CISC5950 Project_1 Report

Jingyan Xu, Haoran Xue, Sheng Yun

April 2, 2023

Introduction

MapReduce is a programming model based on the idea of dividing a large computational problem into smaller sub-problems that can be processed separately in parallel. The framework consists of two main components: a map function and a reduce function. The input data is first split into a set of key-value pairs in the mapper, then passes these pairs to the reducer phase, aggregating them to produce a set of output values. In some complicated cases, it also allows running multiple rounds of MapReduce to get the desired results. It allows multiple machines to handle the tasks in a cluster, making it possible to analyze large datasets efficiently. In this project, we aimed to solve two data analysis problems using Hadoop MapReduce-based program. For the first part, we concentrated on the New York City (NYC) Parking Violations data to identify the most common time and location tickets issued, the most usual years and types of cars being ticketed, and the color of cars that get parking tickets the most frequently. The second part involved analyzing the NBA shots taken during the 2014-2015 seasons. We aimed to determine each player's "most unwanted defender" based on the fear score. Additionally, we aimed to classify each player's records into four comfortable zones and then considered which zone was the best for James Harden, Chris Paul, Stephen Curry, and LeBron James with the hit rate. To do this, we developed our own Python implementation of the MapReduce framework to analyze the data. Our implementation followed one or multiple rounds of the MapReduce approach.

Dataset Description

NYC Parking Violation Data

This data set must place at dir: "/proj1/q1", and the file name of this csv file must be "data.csv"!

The dataset on NYC Parking Violations is collected by the NYC Department of Finance every year. It is hosted and publicly accessible on the platform known as NYC Open Data, which attempts to give open access to a variety of datasets and data resources published and managed by New York City government organizations. The NYC Parking Violation dataset contains the information on each ticket violation record in New York City. The dataset contains a total of 11535314 rows representing each ticket and 43 columns with more detailed information, such as the date, location the violation occurred, and much more, including the information of vehicles, such as the type and the color of vehicles.

NBA Shot Logs Data

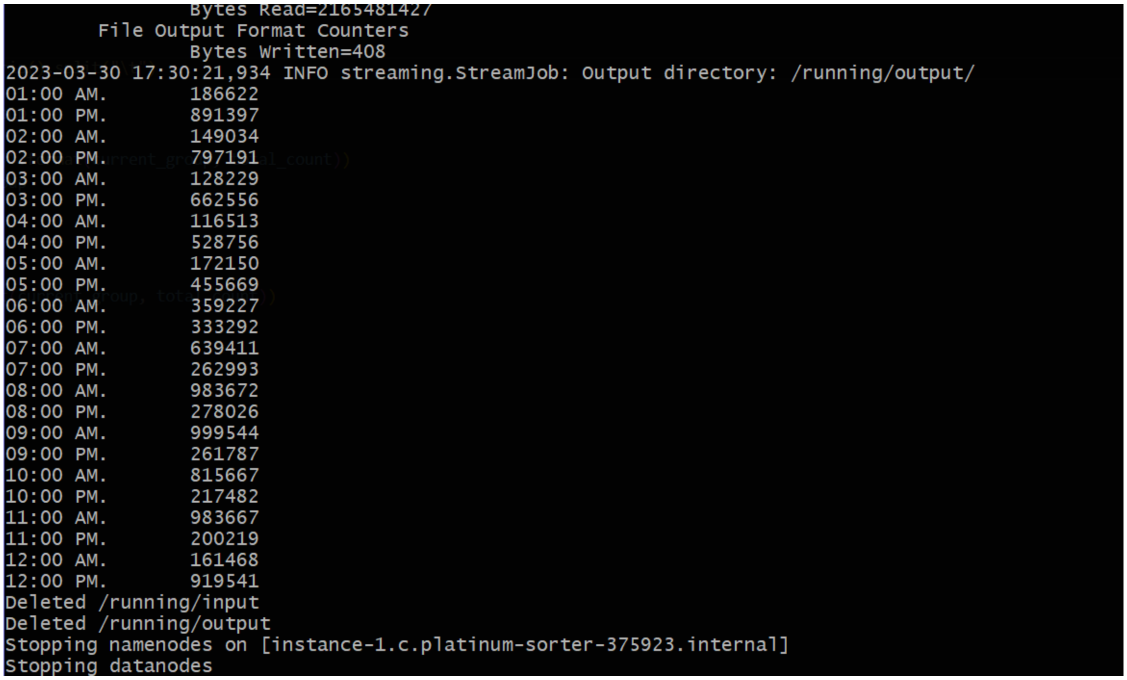
This data set must place at dir: "/proj1/q2", and the file name of this csv file must be "shot_logs.csv"!

The dataset on NBA Shot Logs records the shots taken during the 2014-2015 season, which provides a rich source of data on the shot-taking behaviour of NBA players. The dataset includes 128069 rows that represent each shot attempts taken by each player throughout the season, and 21 columns show information on who took the shots, who was the nearest defender, time on the shot clock, and much more.

First Part of the Project

In this part, we were expected to set up the Hadoop Mapreduce-based framework along with the scheduler for resource management to analyze the violation tickets situation in New York City. We used the NYC Parking Violations 2023 dataset. We analyzed the ticket violations situation during the period of 6/10/2022 to 2/23/2023. Specifically, we were going to answer the following four questions:

- When are tickets most likely to be issued?
- What are the most common years and types of cars to be ticketed?
- Where are tickets most commonly issued?
- Which color of the vehicle is most likely to get a ticket?



```
                Bytes Read=2165481427
        File Output Format Counters
                Bytes Written=408
2023-03-30 17:30:21,934 INFO streaming.StreamJob: Output directory: /running/output/
01:00 AM.       186622
01:00 PM.       891397
02:00 AM.       149034
02:00 PM.       797191
03:00 AM.       128229
03:00 PM.       662556
04:00 AM.       116513
04:00 PM.       528756
05:00 AM.       172150
05:00 PM.       455669
06:00 AM.       359227
06:00 PM.       333292
07:00 AM.       639411
07:00 PM.       262993
08:00 AM.       983672
08:00 PM.       278026
09:00 AM.       999544
09:00 PM.       261787
10:00 AM.       815667
10:00 PM.       217482
11:00 AM.       983667
11:00 PM.       200219
12:00 AM.       161468
12:00 PM.       919541
Deleted /running/input
Deleted /running/output
Stopping namenodes on [instance-1.c.platinum-sorter-375923.internal]
Stopping datanodes
```

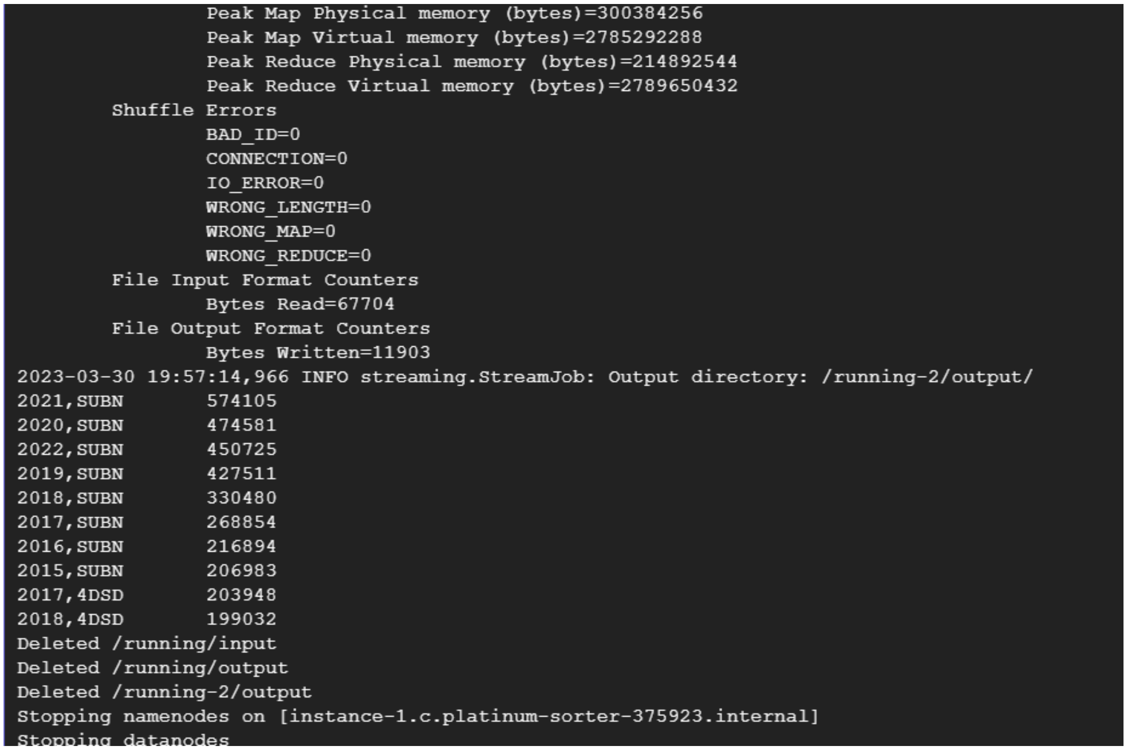
Figure 1: Q1-question1

When are tickets most likely to be issued?

We developed a single round of MapReduce to answer this question. The NYC Parking Violation dataset contains both violation issued date and time. The data showed which day the violation was issued and the time was accurate time of the violation occurred. To answer this question more convincingly we decided to use the accurate time to define when are tickets most likely to be issued. The purpose of our MapReduce script was to extract the time values from the input dataset, round them down to the nearest hour, and output the resulting hours along with a count of 1 for each hour. This output could then be used as input to a MapReduce job to produce a count of events per hour.

We first processed the value at the mapper phase. To start with, we set up a loop that read the input dataset line by line, and then we extracted the time value which was stored in the 20th field in the dataset. Then we filtered out any invalid values by checking if the time variable is not empty, if the first character of time is not equal to 'V', and if the length of time is exactly 5, to ensure the time variable was valid and useful. Then we rounded the time value down to the nearest hour by taking the first two characters of time and appending '00' in the middle, and then followed by the last character of time, which represented AM or PM. Then we printed out the produced key-value pair of hour,1 and passed them to the reducer phase. In the reducer phase, we first split the pairs of values into two parts, the first field is assumed to be the hour value, and the second field is assumed to be the count of events for that hour. The count value has been converted from a string to an integer. Then we checked if the hour value have changed since the last line of input. If it had, the program printed the hour value and the current count of violations. The violations_count' variable is then reset to 0, and the current_time is updated to the new hour value. Otherwise, if the hour value had not changed, the program simply added the count value to the violation_count. Finally, our program printed the last hour value and the final counts of violation.

Based on our development, the results showed that at regular work time, from 8:00 AM to 5:00 PM, the tickets being issued were much more than the time out of this range. It was reasonable because, during this period, there were more people out there, the street was usually crowded and much more vehicles on the street, which increased the violation cases. Additionally, the most often time tickets get issued was 9:00 AM with 999544 counts.



```
                Peak Map Physical memory (bytes)=300384256
                Peak Map Virtual memory (bytes)=2785292288
                Peak Reduce Physical memory (bytes)=214892544
                Peak Reduce Virtual memory (bytes)=2789650432
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=67704
        File Output Format Counters
                Bytes Written=11903
2023-03-30 19:57:14,966 INFO streaming.StreamJob: Output directory: /running-2/output/
2021,SUBN       574105
2020,SUBN       474581
2022,SUBN       450725
2019,SUBN       427511
2018,SUBN       330480
2017,SUBN       268854
2016,SUBN       216894
2015,SUBN       206983
2017,4DSD       203948
2018,4DSD       199032
Deleted /running/input
Deleted /running/output
Deleted /running-2/output
Stopping namenodes on [instance-1.c.platinum-sorter-375923.internal]
Stopping datanodes
```

Figure 2: Q1-question2

What are the most common years and types of cars to be ticketed?

