Map-Reduce Implementation:

Twitter Followers Count (Pseudocode):

```
map(String input_key, String input_value):
        // input_key: document name
        // input_value: document contents
        for each row r in input_value:
                follower, user =  r[0], r[1]
                EmitIntermediate(user, 1);

reduce(String intermediate_key, Iterator intermediate_values):
        // intermediate _key: userID
        // intermediate_values: a list of followers count associated with userID
        int followers_count = 0;
        for each val in intermediate_values:
                followers_count += ParseInt(val);
        Emit(AsString(followers_count));
```

Solution Discussion:

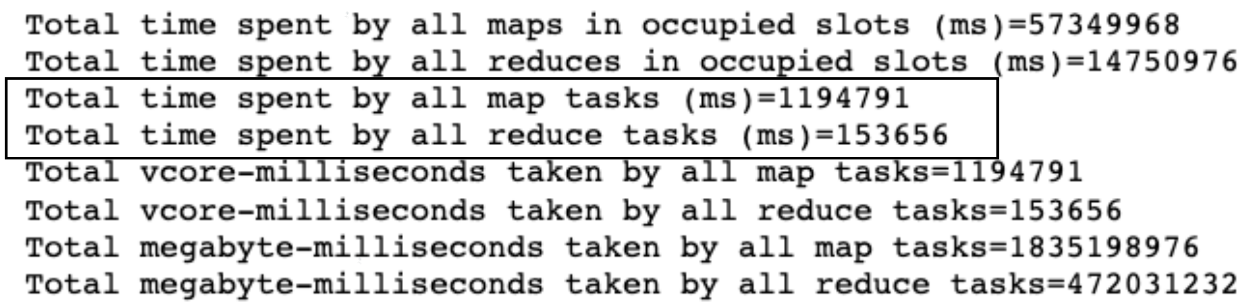
The purpose of the program is to count the number of followers for each user on Twitter. The MapReduce library splits the input file(s) into *M* pieces, and starts many copies of the program on a cluster of machines managed by the master node.

A worker who is assigned a map task reads the contents of the corresponding input split. It parses key/value pairs out of the input data and passes each pair to the user-defined map function. The map function parses each line to extract the document contents (i.e. follower, user) and for each user it emits an integer one, i.e. a key/value pair <user, 1>. Next, the intermediate key/value pairs produced by the map function are combined together and buffered in memory.

When a reduce worker is notified by the master about the location of the buffered pairs, it uses remote procedure calls to read the data from the local disks of the map workers. The reduce worker iterates over the intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values as an Iterable type to the reduce function. The reduce function then computes the total count of followers for each user by iterating over all the intermediate values and calculating their respective sums.

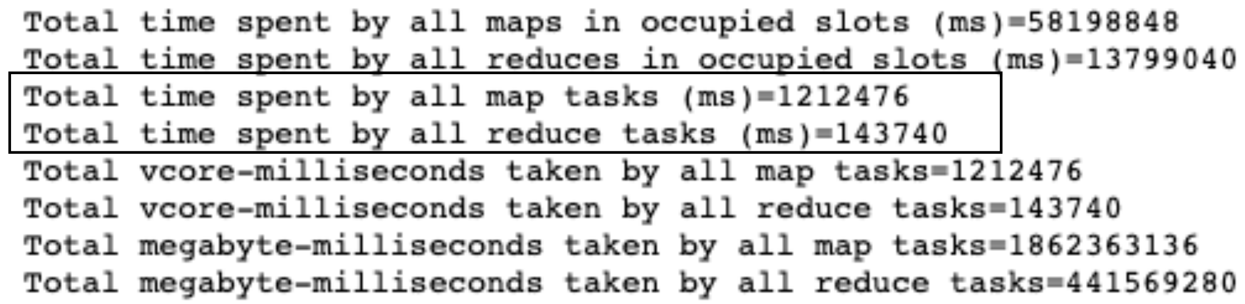
Running Time Measurements

First run: *Time taken by 20 mappers and 10 reducers (1 master, 5 workers)*



```
Total time spent by all maps in occupied slots (ms)=57349968
Total time spent by all reduces in occupied slots (ms)=14750976
Total time spent by all map tasks (ms)=1194791
Total time spent by all reduce tasks (ms)=153656
Total vcore-milliseconds taken by all map tasks=1194791
Total vcore-milliseconds taken by all reduce tasks=153656
Total megabyte-milliseconds taken by all map tasks=1835198976
Total megabyte-milliseconds taken by all reduce tasks=472031232
```

