
    job.setNumReduceTasks(NUM_REDUCER);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion( verbose: true) ? 0 : 1);
  }
}
```

II. Performance Comparison

Record of 16 Steps being run on AWS. After the first 8 steps, I edited the cluster configuration to 10 core nodes and 1 master node.



Properties | Bootstrap actions | Instances (Hardware) | Steps | Applications | Configurations | Monitoring | Events | Tags (1)

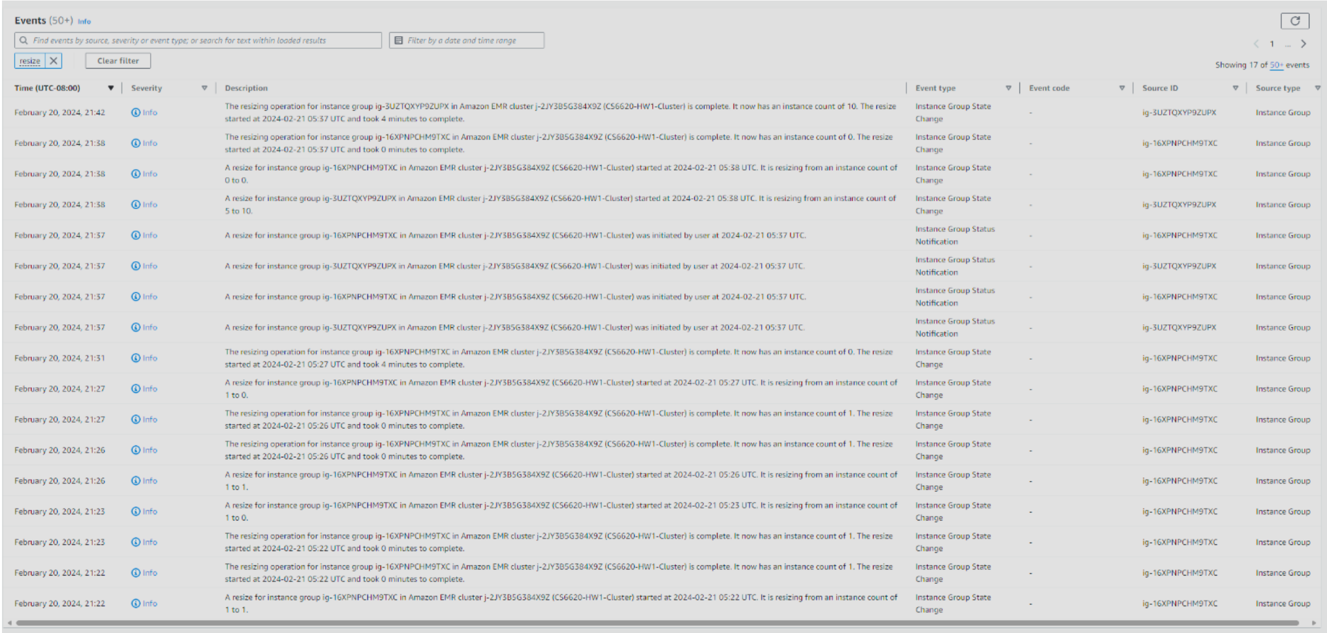
Steps (16) Info

Each step is a unit of work that contains instructions to manipulate data for processing by software installed on the cluster.