To answer this question, we designed multiple rounds of the MapReduce program. We selected the features of Vehicle Year and Vehicle Make from the NYC Parking Violation dataset. In the first round, we extracted the year and type of each car and checked if the year value was valid and fell within the range of 1500 to 2024. If the year value was valid, we produced a year and made, along with 1 a key-value pair. The pairs would be passed to the first reducer phase, where to get our first-round outputs. The reducer split the pairs into two parts, the first field was a combination of year and makes values which should be the key, and the second field was the count of cars for the combination of the year and make. Then the reducer did similar things as the previous question, checking if the pair of years and make value has changed. If it had changed, it printed the year and made the value and the current count of violations. The program would add the count value to the violations_count variable if it did not change. Then we aggregated the counts of each car year-type pair as our first round of MapReduce outputs. The second round of MapReduce consisted of another mapper and reducer. The mapper2 would read the output from the first reducer to swap the keys and values to enable sorting by count and produce a new key-value pair. The output was then passed to the reducer. The reducer grouped the key-value pairs by counting from the output of the mapper as input. It chose the year and type of cars with the highest frequency and emitted it as a key-value pair along with the count for each group of key-value pairs with the exact count. The output is arranged in descending order by count. The output would give us the most common year and type of cars to be ticketed.

Our output was shown in Figure 2, and we could see that the most common years and types of cars to be ticketed was 2021, SUBN, with counts 574105.

Where are tickets most commonly issued?

We designed two rounds of MapReduce to solve this question. The NYC Parking Violation dataset includes various information related to location, such as the Street Name, Street Code, and the Violation Location. To answer this question more compelling, we decided to use the features of Street Name and Violation Location together to show the place of tickets most likely to be issued. In the first round, we extracted the information of the street name and the violation locations from the dataset,

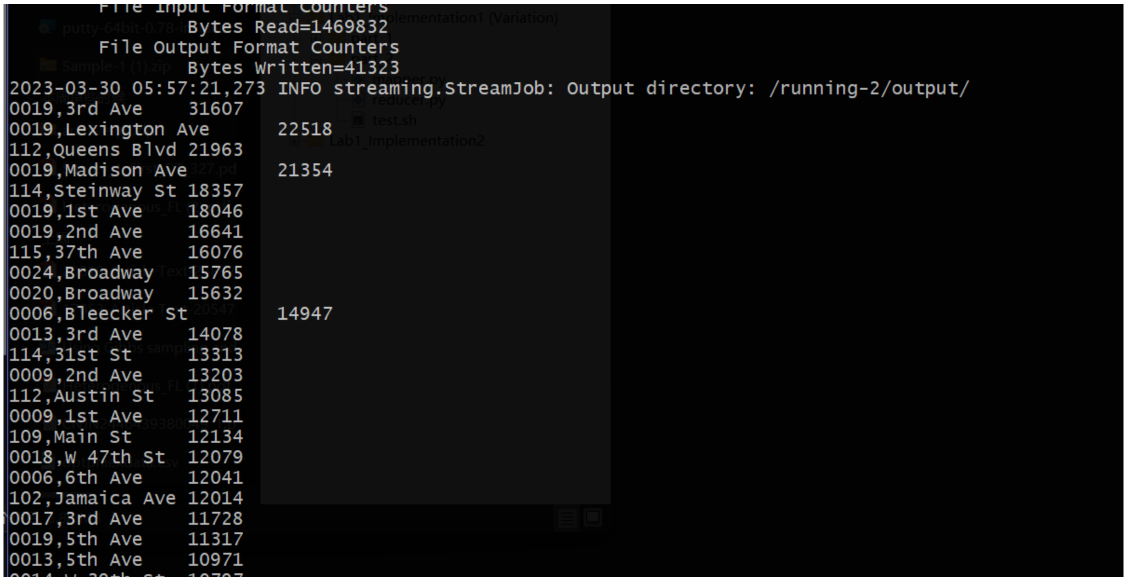


Figure 3: Q1-question3

and then we generated the key-value pair of a combination of street name and location, along with 1 if the location variable is valid to use. For these key-value pairs generated from the mapper phase, the key should be the combination of street name and violation location, while the value should be counted 1. After generating this, we passed the pairs to the reducer phase and got the aggregated counts for each street and location for this station. The second MapReduce program involved another round of mapper and reducer. In the mapper phase of the second round of MapReduce, we reproduced the pairs by swapping the keys and values and then generating new key-value pairs of 1, the combination of street names and locations. Then we aggregated the counts and printed the final counts for each location in the final reducer phase. The whole progress was developed as similar to the question2 because we got the same type of key-value pairs. Therefore, we could handle the values in this way to solve this question.

Our output showed in Figure 3, where the 3rd Ave, street code 0019, was the ticket most commonly issued place, with counts 31607.

Which color of the vehicle is most likely to get a ticket?

For this question, we developed three rounds of MapReduce to achieve the goal. We mainly focused on the feature of Vehicle Color in the NYC Parking Violation dataset to find out the color of the most often ticketed vehicles. Here we met a struggling challenge in that the color of vehicles listed in the dataset were duplicated, which meant that several color names referred to the same color exactly. Therefore we tried to find a way to group the same color to get the total counts of these colors. We used the first round of MapReduce that generated the key-value pairs of color and counts, which were extracted from the dataset at mapper and then got the aggregated counts for each color at the reducer phase, and followed by the second round of MapReduce to make the counts could be sorted by producing new key-value pairs. At this stage, we got the output of all colors with the counts. The most complicated progress was that there were multiple ways to represent the same color, such as 'WH', 'WHI' and 'WHITE' all represented white color, 'BK', 'BLK', and 'BLACK' all represented black color, 'GRY', 'GREY' and 'GY' all represented for grey color, and much more. This problem would cause inaccurate results of counting. For example, if we did not group the color, we got the color of most likely getting tickets cars was grey, and the following top color was white. Therefore, to fix and solve this problem, we added another round of MapReduce to group the same color meaning together, however, we only considered the top 20 often get ticketed colors due to a large number of records. Because these colors occurred much more times than others even though we have not group them. We created a new dictionary to collect the same color as the only one. After this, we got seven colors, BLACK, BLUE, BROWN, GRAY, GREEN, RED, and WHITE, and we noticed that WHITE

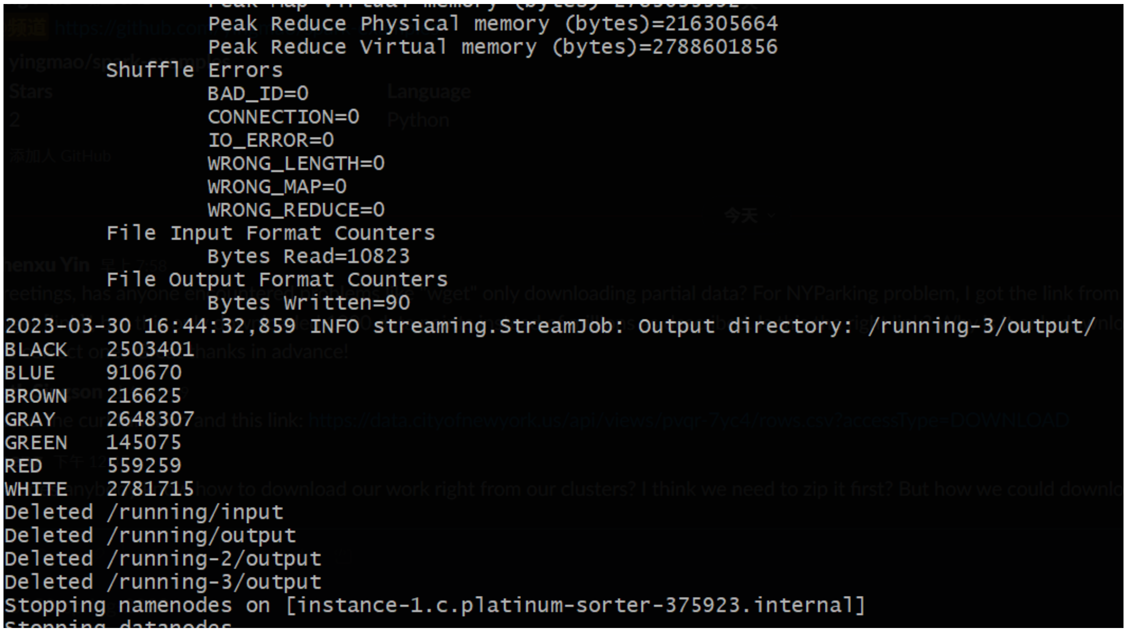


Figure 4: Q1-question4

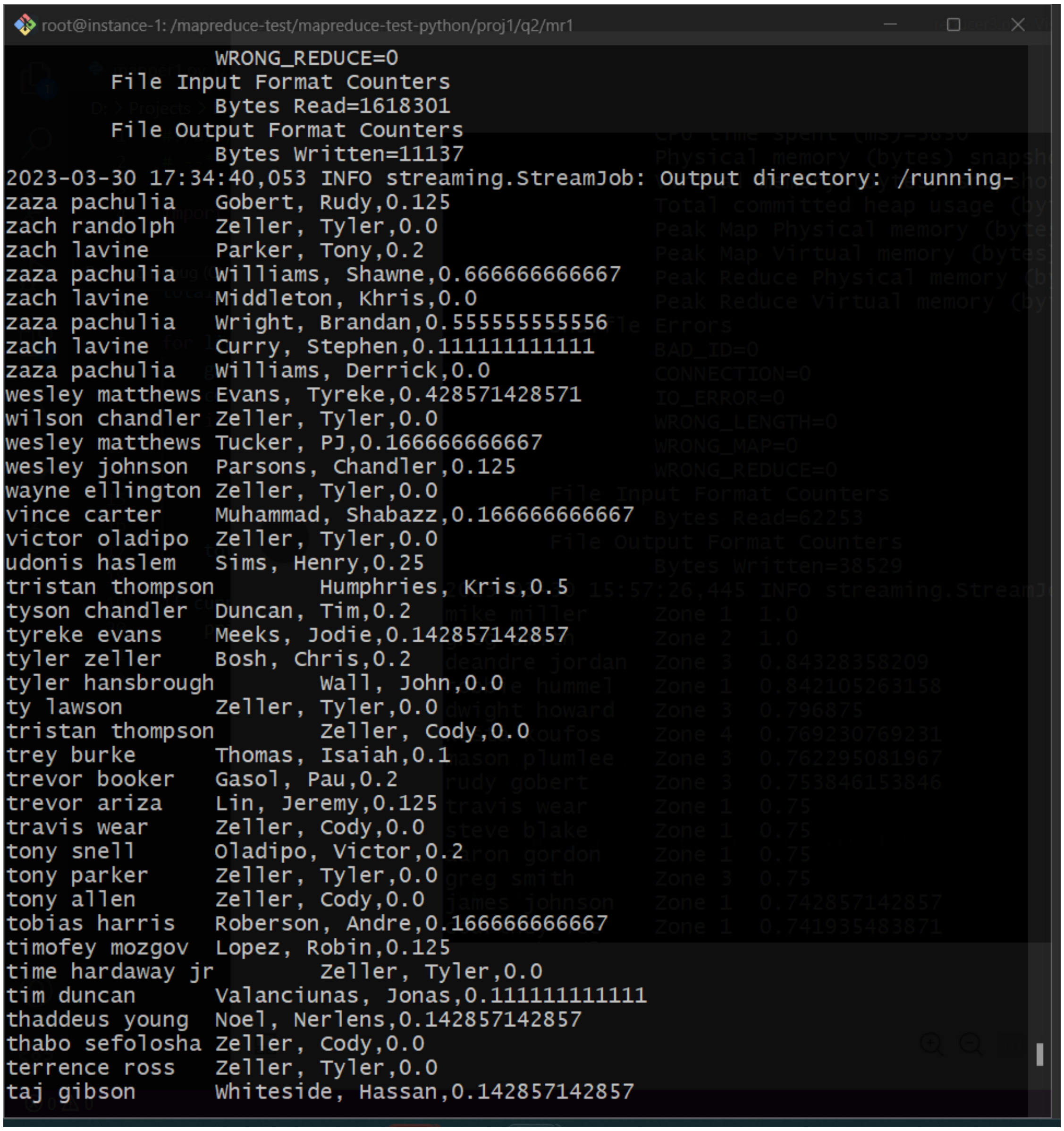
vehicles, with 2781715 times, were the most likely to be ticketed. (The output is shown in Figure 4.)

Second Part of project

The purpose of this part question is to analyze the NBA Shot Logs dataset during the 2014-2015 season. We designed our own MapReduced-based program to determine who a player's "most despised defender" for each player is. We also developed a MapRedcued-based algorithm to classify each player's records into four comfortable zones, and specifically, we aimed to find out which zone is optimal for James Harden, Chris Paul, Stephen Curry, and Lebron James in terms of hit rate.