Second run: *Time taken by 20 mappers and 10 reducers (1 master, 5 workers)*

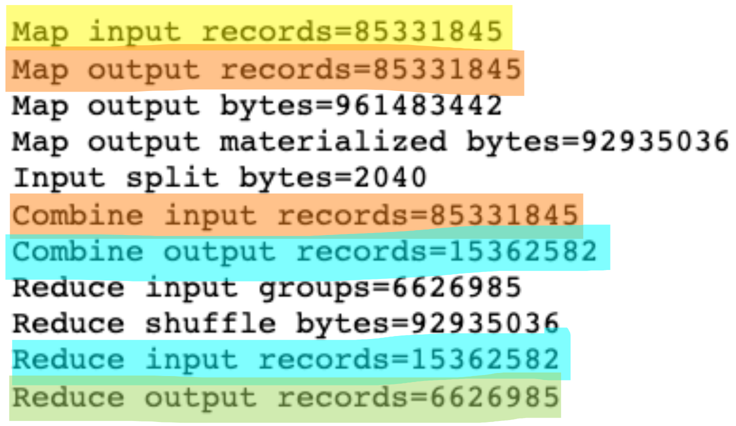


```
Total time spent by all maps in occupied slots (ms)=58198848
Total time spent by all reduces in occupied slots (ms)=13799040
Total time spent by all map tasks (ms)=1212476
Total time spent by all reduce tasks (ms)=143740
Total vcore-milliseconds taken by all map tasks=1212476
Total vcore-milliseconds taken by all reduce tasks=143740
Total megabyte-milliseconds taken by all map tasks=1862363136
Total megabyte-milliseconds taken by all reduce tasks=441569280
```

Data transferred to Mappers , from Mappers to Combiners, from Combiners to Reducers. The output of the Mapper becomes the input to the Combiner, and the output of the Combiner is the input of the Reducer.

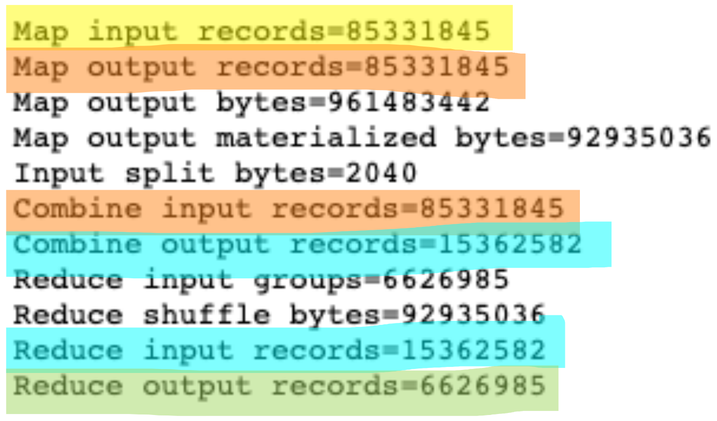
Log for Data Transfer

First run:



```
Map input records=85331845
Map output records=85331845
Map output bytes=961483442
Map output materialized bytes=92935036
Input split bytes=2040
Combine input records=85331845
Combine output records=15362582
Reduce input groups=6626985
Reduce shuffle bytes=92935036
Reduce input records=15362582
Reduce output records=6626985
```

Second run:



```
Map input records=85331845
Map output records=85331845
Map output bytes=961483442
Map output materialized bytes=92935036
Input split bytes=2040
Combine input records=85331845
Combine output records=15362582
Reduce input groups=6626985
Reduce shuffle bytes=92935036
Reduce input records=15362582
Reduce output records=6626985
```

MapReduce Speedup Analysis

The MapReduce program was configured to run with 1 master and 5 workers on AWS. The master and the MapReduce library part of the user program are inherently sequential. The MapReduce library executes all of the work for a MapReduce operation on the local machine and the master picks idle workers and assigns each one a map task or a reduce task sequentially. Nevertheless, MapReduce can drastically speed up big data tasks by breaking down large datasets and processing them in parallel, the FollowersCount program does exactly that by launching 20 mapper tasks in parallel followed by 10 reducer tasks in parallel. Therefore, the MapReduce program is expected to have a good speedup.