Concurrent steps: 1

Filter steps by status ▼ | Find steps

☐	Step ID	Status	Name	Log files	Creation time (UTC-08:00)	Start time (UTC-08:00)	Elapsed time
☐	s-0937167FSCQJUYTITMT	Completed	WordCount-PerTaskTally-Run2-11instances	controller | syslog | stderr | stdout	February 20, 2024 at 21:49	February 20, 2024 at 21:58	1 minute, 34 seconds
☐	s-008457133Z42YIQKR6M	Completed	WordCount-PerTaskTally-Run1-11instances	controller | syslog | stderr | stdout	February 20, 2024 at 21:49	February 20, 2024 at 21:57	1 minute, 24 seconds
☐	s-01724783KHD42X11FFGK	Completed	WordCount-PerMapTally-Run2-11instances	controller | syslog | stderr | stdout	February 20, 2024 at 21:48	February 20, 2024 at 21:54	1 minute, 58 seconds
☐	s-080491321M5PTWQ9ISN0	Completed	WordCount-PerMapTally-Run1-11instances	controller | syslog | stderr | stdout	February 20, 2024 at 21:48	February 20, 2024 at 21:52	1 minute, 56 seconds
☐	s-0184572XVD48GNG8GCA	Completed	WordCount-SiCombiner-Run2-11instances	controller | syslog | stderr | stdout	February 20, 2024 at 21:47	February 20, 2024 at 21:51	1 minute, 40 seconds
☐	s-05552152GYR8CD0T1B4M	Completed	WordCount-SiCombiner-Run1	controller | syslog | stderr | stdout	February 20, 2024 at 21:47	February 20, 2024 at 21:49	1 minute, 46 seconds
☐	s-09225422IIS8S7R4EKEZ	Completed	WordCount-NoCombiner-Run2-11instances	controller | syslog | stderr | stdout	February 20, 2024 at 21:46	February 20, 2024 at 21:47	1 minute, 38 seconds
☐	s-09234033EZF57TWW4ETH	Completed	WordCount-NoCombiner-Run1-11Instances	controller | syslog | stderr | stdout	February 20, 2024 at 21:45	February 20, 2024 at 21:45	1 minute, 48 seconds
☐	s-0029067DQ2HHCGH404P	Completed	WordCount-PerTaskTally-Run2	controller | syslog | stderr | stdout	February 20, 2024 at 21:25	February 20, 2024 at 21:25	1 minute, 50 seconds
☐	s-0539275Z4N19CNYT04Z	Completed	WordCount-PerTaskTally-Run1	controller | syslog | stderr | stdout	February 20, 2024 at 21:20	February 20, 2024 at 21:21	1 minute, 52 seconds
☐	s-01712145QUN7HHEXTF5	Completed	WordCount-PerMapTally-Run2	controller | syslog | stderr | stdout	February 20, 2024 at 21:14	February 20, 2024 at 21:15	2 minutes, 48 seconds
☐	s-02098333LQ3B49PU984X	Completed	WordCount-PerMapTally-Run1	controller | syslog | stderr | stdout	February 20, 2024 at 21:09	February 20, 2024 at 21:09	2 minutes, 42 seconds