Players' most unwanted defender

In this question, we were asked to define the player's "most unwanted defender" for each player. We designed a two-round of Mapreduce-based program to solve this. Each round of MapReduce plays a different role in solving this question. We used the first MapReduce to determine the fear score for each shooter/defender pair, which was the shooting result when the shooter faced the defender. The second MapReduce we designed to find out the player's most "unwanted defender", for each player based on the fear score we calculated manually from the first phase. We created our first round of MapReduce by extracting the values of 'PLAYER_NAME', 'CLOSEST_DEFENDER', and 'SHOT_RESULT' from the NBA Shot Logs dataset, as shooter, defender, and results were three central values we needed to determine the fear score for each shooter/defender pair. Then the reducer took in the output from a mapper, and we got the output key-value pairs of the player and his defender, with the hit rate computed as the ratio of shots made to total shots taken by the shooter against the defender. The program calculated and printed the shooting ratio for the last shooter/defender pair when all data were processed. In the second round of MapReduce, the mapper produced new key-value pairs of shooter/defender combination and the fear score. Then we moved to the reducer phase. We first initialized some values by setting current_shooter to 0 so that we could track when we processed a new shooter. We defined the value of max_fear_score, which was set to 2, that higher than the maximum possible shooting percentage of 1.0, so we could update it with the first valid shooting percentage we encountered. The value of max_defender was set to None initially, as we have not encountered any shooting percentages yet. Then we split the shooter, defender and fear_scores from pairs, and we did the normal reducer process, checking if we were still processing the same shooter. If we were, we then checked if the current defender had a lower shooting percentage than the previous "most unwanted



```
root@instance-1: /mapreduce-test/mapreduce-test-python/proj1/q2/mr1
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=1618301
        File Output Format Counters
                Bytes Written=11137
2023-03-30 17:34:40,053 INFO streaming.StreamJob: Output directory: /running-
zaza pachulia   Gobert, Rudy,0.125
zach randolph   Zeller, Tyler,0.0
zach lavine     Parker, Tony,0.2
zaza pachulia   Williams, Shawne,0.666666666667
zach lavine     Middleton, Khris,0.0
zaza pachulia   Wright, Brandan,0.555555555556
zach lavine     Curry, Stephen,0.111111111111
zaza pachulia   Williams, Derrick,0.0
wesley matthews Evans, Tyreke,0.428571428571
wilson chandler Zeller, Tyler,0.0
wesley matthews Tucker, PJ,0.166666666667
wesley johnson  Parsons, Chandler,0.125
wayne ellington Zeller, Tyler,0.0
vince carter    Muhammad, Shabazz,0.166666666667
victor oladipo  Zeller, Tyler,0.0
udonis haslem   Sims, Henry,0.25
tristan thompson        Humphries, Kris,0.5
tyson chandler  Duncan, Tim,0.2
tyreke evans    Meeks, Jodie,0.142857142857
tyler zeller    Bosh, Chris,0.2
tyler hansbrough        Wall, John,0.0
ty lawson       Zeller, Tyler,0.0
tristan thompson        Zeller, Cody,0.0
trey burke      Thomas, Isaiah,0.1
trevor booker   Gasol, Pau,0.2
trevor ariza    Lin, Jeremy,0.125
travis wear     Zeller, Cody,0.0
tony snell      Oladipo, Victor,0.2
tony parker     Zeller, Tyler,0.0
tony allen      Zeller, Cody,0.0
tobias harris   Roberson, Andre,0.166666666667
timofey mozgov  Lopez, Robin,0.125
time hardaway jr        Zeller, Tyler,0.0
tim duncan      Valanciunas, Jonas,0.111111111111
thaddeus young  Noel, Nerlens,0.142857142857
thabo sefolosha Zeller, Cody,0.0
terrence ross   Zeller, Tyler,0.0
taj gibson      Whiteside, Hassan,0.142857142857
```

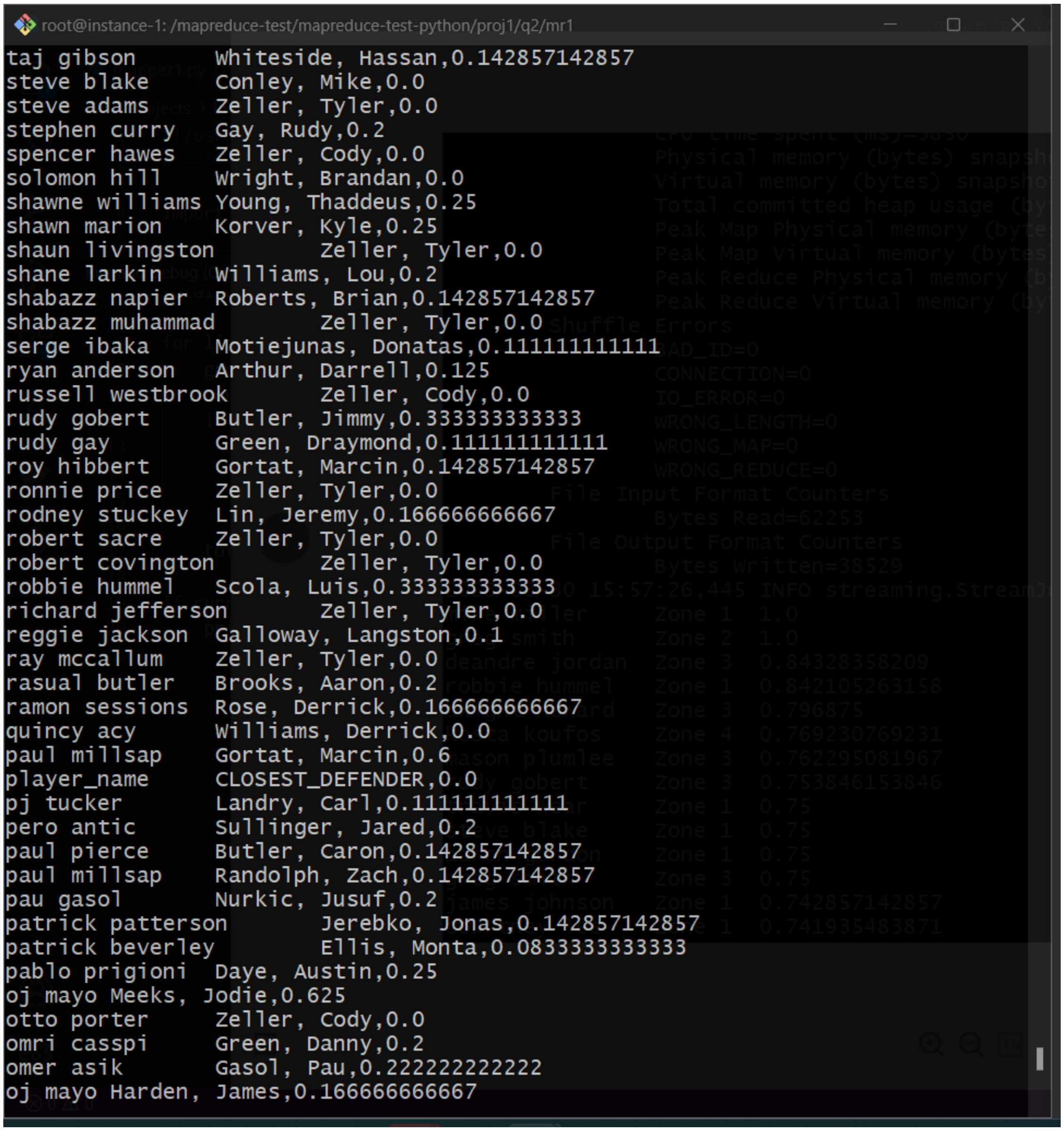
Figure 5: Q2-question1-1

defender" (we do not want it to be 0 since we noticed that there is a condition that the defender only defended the shooter once and the shooter missed the shot, and this cannot conclude that the shooter feared that defender. We simply eliminate this possibility by not allowing the fear score to be 0 (if the percentage is low and not equal to zero, this means the hit rate is really low, and the shooter attempted a lot). However, if we could do it better, we could set up a criterion of the least certain amount of shot attempts for the shooters), and update the max_fear_score and max_defender variables to reflect this. Otherwise, if we have processed a new shooter, we printed out the result for the previous shooter and updated the current_shooter, max_fear_score, and max_defender variables to start processing the new shooter.

When all data were processed, we put the results on of all pairs of players, from Figure 5 to Figure 12.

Classify four comfortable zone

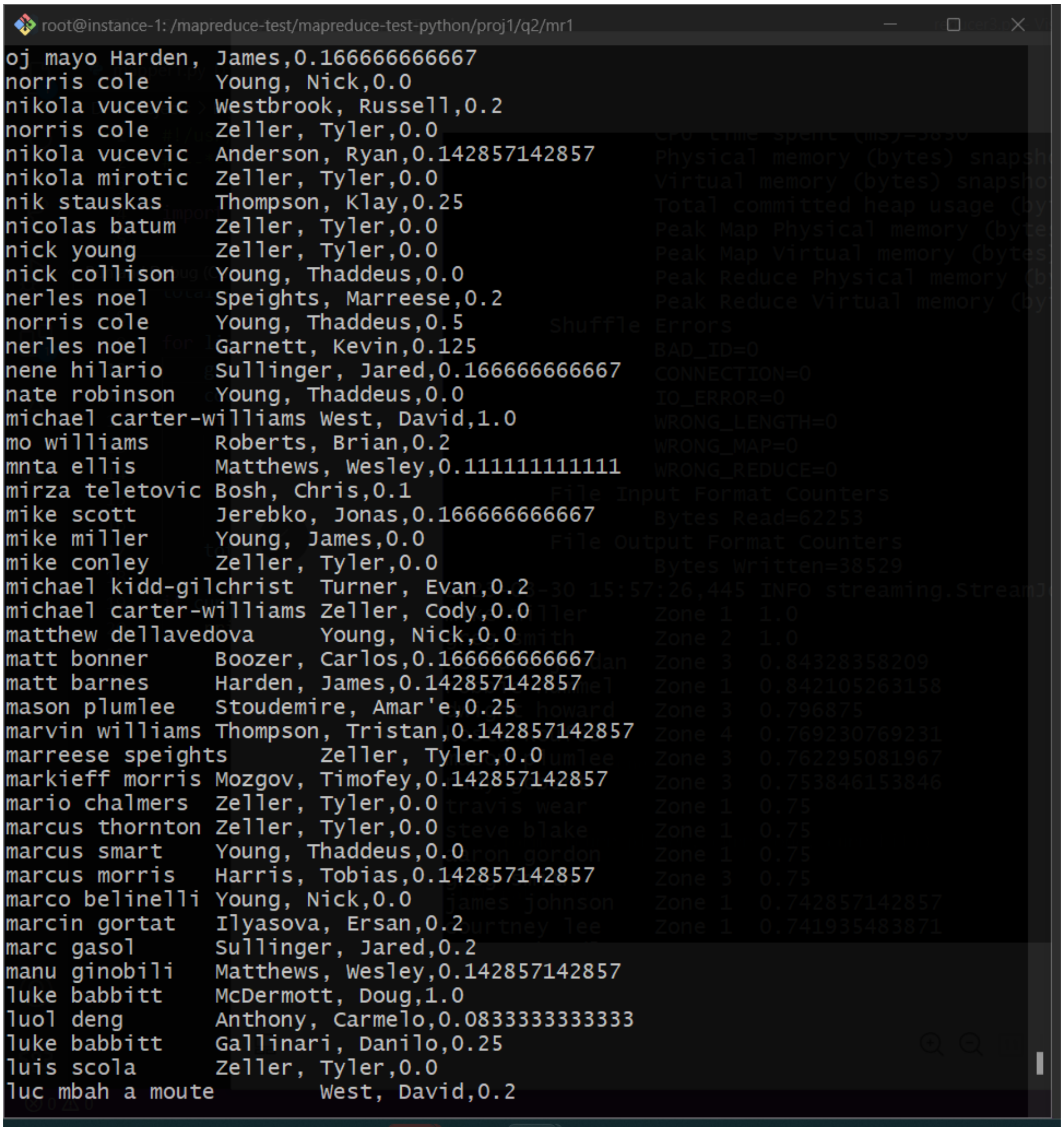
For this question, we need to classify the shooting player's comfortable zones with the criteria: {SHOT DIST, CLOSE DEF DIST, SHOT CLOCK}. Since we were huge NBA fans and had a good under-



