CS 6240: Assignment 2

Xiao Wang

GitHub: https://github.ccs.neu.edu/xiaowang/CS6240-MapReduce

1. Map-Reduce Algorithms

No Combiner:

```
map(station, temp):
    emit(station, temp)
reduce(station, [temp]):
    sumTemp = sum([temp])
    emit(station, sumTemp / len([temp]))
```

Combiner:

```
map(station, temp):
    emit(station, temp)
combiner(station, [temp]):
    sumTemp = sum([temp])
    emit(station, sumTemp)
reduce(station, [temp]):
    sumTemp = sum([temp])
    emit(station, sumTemp / len([temp]))
```

In-Mapper Combining:

```
map(station, temp):
    hash[station] += temp
cleanup():
    for station in hash:
        emit(station, hash[station])
reduce(station, [temp]):
    sumTemp = sum([temp])
    emit(station, sumTemp / len([temp]))
```

TimeSeries:

```
map(station, year, temp):
    emit((station, year), (year, temp))
keyComparator((station, year)):
    // sorts in increasing order of station first
    // if stationID is same, sorts in increasing order of year
keyHashCode((station, year)):
    // make sure same station go to same reducer
    return station.hashCode()
groupingComparator((station, year)):
    // doesn't consider year
    // each reduce process all (year, temp) for one station
```

```
reduce((station, *), [(year, temp)]):
    forEach in [(year, temp)]:
        sumTemp[year] += temp
        count[year] += temp
    forEach year:
        result += (year, sumTemp[year] / count[year])
    emit(station, result)
```

Each reduce call will process all `[(year, temp)]` for the same station, their order is: first by increasing stationID and second by year.

2. Spark Scala Programs

NoCombiner

```
val weather = sc.textFile("input")
val TMAX = weather.map(text => text.split(","))
                  .filter(entry => entry(2) == "TMAX")
val TMIN = weather.map(text => text.split(","))
                  .filter(entry => entry(2) == "TMIN")

val sumTMAX = TMAX.map(entry => Integer.parseInt(entry(3)))
                  .reduce((sum, t) => sum + t)
val sumTMIN = TMIN.map(entry => Integer.parseInt(entry(3)))
                  .reduce((sum, t) => sum + t)

val numTMAX = TMAX.count
val numTMIN = TMIN.count

println(sumTMAX / numTMAX, sumTMIN / numTMIN)
```

Combiner

```
val weather = sc.textFile("input")
val TMAX = weather.map(text => text.split(","))
                  .filter(entry => entry(2) == "TMAX")
val TMIN = weather.map(text => text.split(","))
                  .filter(entry => entry(2) == "TMIN")

val sumTMAX = TMAX.map(entry => Integer.parseInt(entry(3)))
                  .combineByKey()
                  .reduce((sum, t) => sum + t)
val sumTMIN = TMIN.map(entry => Integer.parseInt(entry(3)))
                  .combineByKey()
                  .reduce((sum, t) => sum + t)

val numTMAX = TMAX.count
val numTMIN = TMIN.count

println(sumTMAX / numTMAX, sumTMIN / numTMIN)
```

Q: Discuss the choice of aggregate function for the first problem (see step 3 above). In particular, which Spark Scala function(s) implement(s) NoCombiner, Combiner, and InMapperComb; and why?
A: The `combineByKey()` serves as the combiner because it combines intermediate results before executing `reduce()`.

3. Performance Comparison

CPU time (ms)	Run1	Run2
NoCombiner	183340	179350
Combiner	170770	169340
InMapperComb	133920	131720
TimeSeries	76090	

Q: Was the Combiner called at all in program Combiner? Was it called more than once per Map task?
A: Yes. From the log file where "`Combine input records=9213198 Combine output records=233643`", we can see the Combiner was called and the number of intermediate results decreased by almost 50 times. From the log file we can't say the Combiner was called more than once per Map task.

Q: Was the local aggregation effective in InMapperComb compared to NoCombiner?
A: Yes. From the performance chart above, we see that the InMapperComb takes 1/3 less time than NoCombiner. And from the log file of InMapperComb "`Map output records=233643`" and NoCombiner "`Map output records=9213198`", we see that InMapperComb reduced huge map output which was the main cause for the performance improvement.