☐	s-013743531X4VT4KE7S20	Completed	WordCount-SiCombiner-Run2	controller | syslog | stderr | stdout	February 20, 2024 at 21:05	February 20, 2024 at 21:05	2 minutes
☐	s-0982400BUKRWTV0UHBE	Completed	WordCount-SiCombiner-Run1	controller | syslog | stderr | stdout	February 20, 2024 at 21:00	February 20, 2024 at 21:00	1 minute, 58 seconds
☐	s-0553350WPX68EXEIYA9	Completed	WordCount-NoCombiner-Run2	controller | syslog | stderr | stdout	February 20, 2024 at 20:56	February 20, 2024 at 20:56	2 minutes, 38 seconds
☐	s-081399523BW1H7OCBUXI	Completed	WordCount-NoCombiner-Run1	controller | syslog | stderr | stdout	February 20, 2024 at 20:14	February 20, 2024 at 20:14	2 minutes, 54 seconds



Events (50+) Info

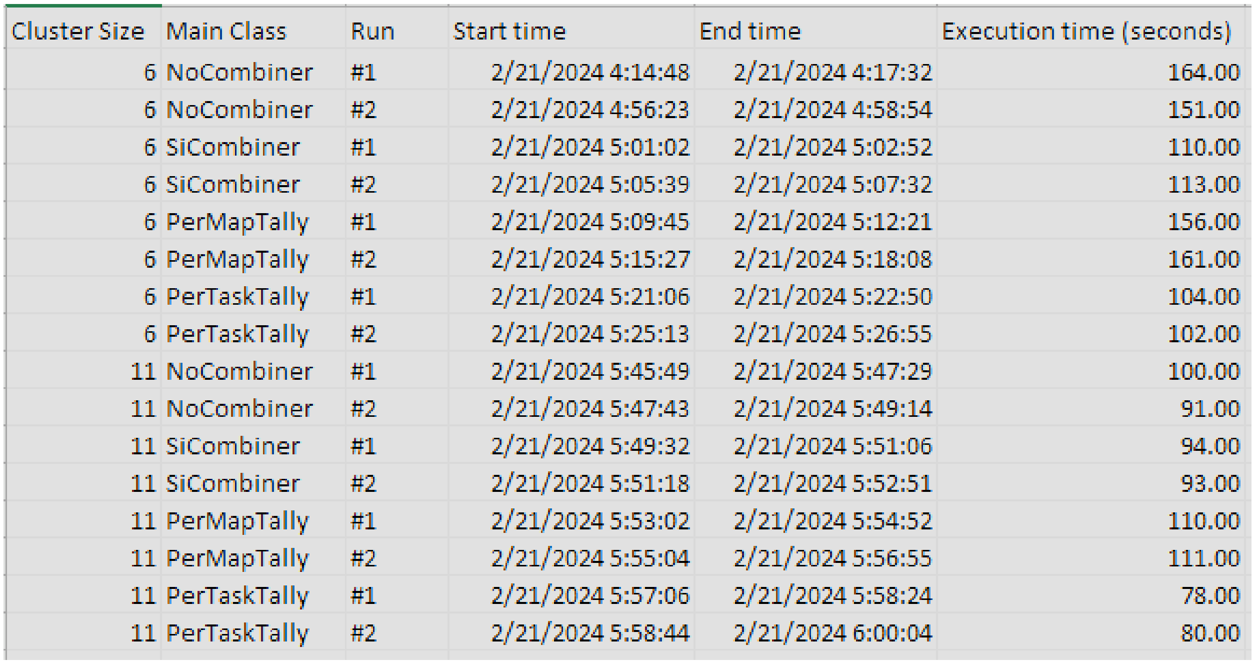
Find events by source, severity or event type; or search for text within loaded results | Filter by a date and time range

resize ✕ | Clear filter

< 1 ... >

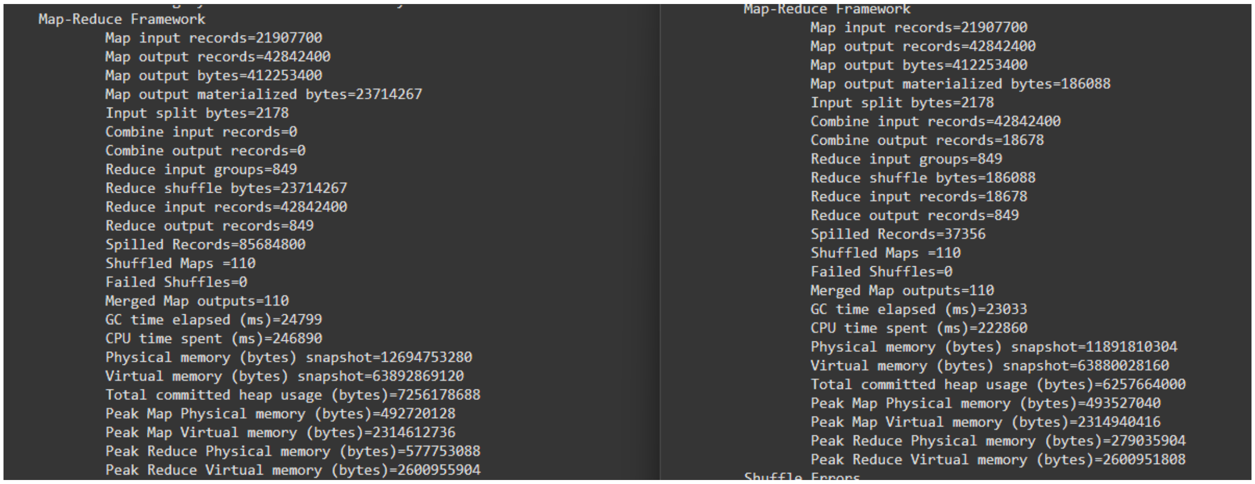
Showing 17 of 50+ events

Time (UTC-08:00)	Severity	Description	Event type	Event code	Source ID	Source type
February 20, 2024, 21:42	Info	The resizing operation for instance group ig-3UZTQXYP9ZUPX in Amazon EMR cluster j-2JY3B5G384X9Z (CS6620-HW1-Cluster) is complete. It now has an instance count of 10. The resize started at 2024-02-21 05:37 UTC and took 4 minutes to complete.	Instance Group State Change	-	ig-3UZTQXYP9ZUPX	Instance Group
February 20, 2024, 21:38	Info	The resizing operation for instance group ig-16XPNPCHM9TXC in Amazon EMR cluster j-2JY3B5G384X9Z (CS6620-HW1-Cluster) is complete. It now has an instance count of 0. The resize started at 2024-02-21 05:37 UTC and took 0 minutes to complete.	Instance Group State Change	-	ig-16XPNPCHM9TXC	Instance Group
February 20, 2024, 21:38	Info	A resize for instance group ig-16XPNPCHM9TXC in Amazon EMR cluster j-2JY3B5G384X9Z (CS6620-HW1-Cluster) started at 2024-02-21 05:38 UTC. It is resizing from an instance count of 0 to 0.	Instance Group State Change	-	ig-16XPNPCHM9TXC	Instance Group
February 20, 2024, 21:38	Info	A resize for instance group ig-3UZTQXYP9ZUPX in Amazon EMR cluster j-2JY3B5G384X9Z (CS6620-HW1-Cluster) started at 2024-02-21 05:38 UTC. It is resizing from an instance count of 5 to 10.	Instance Group State Change	-	ig-3UZTQXYP9ZUPX	Instance Group