root@instance-1: /mapreduce-test/mapreduce-test-python/proj1/q2/mr1

```
taj gibson      Whiteside, Hassan,0.142857142857
steve blake     Conley, Mike,0.0
steve adams     Zeller, Tyler,0.0
stephen curry   Gay, Rudy,0.2
spencer hawes   Zeller, Cody,0.0
solomon hill    Wright, Brandan,0.0
shawne williams Young, Thaddeus,0.25
shawn marion    Korver, Kyle,0.25
shaun livingston        Zeller, Tyler,0.0
shane larkin    Williams, Lou,0.2
shabazz napier  Roberts, Brian,0.142857142857
shabazz muhammad        Zeller, Tyler,0.0
serge ibaka     Motiejunas, Donatas,0.111111111111
ryan anderson   Arthur, Darrell,0.125
russell westbrook       Zeller, Cody,0.0
rudy gobert     Butler, Jimmy,0.333333333333
rudy gay        Green, Draymond,0.111111111111
roy hibbert     Gortat, Marcin,0.142857142857
ronnie price    Zeller, Tyler,0.0
rodney stuckey  Lin, Jeremy,0.166666666667
robert sacre    Zeller, Tyler,0.0
robert covington        Zeller, Tyler,0.0
robbie hummel   Scola, Luis,0.333333333333
richard jefferson       Zeller, Tyler,0.0
reggie jackson  Galloway, Langston,0.1
ray mccallum    Zeller, Tyler,0.0
rasual butler   Brooks, Aaron,0.2
ramon sessions  Rose, Derrick,0.166666666667
quincy acy      Williams, Derrick,0.0
paul millsap    Gortat, Marcin,0.6
player_name     CLOSEST_DEFENDER,0.0
pj tucker       Landry, Carl,0.111111111111
pero antic      Sullinger, Jared,0.2
paul pierce     Butler, Caron,0.142857142857
paul millsap    Randolph, Zach,0.142857142857
pau gasol       Nurkic, Jusuf,0.2
patrick patterson       Jerebko, Jonas,0.142857142857
patrick beverley        Ellis, Monta,0.0833333333333
pablo prigioni  Daye, Austin,0.25
oj mayo Meeks, Jodie,0.625
otto porter     Zeller, Cody,0.0
omri casspi     Green, Danny,0.2
omer asik       Gasol, Pau,0.222222222222
oj mayo Harden, James,0.166666666667
```

Figure 6: Q2-question1-2



```
root@instance-1: /mapreduce-test/mapreduce-test-python/proj1/q2/mr1
oj mayo Harden, James,0.166666666667
norris cole     Young, Nick,0.0
nikola vucevic  Westbrook, Russell,0.2
norris cole     Zeller, Tyler,0.0
nikola vucevic  Anderson, Ryan,0.142857142857
nikola mirotic  Zeller, Tyler,0.0
nik stauskas    Thompson, Klay,0.25
nicolas batum   Zeller, Tyler,0.0
nick young      Zeller, Tyler,0.0
nick collison   Young, Thaddeus,0.0
nerles noel     Speights, Marreese,0.2
norris cole     Young, Thaddeus,0.5
nerles noel     Garnett, Kevin,0.125
nene hilario    Sullinger, Jared,0.166666666667
nate robinson   Young, Thaddeus,0.0
michael carter-williams West, David,1.0
mo williams     Roberts, Brian,0.2
mnta ellis      Matthews, Wesley,0.111111111111
mirza teletovic Bosh, Chris,0.1
mike scott      Jerebko, Jonas,0.166666666667
mike miller     Young, James,0.0
mike conley     Zeller, Tyler,0.0
michael kidd-gilchrist  Turner, Evan,0.2
michael carter-williams Zeller, Cody,0.0
matthew dellavedova     Young, Nick,0.0
matt bonner     Boozer, Carlos,0.166666666667
matt barnes     Harden, James,0.142857142857
mason plumlee   Stoudemire, Amar'e,0.25
marvin williams Thompson, Tristan,0.142857142857
marreese speights       Zeller, Tyler,0.0
markieff morris Mozgov, Timofey,0.142857142857
mario chalmers Zeller, Tyler,0.0
marcus thornton Zeller, Tyler,0.0
marcus smart    Young, Thaddeus,0.0
marcus morris   Harris, Tobias,0.142857142857
marco belinelli Young, Nick,0.0
marcin gortat   Ilyasova, Ersan,0.2
marc gasol      Sullinger, Jared,0.2
manu ginobili   Matthews, Wesley,0.142857142857
luke babbitt    McDermott, Doug,1.0
luol deng       Anthony, Carmelo,0.0833333333333
luke babbitt    Gallinari, Danilo,0.25
luis scola      Zeller, Tyler,0.0
luc mbah a moute        West, David,0.2
```

Figure 7: Q2-question1-3

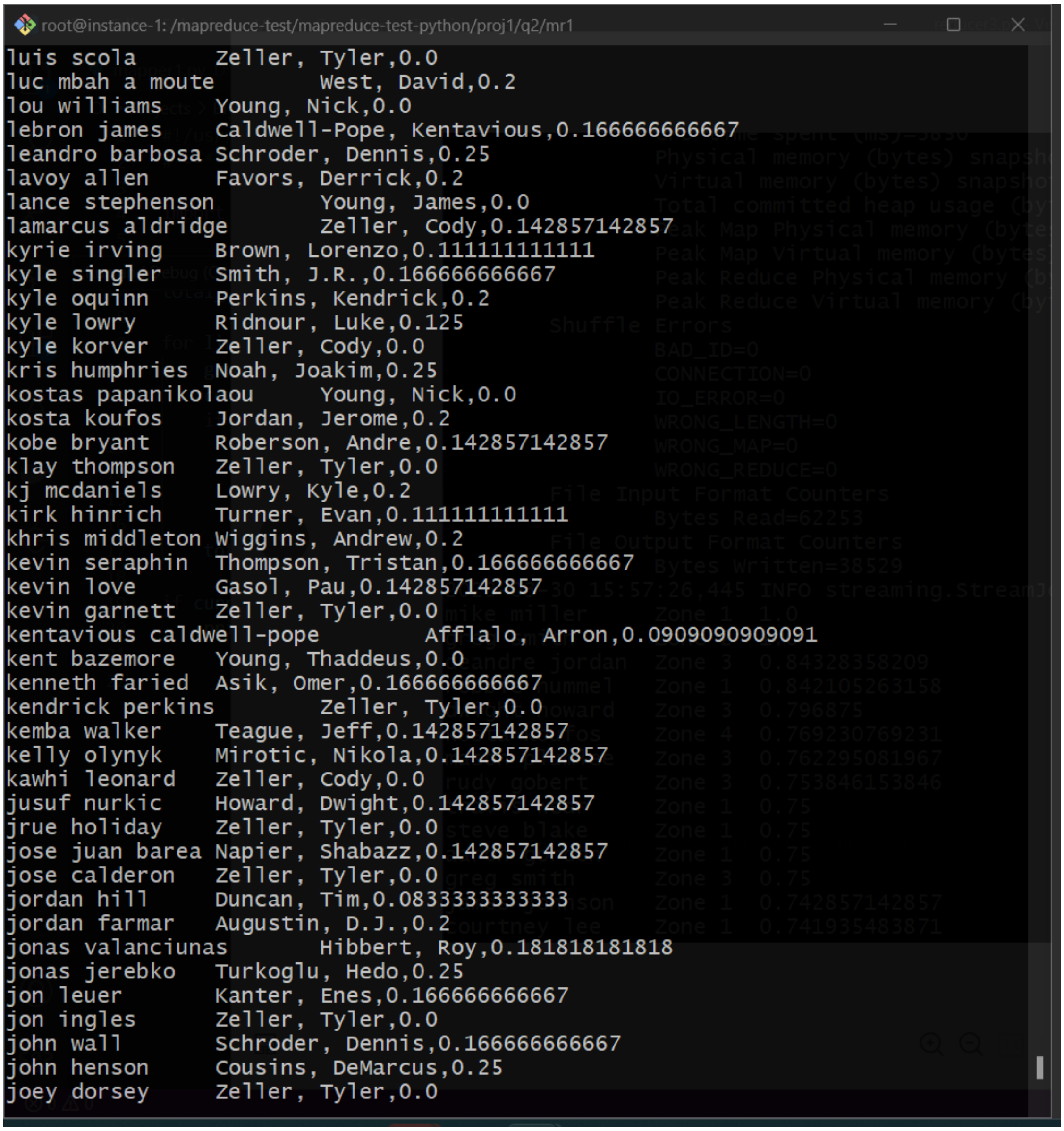
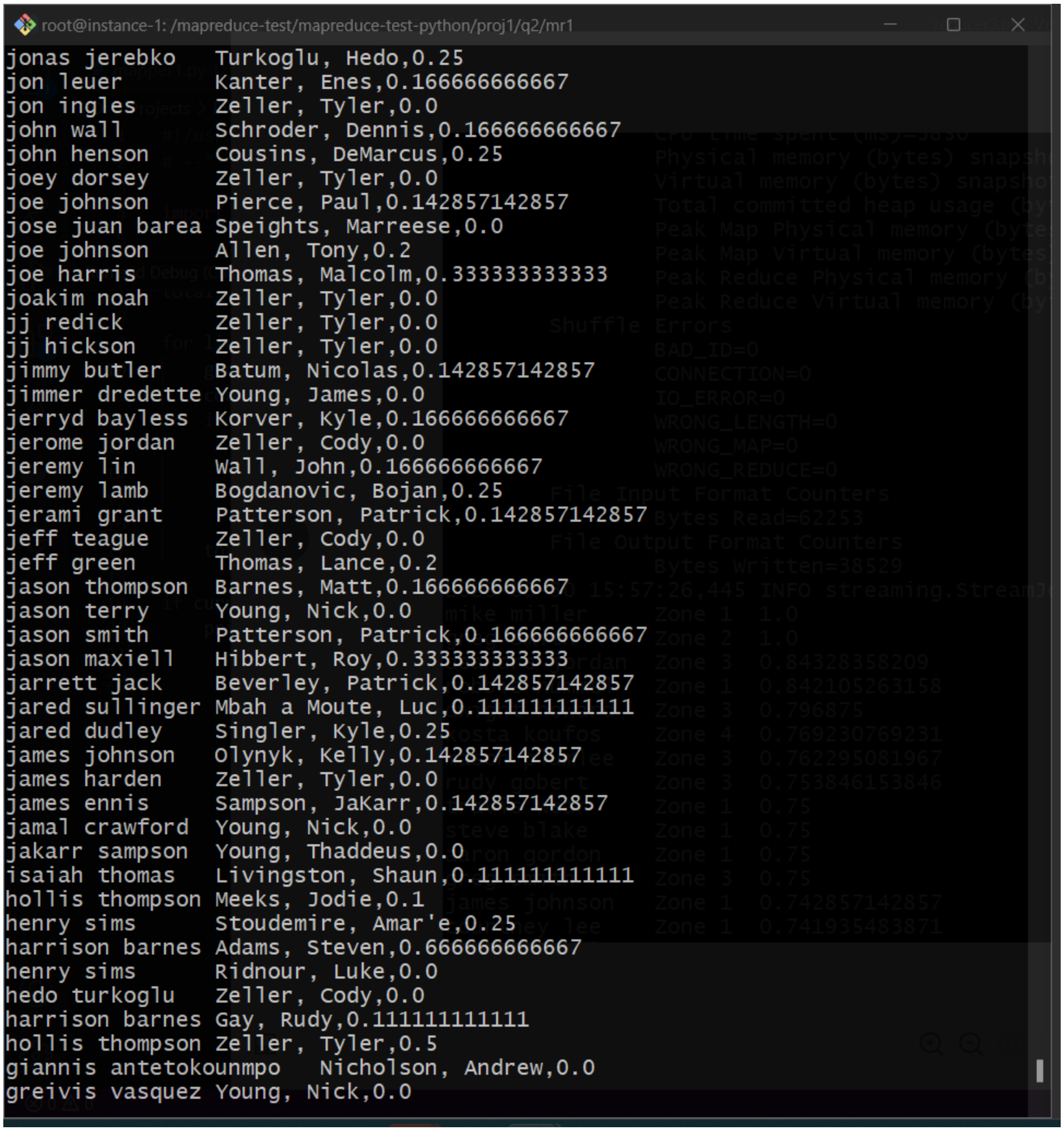


Figure 8: Q2-question1-4



root@instance-1: /mapreduce-test/mapreduce-test-python/proj1/q2/mr1

```
jonas jerebko   Turkoglu, Hedo,0.25
jon leuer       Kanter, Enes,0.166666666667
jon ingles      Zeller, Tyler,0.0
john wall       Schroder, Dennis,0.166666666667
john henson     Cousins, DeMarcus,0.25
joey dorsey     Zeller, Tyler,0.0
joe johnson     Pierce, Paul,0.142857142857
jose juan barea Speights, Marreese,0.0
joe johnson     Allen, Tony,0.2
joe harris      Thomas, Malcolm,0.333333333333
joakim noah     Zeller, Tyler,0.0
jj redick       Zeller, Tyler,0.0
jj hickson      Zeller, Tyler,0.0
jimmy butler    Batum, Nicolas,0.142857142857
jimmer dredette Young, James,0.0
jerryd bayless  Korver, Kyle,0.166666666667
jerome jordan   Zeller, Cody,0.0
jeremy lin      Wall, John,0.166666666667
jeremy lamb     Bogdanovic, Bojan,0.25
jerami grant    Patterson, Patrick,0.142857142857
jeff teague     Zeller, Cody,0.0
jeff green      Thomas, Lance,0.2
jason thompson  Barnes, Matt,0.166666666667
jason terry     Young, Nick,0.0
jason smith     Patterson, Patrick,0.166666666667
jason maxiell   Hibbert, Roy,0.333333333333
jarrett jack    Beverley, Patrick,0.142857142857
jared sullinger Mbah a Moute, Luc,0.111111111111
jared dudley    Singler, Kyle,0.25
james johnson   Olynyk, Kelly,0.142857142857
james harden    Zeller, Tyler,0.0
james ennis     Sampson, JaKarr,0.142857142857
jamal crawford  Young, Nick,0.0
jakarr sampson  Young, Thaddeus,0.0
isaiah thomas   Livingston, Shaun,0.111111111111
hollis thompson Meeks, Jodie,0.1
henry sims      Stoudemire, Amar'e,0.25
harrison barnes Adams, Steven,0.666666666667
henry sims      Ridnour, Luke,0.0
hedo turkoglu   Zeller, Cody,0.0
harrison barnes Gay, Rudy,0.111111111111
hollis thompson Zeller, Tyler,0.5
giannis antetokounmpo   Nicholson, Andrew,0.0
greivis vasquez Young, Nick,0.0
```

Figure 9: Q2-question1-5

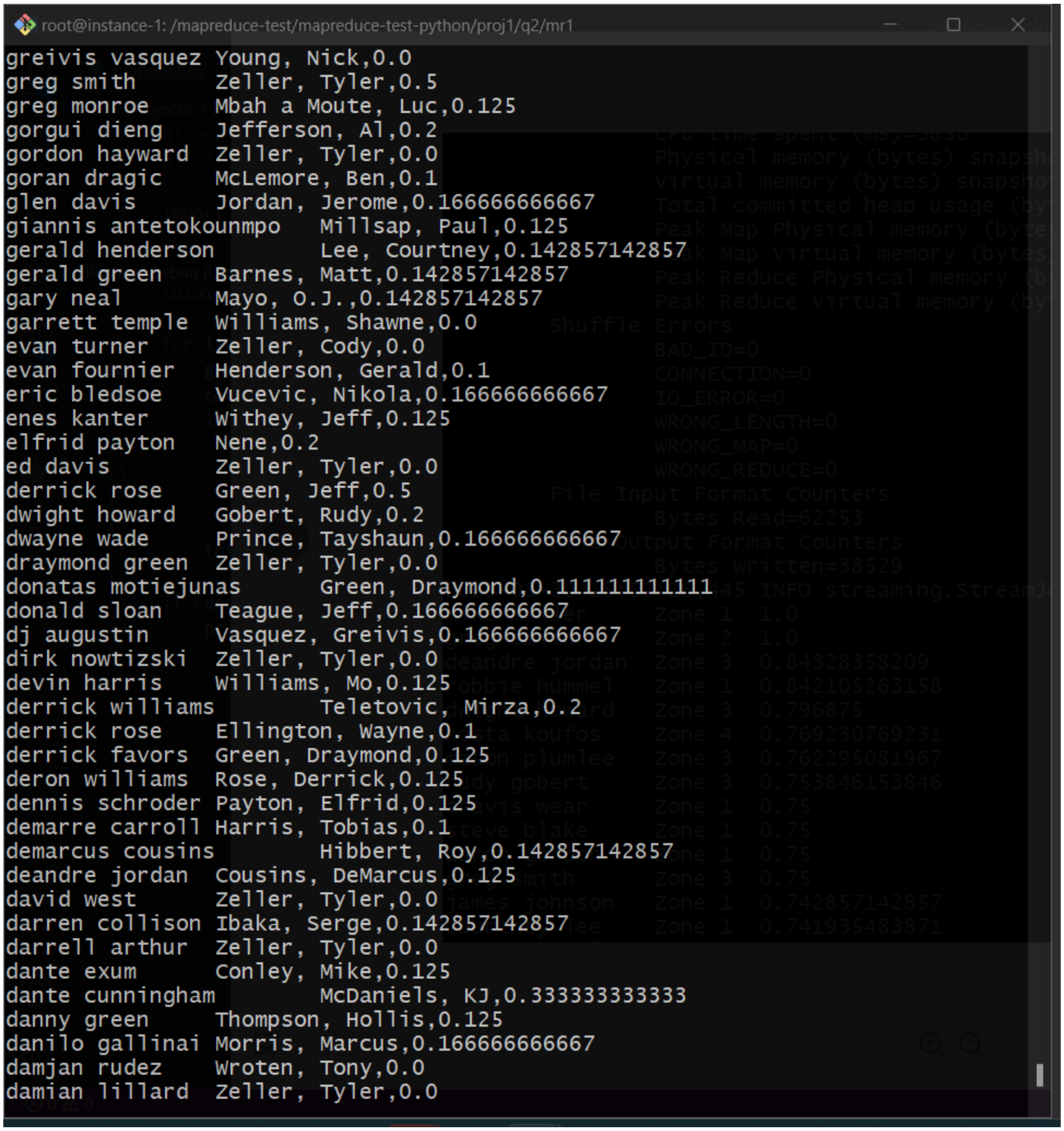
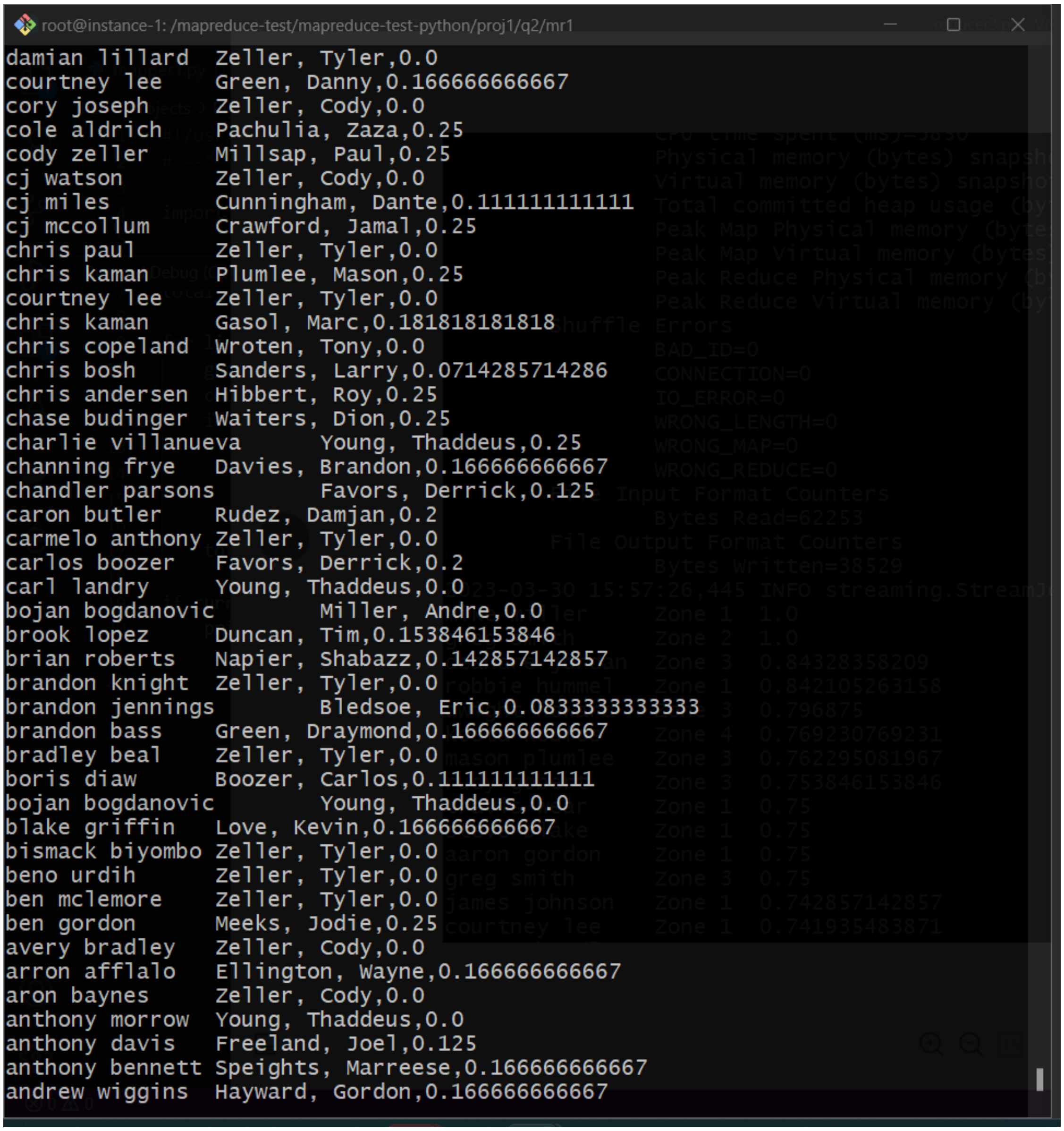
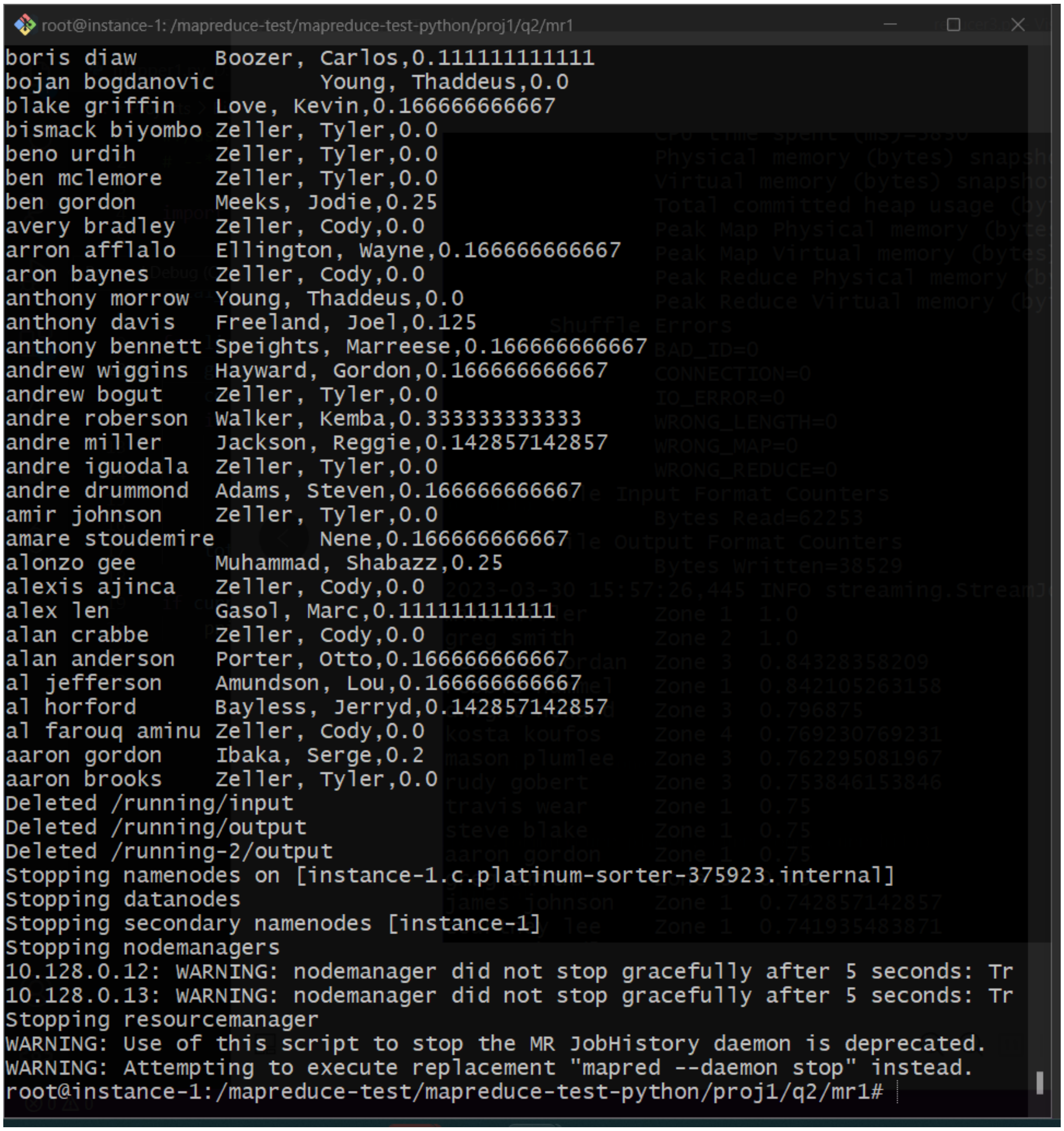