February 20, 2024, 21:37	Info	A resize for instance group ig-16XPNPCHM9TXC in Amazon EMR cluster j-2JY3B5G384X9Z (CS6620-HW1-Cluster) was initiated by user at 2024-02-21 05:37 UTC.	Instance Group Status Notification	-	ig-16XPNPCHM9TXC	Instance Group
February 20, 2024, 21:37	Info	A resize for instance group ig-3UZTQXYP9ZUPX in Amazon EMR cluster j-2JY3B5G384X9Z (CS6620-HW1-Cluster) was initiated by user at 2024-02-21 05:37 UTC.	Instance Group Status Notification	-	ig-3UZTQXYP9ZUPX	Instance Group
February 20, 2024, 21:37	Info	A resize for instance group ig-16XPNPCHM9TXC in Amazon EMR cluster j-2JY3B5G384X9Z (CS6620-HW1-Cluster) was initiated by user at 2024-02-21 05:37 UTC.	Instance Group Status Notification	-	ig-16XPNPCHM9TXC	Instance Group
February 20, 2024, 21:37	Info	A resize for instance group ig-3UZTQXYP9ZUPX in Amazon EMR cluster j-2JY3B5G384X9Z (CS6620-HW1-Cluster) was initiated by user at 2024-02-21 05:37 UTC.	Instance Group Status Notification	-	ig-3UZTQXYP9ZUPX	Instance Group
February 20, 2024, 21:31	Info	The resizing operation for instance group ig-16XPNPCHM9TXC in Amazon EMR cluster j-2JY3B5G384X9Z (CS6620-HW1-Cluster) is complete. It now has an instance count of 0. The resize started at 2024-02-21 05:27 UTC and took 4 minutes to complete.	Instance Group State Change	-	ig-16XPNPCHM9TXC	Instance Group
February 20, 2024, 21:27	Info	A resize for instance group ig-16XPNPCHM9TXC in Amazon EMR cluster j-2JY3B5G384X9Z (CS6620-HW1-Cluster) started at 2024-02-21 05:27 UTC. It is resizing from an instance count of 1 to 0.	Instance Group State Change	-	ig-16XPNPCHM9TXC	Instance Group
February 20, 2024, 21:27	Info	The resizing operation for instance group ig-16XPNPCHM9TXC in Amazon EMR cluster j-2JY3B5G384X9Z (CS6620-HW1-Cluster) is complete. It now has an instance count of 1. The resize started at 2024-02-21 05:26 UTC and took 0 minutes to complete.	Instance Group State Change	-	ig-16XPNPCHM9TXC	Instance Group
February 20, 2024, 21:26	Info	The resizing operation for instance group ig-16XPNPCHM9TXC in Amazon EMR cluster j-2JY3B5G384X9Z (CS6620-HW1-Cluster) is complete. It now has an instance count of 1. The resize started at 2024-02-21 05:26 UTC and took 0 minutes to complete.	Instance Group State Change	-	ig-16XPNPCHM9TXC	Instance Group
February 20, 2024, 21:26	Info	A resize for instance group ig-16XPNPCHM9TXC in Amazon EMR cluster j-2JY3B5G384X9Z (CS6620-HW1-Cluster) started at 2024-02-21 05:26 UTC. It is resizing from an instance count of 1 to 1.	Instance Group State Change	-	ig-16XPNPCHM9TXC	Instance Group
February 20, 2024, 21:23	Info	A resize for instance group ig-16XPNPCHM9TXC in Amazon EMR cluster j-2JY3B5G384X9Z (CS6620-HW1-Cluster) started at 2024-02-21 05:23 UTC. It is resizing from an instance count of 1 to 0.	Instance Group State Change	-	ig-16XPNPCHM9TXC	Instance Group
February 20, 2024, 21:23	Info	The resizing operation for instance group ig-16XPNPCHM9TXC in Amazon EMR cluster j-2JY3B5G384X9Z (CS6620-HW1-Cluster) is complete. It now has an instance count of 1. The resize started at 2024-02-21 05:22 UTC and took 0 minutes to complete.	Instance Group State Change	-	ig-16XPNPCHM9TXC	Instance Group
February 20, 2024, 21:22	Info	The resizing operation for instance group ig-16XPNPCHM9TXC in Amazon EMR cluster j-2JY3B5G384X9Z (CS6620-HW1-Cluster) is complete. It now has an instance count of 1. The resize started at 2024-02-21 05:22 UTC and took 0 minutes to complete.	Instance Group State Change	-	ig-16XPNPCHM9TXC	Instance Group
February 20, 2024, 21:22	Info	A resize for instance group ig-16XPNPCHM9TXC in Amazon EMR cluster j-2JY3B5G384X9Z (CS6620-HW1-Cluster) started at 2024-02-21 05:22 UTC. It is resizing from an instance count of 1 to 1.	Instance Group State Change	-	ig-16XPNPCHM9TXC	Instance Group



Cluster Size	Main Class	Run	Start time	End time	Execution time (seconds)
6	NoCombiner	#1	2/21/2024 4:14:48	2/21/2024 4:17:32	164.00
6	NoCombiner	#2	2/21/2024 4:56:23	2/21/2024 4:58:54	151.00
6	SiCombiner	#1	2/21/2024 5:01:02	2/21/2024 5:02:52	110.00
6	SiCombiner	#2	2/21/2024 5:05:39	2/21/2024 5:07:32	113.00
6	PerMapTally	#1	2/21/2024 5:09:45	2/21/2024 5:12:21	156.00
6	PerMapTally	#2	2/21/2024 5:15:27	2/21/2024 5:18:08	161.00
6	PerTaskTally	#1	2/21/2024 5:21:06	2/21/2024 5:22:50	104.00
6	PerTaskTally	#2	2/21/2024 5:25:13	2/21/2024 5:26:55	102.00
11	NoCombiner	#1	2/21/2024 5:45:49	2/21/2024 5:47:29	100.00
11	NoCombiner	#2	2/21/2024 5:47:43	2/21/2024 5:49:14	91.00
11	SiCombiner	#1	2/21/2024 5:49:32	2/21/2024 5:51:06	94.00
11	SiCombiner	#2	2/21/2024 5:51:18	2/21/2024 5:52:51	93.00
11	PerMapTally	#1	2/21/2024 5:53:02	2/21/2024 5:54:52	110.00