Figure 10: Q2-question1-6



```
root@instance-1: /mapreduce-test/mapreduce-test-python/proj1/q2/mr1
damian lillard  Zeller, Tyler,0.0
courtney lee    Green, Danny,0.166666666667
cory joseph     Zeller, Cody,0.0
cole aldrich    Pachulia, Zaza,0.25
cody zeller     Millsap, Paul,0.25
cj watson       Zeller, Cody,0.0
cj miles        Cunningham, Dante,0.111111111111
cj mccollum     Crawford, Jamal,0.25
chris paul      Zeller, Tyler,0.0
chris kaman     Plumlee, Mason,0.25
courtney lee    Zeller, Tyler,0.0
chris kaman     Gasol, Marc,0.181818181818
chris copeland  Wroten, Tony,0.0
chris bosh      Sanders, Larry,0.0714285714286
chris andersen  Hibbert, Roy,0.25
chase budinger  Waiters, Dion,0.25
charlie villanueva      Young, Thaddeus,0.25
channing frye   Davies, Brandon,0.166666666667
chandler parsons        Favors, Derrick,0.125
caron butler    Rudez, Damjan,0.2
carmelo anthony Zeller, Tyler,0.0
carlos boozer   Favors, Derrick,0.2
carl landry     Young, Thaddeus,0.0
bojan bogdanovic        Miller, Andre,0.0
brook lopez     Duncan, Tim,0.153846153846
brian roberts   Napier, Shabazz,0.142857142857
brandon knight  Zeller, Tyler,0.0
brandon jennings        Bledsoe, Eric,0.0833333333333
brandon bass    Green, Draymond,0.166666666667
bradley beal    Zeller, Tyler,0.0
boris diaw      Boozer, Carlos,0.111111111111
bojan bogdanovic        Young, Thaddeus,0.0
blake griffin   Love, Kevin,0.166666666667
bismack biyombo Zeller, Tyler,0.0
beno urdih      Zeller, Tyler,0.0
ben mclemore    Zeller, Tyler,0.0
ben gordon      Meeks, Jodie,0.25
avery bradley   Zeller, Cody,0.0
arron afflalo   Ellington, Wayne,0.166666666667
aron baynes     Zeller, Cody,0.0
anthony morrow  Young, Thaddeus,0.0
anthony davis   Freeland, Joel,0.125
anthony bennett Speights, Marreese,0.166666666667
andrew wiggins  Hayward, Gordon,0.166666666667
```

Figure 11: Q2-question1-7



```
root@instance-1: /mapreduce-test/mapreduce-test-python/proj1/q2/mr1
boris diaw      Boozer, Carlos,0.111111111111
bojan bogdanovic        Young, Thaddeus,0.0
blake griffin   Love, Kevin,0.166666666667
bismack biyombo Zeller, Tyler,0.0
beno urdih      Zeller, Tyler,0.0
ben mclemore    Zeller, Tyler,0.0
ben gordon      Meeks, Jodie,0.25
avery bradley   Zeller, Cody,0.0
arron afflalo   Ellington, Wayne,0.166666666667
aron baynes     Zeller, Cody,0.0
anthony morrow  Young, Thaddeus,0.0
anthony davis   Freeland, Joel,0.125
anthony bennett Speights, Marreese,0.166666666667
andrew wiggins  Hayward, Gordon,0.166666666667
andrew bogut    Zeller, Tyler,0.0
andre roberson  Walker, Kemba,0.333333333333
andre miller    Jackson, Reggie,0.142857142857
andre iguodala  Zeller, Tyler,0.0
andre drummond  Adams, Steven,0.166666666667
amir johnson    Zeller, Tyler,0.0
amare stoudemire        Nene,0.166666666667
alonzo gee      Muhammad, Shabazz,0.25
alexis ajinca   Zeller, Cody,0.0
alex len        Gasol, Marc,0.111111111111
alan crabbe     Zeller, Cody,0.0
alan anderson   Porter, Otto,0.166666666667
al jefferson    Amundson, Lou,0.166666666667
al horford      Bayless, Jerryd,0.142857142857
al farouq aminu Zeller, Cody,0.0
aaron gordon    Ibaka, Serge,0.2
aaron brooks    Zeller, Tyler,0.0
Deleted /running/input
Deleted /running/output
Deleted /running-2/output
Stopping namenodes on [instance-1.c.platinum-sorter-375923.internal]
Stopping datanodes
Stopping secondary namenodes [instance-1]
Stopping nodemanagers
10.128.0.12: WARNING: nodemanager did not stop gracefully after 5 seconds: Tr
10.128.0.13: WARNING: nodemanager did not stop gracefully after 5 seconds: Tr
Stopping resourcemanager
WARNING: Use of this script to stop the MR JobHistory daemon is deprecated.
WARNING: Attempting to execute replacement "mapred --daemon stop" instead.
root@instance-1:/mapreduce-test/mapreduce-test-python/proj1/q2/mr1#
```

Figure 12: Q2-question1-8

standing of the basketball game that enabled us to have prior knowledge of building the classifier, we omitted the potential process of using a machine learning algorithm to build up a machine learning classifier. We knew that NBA had changed drastically since the Golden State Warriors won their first championship in the 21st century in the 2014-2015 season, which was one of the reasons why we were analyzing this matter. The game of basketball used to believe in the philosophy of ”the closer to the basket, the better”, but since 2015, players and teams have been increasingly attempting to shoot 3's, even contested threes. Thus, we could classify the shot distance into 5 feet and shots from above 5 feet. The shot distance within 5 feet could demonstrate the player's hit rate(close shots, dunks and layups) in close range, and the above would be the shooting hit rate. Then we considered that the defender also played an essential role since it would alter the shooter's decisions and hit rate. We also want to know if someone could comfortably make a difficult shot with the defender in their eyes. We divided the defender's distance data into within three feet and above three feet since we knew that NBA players had an exceptional wingspan that could cover a great amount of contesting area. We thought three feet was a suitable threshold for this classifier. Then we considered that the shot clock was one of the important conditions that impacted the shooter's hit rate since some players would panic in crunch time and missed shots even if they were good shooters in regular time. We divided the shot clock into 5 seconds which meant clutch time and otherwise was the regular time. Also, we saw many times that in clutch time, the stars could take over the game and made shots no matter if anyone was contesting(they called it a better offence and beat the better defence). Thus, we thought we could ignore the factor of the shot distance and the defender's distance when we were in clutch time. Despite these, we knew that Big's would usually avoid attempting shots far away, but it did not mean they would never make one beyond the 5 feet. They may have hit one or two occasions and had a better hit rate since they attempted less and got lucky. However, this would be over our project's objection and get too complicated.

We built the classifier as Zone1: Close shot and got contested in regular time:(shot with 5 feet, the defender is in 3 feet, and the shot clock is in regular time); Zone2: Shots beyond 5 feet and got contested in regular time: (shot over 5 feet, the defender is in 3 feet and the shot clock is in regular time); Zone3: wide-open shots in regular time: (shot from anywhere open, the defender is farther than 3 feet and the shot clock is in regular time); zone4 is otherwise, which means that it is in clutch time:(shoot from anywhere no matter if any defender is contesting). In our first mapper function, we divide the metrics(matrix) into four zones and output each player's shots with a key-value pair with (zone number, made?, 1). Then the stream will push to the reducer1, and the reducer will calculate the hit rate and aggregate each player's every zone along with their hit rates(output:(info, made, attempt, hit rate). In the second mapper, we did not make anything unique, and in reducer2, we found the largest hit rate of their zones for each player. Then we would output the zones with their highest hit rate for each player. This would result in our map reduce process.

From the output, we would get the required four players' comfortable zones, and we also included the screenshot highlighted with their stat from Figure 13 to Figure 17, including the timestamp.

For Stephen Curry: Zone1, 0.624

LeBron James: Zone1, 0.691

Chris Paul: Zone3, 0.523

James Harden: Zone1, 0.533

Bonus Question

K-Means is an unsupervised machine learning algorithm that is commonly used for clustering and segmentation analysis. The algorithm is based on the idea of grouping similar data points into a fixed number of clusters. However, K-Means' biggest challenge is determining the optimal number of classes. This will significantly affect the prediction accuracy of the model. In the Bonus Question, we aimed to develop a MapReduce-based program based on the K-Means algorithm, first to predict the probability of getting a ticket if a black car parking illegally at street codes 34510, 10030, 34050, second to define the parking place that if walking within 0.5 miles to get Lincoln Center. We still worked on the NYC Parking Violations dataset to answer the bonus question.

We first classified the color of parked vehicles into two categories: either black vehicles or not, and if the parking location matched any of the three specified street codes, then produced a key-value pair of color,1 at the mapper phase. The reducer then aggregated the counts for black or not black separately

```
                Peak Map Physical memory (bytes)=301375488
                Peak Map Virtual memory (bytes)=2788589568
                Peak Reduce Physical memory (bytes)=197189632
                Peak Reduce Virtual memory (bytes)=2782437376
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=62253
        File Output Format Counters
                Bytes Written=38556
2023-03-30 19:45:36,664 INFO streaming.StreamJob: Output directory: /running-2/output/
chris andersen  Zone 2  0.0
jerami grant    Zone 4  0.0
kostas papanikolaou     Zone 2  0.0625
darrell arthur  Zone 4  0.1
chris copeland  Zone 4  0.137931034483
jerami grant    Zone 2  0.15
dante exum      Zone 4  0.166666666667
bismack biyombo Zone 2  0.166666666667
udonis haslem   Zone 2  0.166666666667
robbie hummel   Zone 2  0.166666666667
shabazz napier  Zone 1  0.166666666667
zaza pachulia   Zone 4  0.181818181818
al farouq aminu Zone 4  0.181818181818
luc mbah a moute        Zone 4  0.188679245283
trevor ariza    Zone 2  0.191489361702
vince carter    Zone 4  0.193548387097
nate robinson   Zone 4  0.2
ramon sessions  Zone 4  0.2
tony allen      Zone 2  0.206896551724
nik stauskas    Zone 4  0.210526315789
jon leuer       Zone 2  0.214285714286
patrick beverley        Zone 4  0.215384615385
robert sacre    Zone 4  0.21875
omer asik       Zone 2  0.222222222222
pero antic      Zone 2  0.222222222222
luke babbitt    Zone 4  0.222222222222
jason terry     Zone 4  0.222222222222
```