11	PerMapTally	#2	2/21/2024 5:55:04	2/21/2024 5:56:55	111.00
11	PerTaskTally	#1	2/21/2024 5:57:06	2/21/2024 5:58:24	78.00
11	PerTaskTally	#2	2/21/2024 5:58:44	2/21/2024 6:00:04	80.00

- **Do you believe the combiner was called at all in program SiCombiner?**
- **What difference did the use of a combiner make in SiCombiner compared to NoCombiner?**



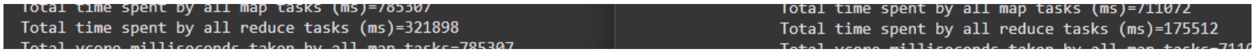
```
Map-Reduce Framework
        Map input records=21907700
        Map output records=42842400
        Map output bytes=412253400
        Map output materialized bytes=23714267
        Input split bytes=2178
        Combine input records=0
        Combine output records=0
        Reduce input groups=849
        Reduce shuffle bytes=23714267
        Reduce input records=42842400
        Reduce output records=849
        Spilled Records=85684800
        Shuffled Maps =110
        Failed Shuffles=0
        Merged Map outputs=110
        GC time elapsed (ms)=24799
        CPU time spent (ms)=246890
        Physical memory (bytes) snapshot=12694753280
        Virtual memory (bytes) snapshot=63892869120
        Total committed heap usage (bytes)=7256178688
        Peak Map Physical memory (bytes)=492720128
        Peak Map Virtual memory (bytes)=2314612736
        Peak Reduce Physical memory (bytes)=577753088
        Peak Reduce Virtual memory (bytes)=2600955904
```

```
Map-Reduce Framework
        Map input records=21907700
        Map output records=42842400
        Map output bytes=412253400
        Map output materialized bytes=186088
        Input split bytes=2178
        Combine input records=42842400
        Combine output records=18678
        Reduce input groups=849
        Reduce shuffle bytes=186088
        Reduce input records=18678
        Reduce output records=849
        Spilled Records=37356
        Shuffled Maps =110
        Failed Shuffles=0
        Merged Map outputs=110
        GC time elapsed (ms)=23033
        CPU time spent (ms)=222860
        Physical memory (bytes) snapshot=11891810304
        Virtual memory (bytes) snapshot=63880028160
        Total committed heap usage (bytes)=6257664000
        Peak Map Physical memory (bytes)=493527040
        Peak Map Virtual memory (bytes)=2314940416
        Peak Reduce Physical memory (bytes)=279035904
        Peak Reduce Virtual memory (bytes)=2600951808
Shuffle Errors
```

To answer both of the questions, we can look at the syslog for NoCombiner (left) and SiCombiner (right) run. When analyzing the syslog files, we can see that the combiner was definitely called in SiCombiner run due to:

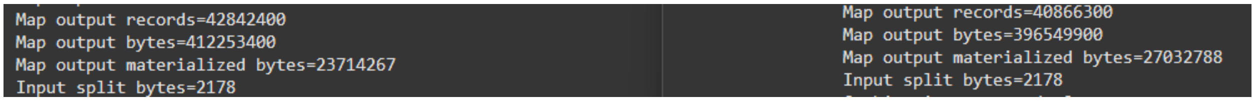
- Combine input records (0 on NoCombiner vs 42842400 on SiCombiner)
- Combine output records (0 on NoCombiner vs 18678 on SiCombiner)
- Reduce input records (42842400 on No Combiner vs 18678 on SiCombiner)

This proves that the combiner was used in the middle of the Map phase and the Reduce phase, which reduced the number of input records on the Reduce phase significantly by doing aggregation before emitting records to the reduce phase. Additionally, having less input records means it took less time for the reduce tasks to run (321898ms on NoCombiner vs 175512ms on SiCombiner)



```
Total time spent by all map tasks (ms)=785307                 Total time spent by all map tasks (ms)=711072
Total time spent by all reduce tasks (ms)=321898              Total time spent by all reduce tasks (ms)=175512
Total vcore-milliseconds taken by all map tasks=785307        Total vcore-milliseconds taken by all map tasks=711
```

- **Was the local aggregation effective in PerMapTally compared to NoCombiner?**



```
                                                    Map output records=40866300
Map output records=42842400                         Map output bytes=396549900
Map output bytes=412253400                          Map output materialized bytes=27032788
Map output materialized bytes=23714267              Input split bytes=2178
Input split bytes=2178
```

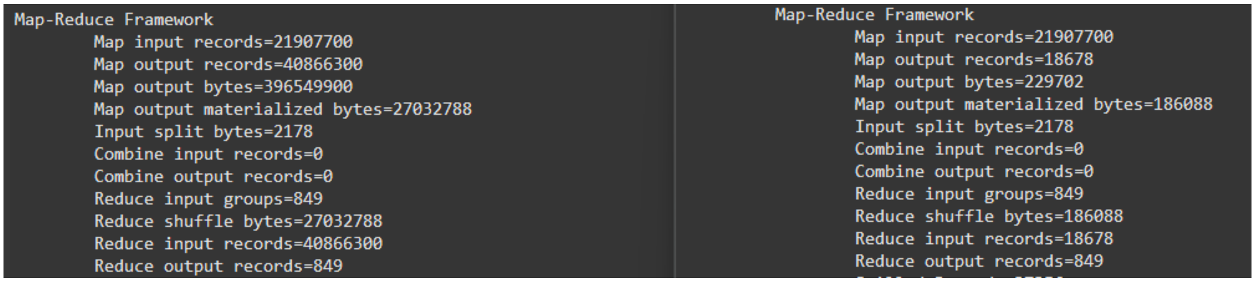
Local aggregation in PerMapTally (right) seems to be a little bit better compared to NoCombiner (left)

- It has a slightly lower map output record (40866300 on PerMapTally vs 42842400 on NoCombiner)

This might be due to the fact that for each line, there is actually not many repetitive words. Thus, having a HashMap storing word count for only one line (PerMapTally) might not be an effective method when considering extra memory and additional runtime required to provision and operate an additional HashMap.

- **What differences do you see between PerMapTally and PerTaskTally? Try to**

explain the reasons.