Figure 13:

```
khris middleton Zone 1  0.62
andre miller    Zone 1  0.622641509434
stephen curry   Zone 1  0.623655913978
carlos boozer   Zone 1  0.624
chris andersen  Zone 4  0.625
jerryd bayless  Zone 1  0.625
marvin williams Zone 2  0.625
nikola vucevic  Zone 1  0.627358490566
henry sims      Zone 1  0.627659574468
alex len        Zone 4  0.631578947368
nene hilario    Zone 1  0.6328125
jared dudley    Zone 1  0.633333333333
andre drummond  Zone 3  0.634146341463
bismack biyombo Zone 3  0.636363636364
gerald green    Zone 1  0.638888888889
jonas valanciunas       Zone 1  0.639593908629
matt barnes     Zone 1  0.640625
pj tucker       Zone 1  0.641509433962
derrick williams        Zone 1  0.641509433962
danilo gallinai Zone 1  0.642857142857
ed davis        Zone 1  0.643410852713
tristan thompson        Zone 3  0.643564356436
amir johnson    Zone 3  0.643835616438
lamarcus aldridge       Zone 1  0.644628099174
rudy gobert     Zone 1  0.646153846154
mike scott      Zone 1  0.647058823529
john henson     Zone 3  0.649122807018
goran dragic    Zone 1  0.649717514124
jon ingles      Zone 1  0.65
jerome jordan   Zone 3  0.65
tyler zeller    Zone 3  0.65034965035
steve adams     Zone 3  0.652173913043
aron baynes     Zone 4  0.652173913043
dwayne wade     Zone 1  0.655555555556
kevin seraphin  Zone 1  0.65625
derrick favors  Zone 1  0.657777777778
anthony davis   Zone 1  0.66049382716
marc gasol      Zone 1  0.661538461538
tyson chandler  Zone 4  0.666666666667
luke babbitt    Zone 1  0.666666666667
matt bonner     Zone 1  0.666666666667
dante exum      Zone 1  0.666666666667
jerome jordan   Zone 4  0.666666666667
```

Figure 14: Stephen Curry

```
tyson chandler  Zone 4  0.666666666667
luke babbitt    Zone 1  0.666666666667
matt bonner     Zone 1  0.666666666667
dante exum      Zone 1  0.666666666667
jerome jordan   Zone 4  0.666666666667
alan crabbe     Zone 1  0.666666666667
bismack biyombo Zone 1  0.676923076923
anthony bennett Zone 1  0.68085106383
jon leuer       Zone 1  0.68085106383
jeremy lamb     Zone 1  0.684210526316
deandre jordan  Zone 1  0.687804878049
chris andersen  Zone 3  0.690909090909
lebron james    Zone 1  0.691176470588
kawhi leonard   Zone 1  0.691176470588
anthony morrow  Zone 1  0.692307692308
lavoy allen     Zone 1  0.701754385965
marcin gortat   Zone 1  0.703703703704
darrell arthur  Zone 1  0.714285714286
ed davis        Zone 3  0.71875
tyson chandler  Zone 3  0.731343283582
courtney lee    Zone 1  0.741935483871
james johnson   Zone 1  0.742857142857
greg smith      Zone 3  0.75
aaron gordon    Zone 1  0.75
steve blake     Zone 1  0.75
travis wear     Zone 1  0.75
rudy gobert     Zone 3  0.753846153846
mason plumlee   Zone 3  0.762295081967
kosta koufos    Zone 4  0.769230769231
dwight howard   Zone 3  0.796875
robbie hummel   Zone 1  0.842105263158
deandre jordan  Zone 3  0.84328358209
greg smith      Zone 2  1.0
mike miller     Zone 1  1.0
Deleted /running/input
Deleted /running/output
Deleted /running-2/output
Stopping namenodes on [instance-1.c.platinum-sorter-375923.internal]
Stopping datanodes
Stopping secondary namenodes [instance-1]
Stopping nodemanagers
10.128.0.12: WARNING: nodemanager did not stop gracefully after 5 seconds: Trying to kill with
 kill -9
```

Figure 15: LeBron James

```
omri casspi     Zone 3  0.520408163265
pau gasol       Zone 3  0.521613832853
wesley matthews Zone 1  0.521739130435
marco belinelli Zone 1  0.521739130435
patrick patterson       Zone 1  0.521739130435
mirza teletovic Zone 1  0.521739130435
carl landry     Zone 3  0.522058823529
harrison barnes Zone 3  0.522727272727
oj mayo Zone 1  0.522727272727
chris paul      Zone 3  0.523178807947
jerami grant    Zone 1  0.52380952381
roy hibbert     Zone 1  0.52380952381
michael carter-williams Zone 1  0.52380952381
al jefferson    Zone 4  0.524390243902
boris diaw      Zone 1  0.524390243902
norris cole     Zone 1  0.524590163934
aron baynes     Zone 3  0.525641025641
giannis antetokounmpo   Zone 3  0.526066350711
dante cunningham        Zone 4  0.526315789474
spencer hawes   Zone 1  0.526315789474
lou williams    Zone 1  0.527777777778
quincy acy      Zone 1  0.527777777778
ronnie price    Zone 1  0.529411764706
kosta koufos    Zone 1  0.529411764706
deandre jordan  Zone 4  0.529411764706
jason maxiell   Zone 2  0.529411764706
tony allen      Zone 1  0.530303030303
timofey mozgov  Zone 1  0.530386740331
andre drummond  Zone 1  0.530405405405
nikola mirotic  Zone 1  0.530612244898
demarcus cousins        Zone 1  0.530821917808
chris andersen  Zone 1  0.53125
tony snell      Zone 2  0.53125
kevin garnett   Zone 1  0.53125
jeff green      Zone 1  0.53164556962
donatas motiejunas      Zone 4  0.531914893617
james harden    Zone 1  0.533333333333
shawn marion    Zone 4  0.533333333333
nick collison   Zone 1  0.533333333333
spencer hawes   Zone 4  0.533333333333
kevin seraphin  Zone 3  0.533898305085
michael kidd-gilchrist  Zone 3  0.534246575342
tony allen      Zone 3  0.534246575342
```

Figure 16: Chris Paul

```
jeff green      Zone 1  0.53164556962
donatas motiejunas      Zone 4  0.531914893617
james harden    Zone 1  0.533333333333
shawn marion    Zone 4  0.533333333333
nick collison   Zone 1  0.533333333333
spencer hawes   Zone 4  0.533333333333
kevin seraphin  Zone 3  0.533898305085
michael kidd-gilchrist  Zone 3  0.534246575342
tony allen      Zone 3  0.534246575342
glen davis      Zone 3  0.534246575342
nicolas batum   Zone 1  0.535714285714
ray mccallum    Zone 1  0.535714285714
paul pierce     Zone 1  0.5375
marcus smart    Zone 1  0.538461538462
chris kaman     Zone 3  0.538461538462
bradley beal    Zone 1  0.538461538462
james johnson   Zone 3  0.539130434783
tristan thompson        Zone 1  0.539267015707
amare stoudemire        Zone 4  0.539682539683
ryan anderson   Zone 1  0.540816326531
donatas motiejunas      Zone 2  0.542168674699
zach lavine     Zone 1  0.542857142857
john wall       Zone 1  0.543689320388
reggie jackson  Zone 1  0.544444444444
jonas valanciunas       Zone 3  0.544715447154
amare stoudemire        Zone 3  0.545454545455
damian lillard  Zone 1  0.545454545455
thaddeus young  Zone 1  0.546583850932
danny green     Zone 1  0.547619047619
tim duncan      Zone 3  0.547872340426
nerles noel     Zone 1  0.548387096774
greg smith      Zone 1  0.548387096774
russell westbrook       Zone 1  0.549382716049
paul millsap    Zone 1  0.549504950495
kyle oquinn     Zone 2  0.55
steve adams     Zone 4  0.55
trevor booker   Zone 1  0.550724637681
gorgui dieng    Zone 1  0.551181102362
jared sullinger Zone 1  0.551470588235
jason thompson  Zone 1  0.55223880597
manu ginobili   Zone 1  0.55223880597
chris copeland  Zone 1  0.552631578947
thabo sefolosha Zone 1  0.553571428571
```

Figure 17: James Harden

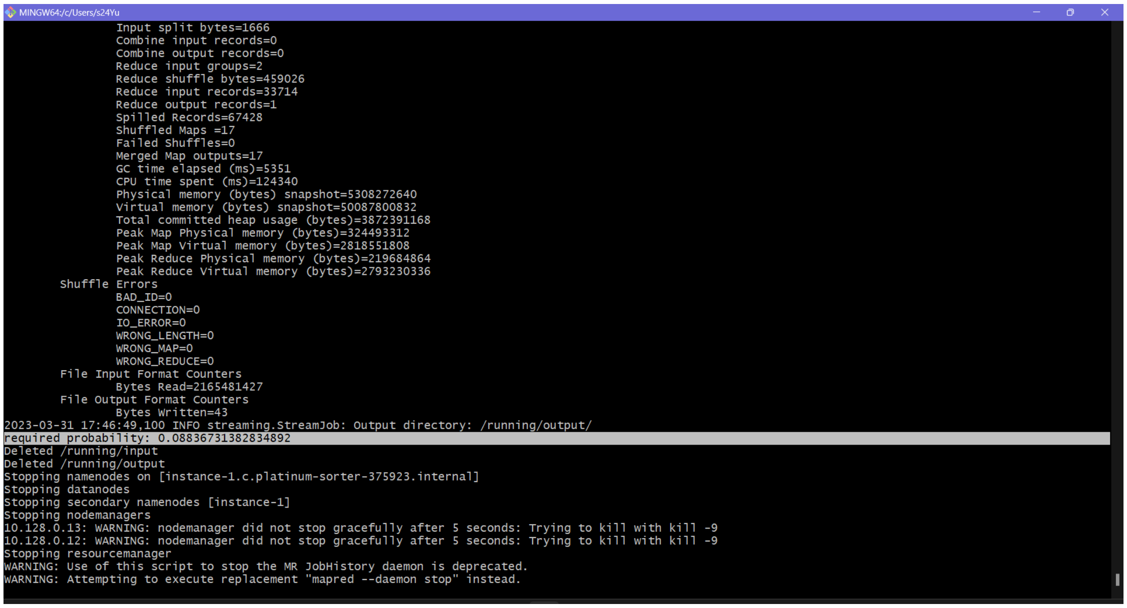


Figure 18: Bonus Question1

and calculated the probability of black vehicles parked at the given location receiving a ticket using the count of black vehicles that received a ticket and the total count of black vehicles parked illegally. The probability was predicted as 0.884 (Figure 18).

For the second question, we developed two rounds of MapReduce to segment the parking areas into different zones based on their proximity to Lincoln Center and recommend the zone with the most parking spots within the 0.5-mile walking distance. From the first mapper, we produced key-value pairs where the key contained two things: the partial sum of all data points and the count of all data points. All this output would be sorted and processed to the reducer. We computed the mean of all data points in the cluster, which was the total sum of all points divided by the total number of points. In the second round of MapReduce, we found the zone with particular datapoints lied so that we knew the zone to which the data point was closest. Then we collected the set of data points that were closest to this zone and whether there was a ticket violation at this point. Now we got the number of parking tickets for each zone from every mapper, and then we printed out the zone with the minimum number of parking tickets as the favourable parking zone, which was zone 2.

The result was shown from Figure 20 to Figure 30

Limitations and Challenges

While MapReduce is well-suited for tasks that can be parallelized, such as data mining and machine learning, there may be better choices for tasks that are efficiently parallelizable. MapReduce usually needs multiple rounds of inputs and outputs to get the desired result. This is because MapReduce is designed to process large datasets in a parallel and distributed manner, which may need multiple iterations of data to produce the results. In these cases, MapReduce may be inefficient and may not provide significant performance gains over traditional sequential programming approaches.

One of the main challenges of MapReduce is the overhead involved in transferring data between nodes in a cluster. Since MapReduce relies on a distributed architecture, data must be transferred between nodes to perform the necessary calculations. This can lead to network bottlenecks and performance issues, particularly when dealing large datasets.

Conclusion

In our project, we developed our MapReduce-based framework and algorithm to analyze NYC Parking Violations in the year 2023 dataset and NBA Shot Logs during the 2014-2015 season dataset separately.