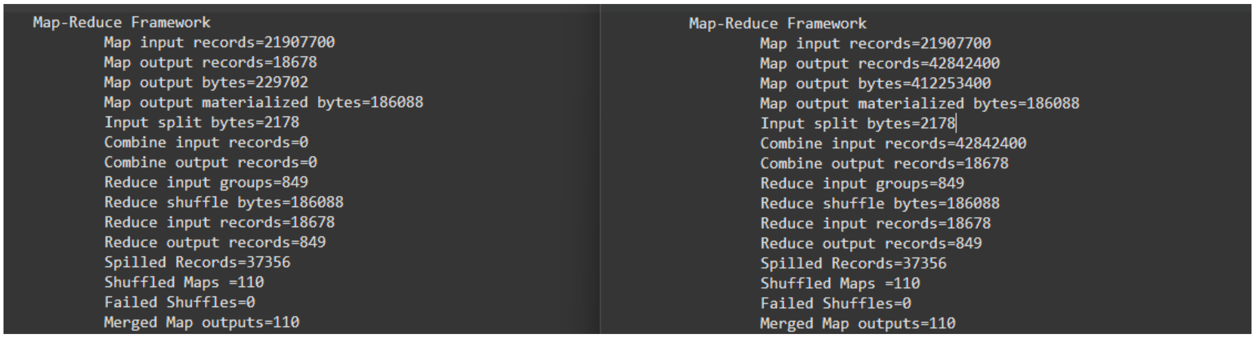


```
Map-Reduce Framework                                    Map-Reduce Framework
        Map input records=21907700                              Map input records=21907700
        Map output records=40866300                             Map output records=18678
        Map output bytes=396549900                              Map output bytes=229702
        Map output materialized bytes=27032788                  Map output materialized bytes=186088
        Input split bytes=2178                                  Input split bytes=2178
        Combine input records=0                                 Combine input records=0
        Combine output records=0                                Combine output records=0
        Reduce input groups=849                                 Reduce input groups=849
        Reduce shuffle bytes=27032788                           Reduce shuffle bytes=186088
        Reduce input records=40866300                           Reduce input records=18678
        Reduce output records=849                               Reduce output records=849
```

When looking at the syslog of PerMapTally (left) vs PerTaskTally (right), there is a significant different in the Map out put record number (40866300 on PerMapTally vs 186088 on PerTaskTally), resulting in a significantly lower Reduce input records number. This can be because of the nature of the input file:

- Within a line of the input file, there is a fewer number of repetitive words. Thus, the HashMap provisioned in PerMapTally would have less value in aggregating the values.
- However, when we consider a split of the input file, there is a larger number of repetitive words. In this scenario, a HashMap provisioned at a mapper instance level (PerTaskTally) would help tremendously with aggregating records and reducing the number of output records.

- **Which one is better: SiCombiner or PerTaskTally? Briefly justify your answer.**



```
Map-Reduce Framework
        Map input records=21907700
        Map output records=18678
        Map output bytes=229702
        Map output materialized bytes=186088
        Input split bytes=2178
        Combine input records=0
        Combine output records=0
        Reduce input groups=849
        Reduce shuffle bytes=186088
        Reduce input records=18678
        Reduce output records=849
        Spilled Records=37356
        Shuffled Maps =110
        Failed Shuffles=0
        Merged Map outputs=110
```

```
Map-Reduce Framework
        Map input records=21907700
        Map output records=42842400
        Map output bytes=412253400
        Map output materialized bytes=186088
        Input split bytes=2178
        Combine input records=42842400
        Combine output records=18678
        Reduce input groups=849
        Reduce shuffle bytes=186088
        Reduce input records=18678
        Reduce output records=849
        Spilled Records=37356
        Shuffled Maps =110
        Failed Shuffles=0
        Merged Map outputs=110
```

Both results, PerTaskTally (left) and SiCombiner (right), while having slightly different aggregation strategies, result in similar number of input records in the reducer phase. Both approaches also maximize the use of parallel computing by either having an in-mapper aggregation using a HashMap or using a custom combiner that aggregating the output records of the mappers records locally. Thus, I would consider them as equally good.

- **NEW: Comparing the results for Configurations 1 and 2, do you believe this MapReduce program scales well to larger clusters? Briefly justify your answer**
  When looking as the total Step runtime from EC2, we can see that the steps that ran with 11 instances tend to be around 20 seconds faster compared to those that ran with 6 instances. This does show that there is a benefit to running more cluster. However, when looking at other factors (CPU time spent, total time spent by all mappers) there is actually less difference between the two configuration. Which means that the program does not benefit from having more instances, thus does not scale particularly well with a larger cluster.