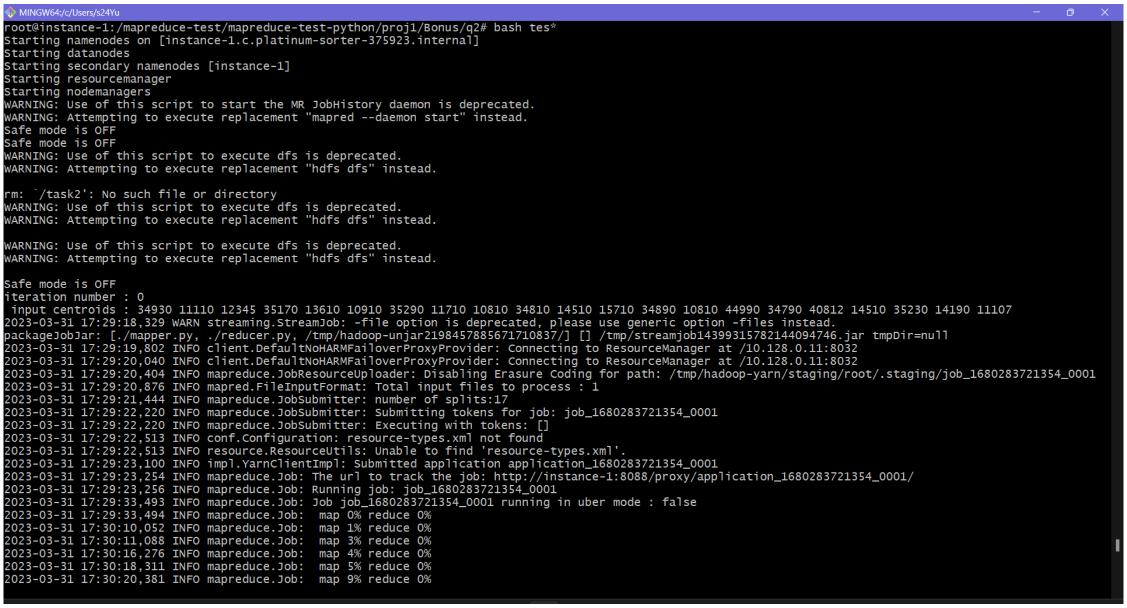


Figure 19:

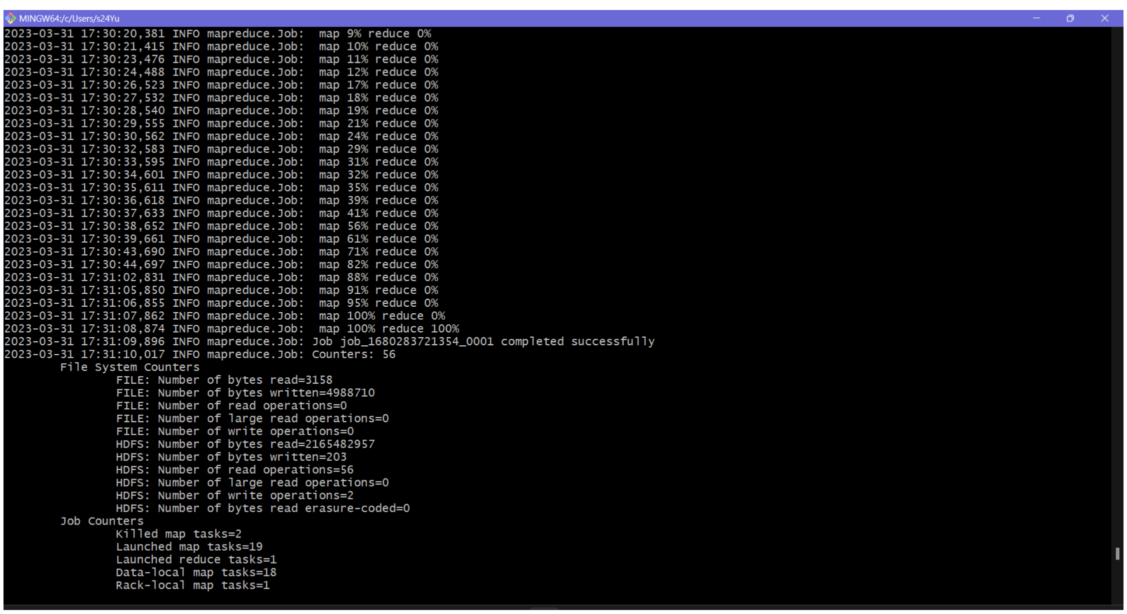


Figure 20:

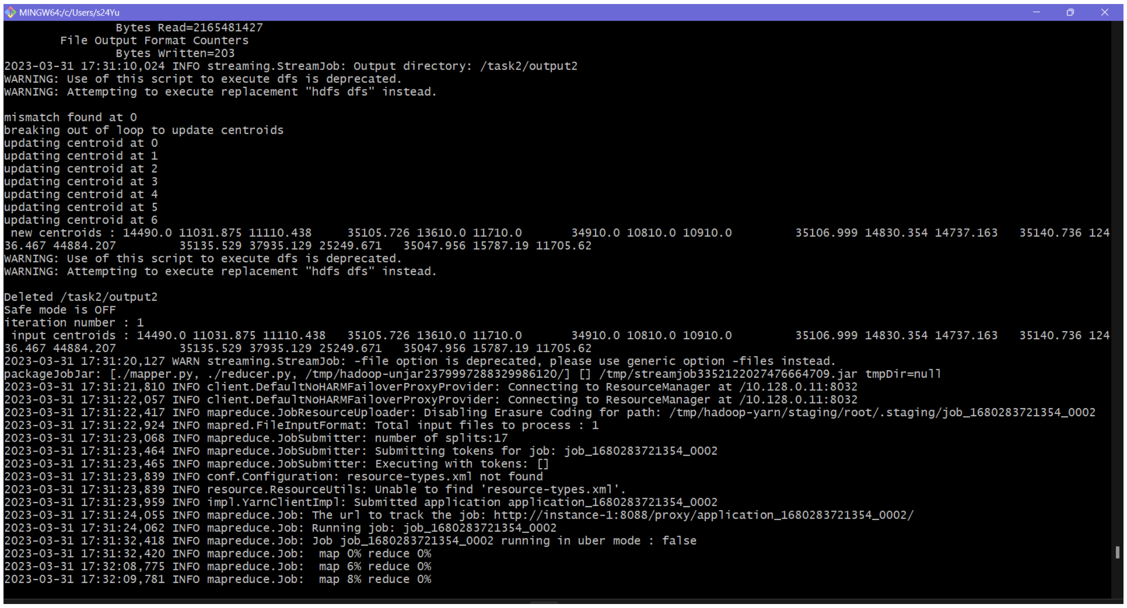


Figure 21:

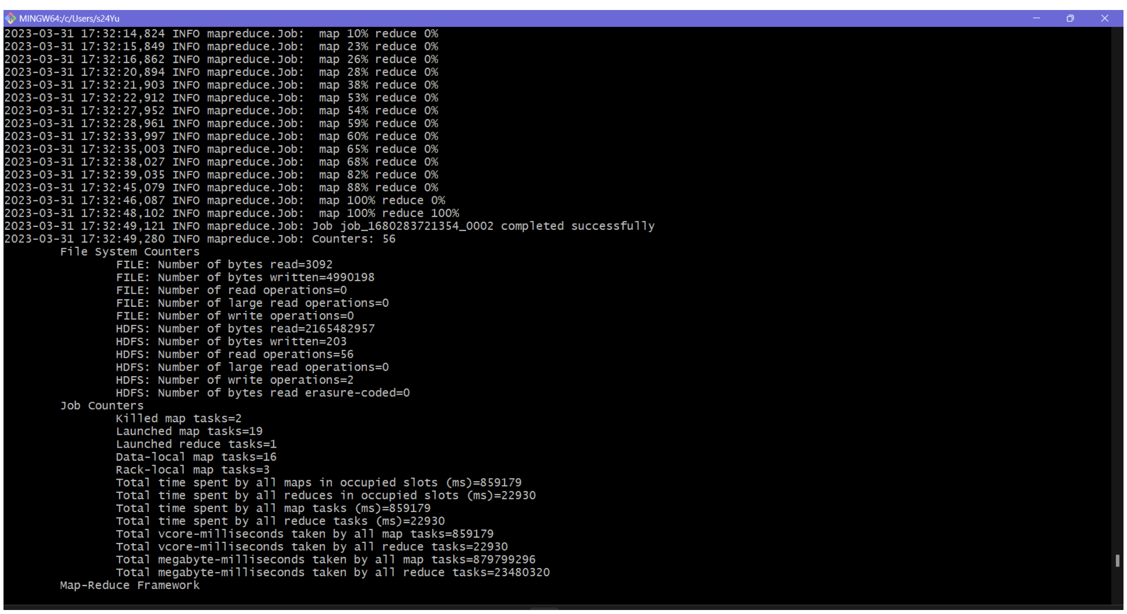


Figure 22:

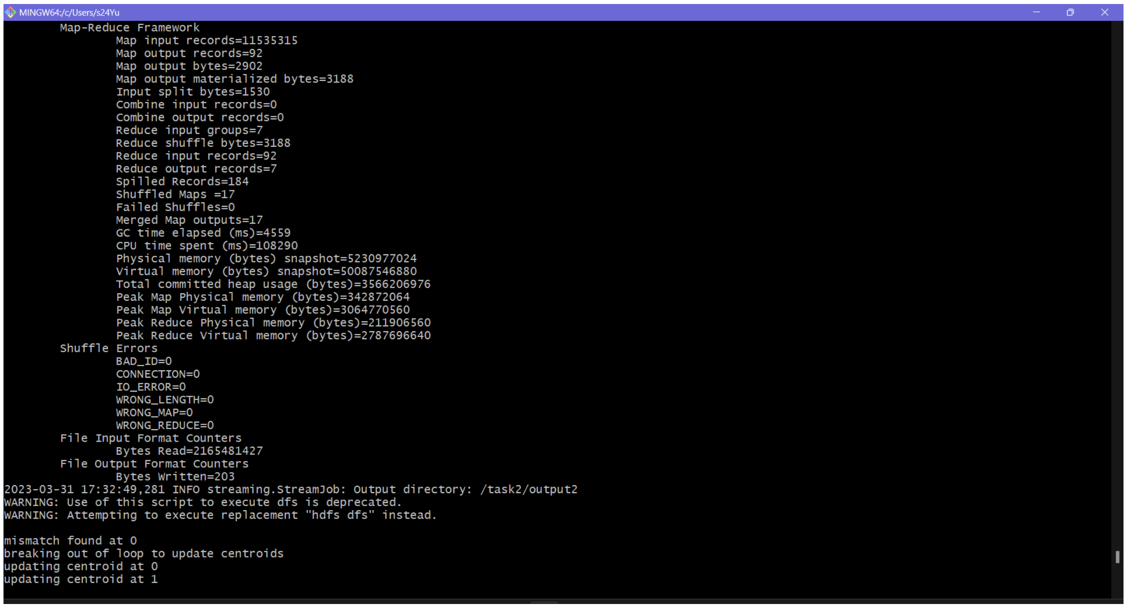


Figure 23:

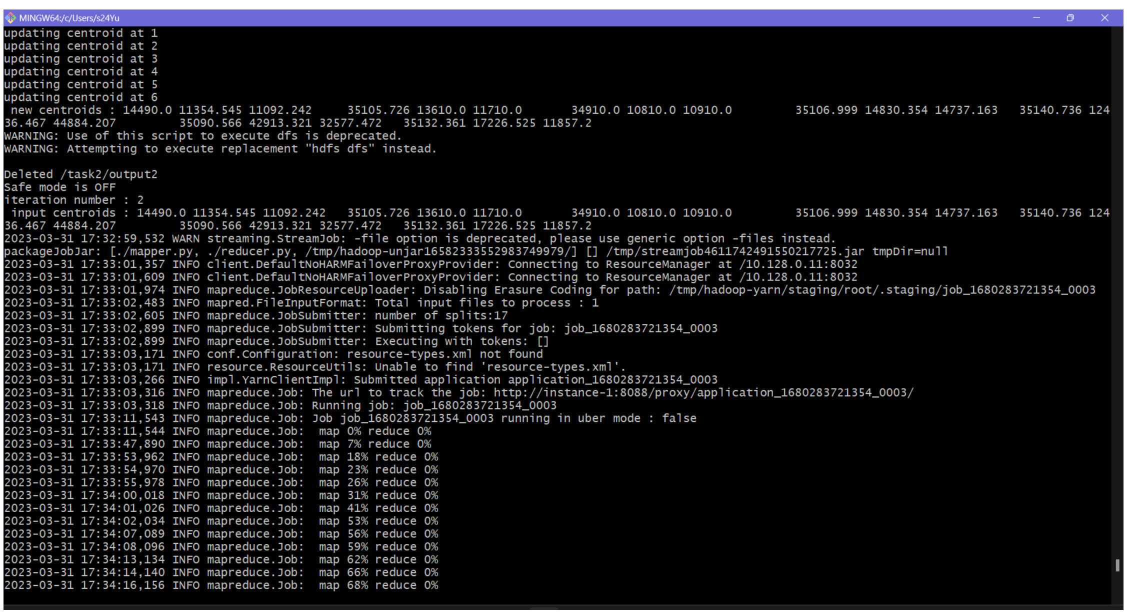


Figure 24:

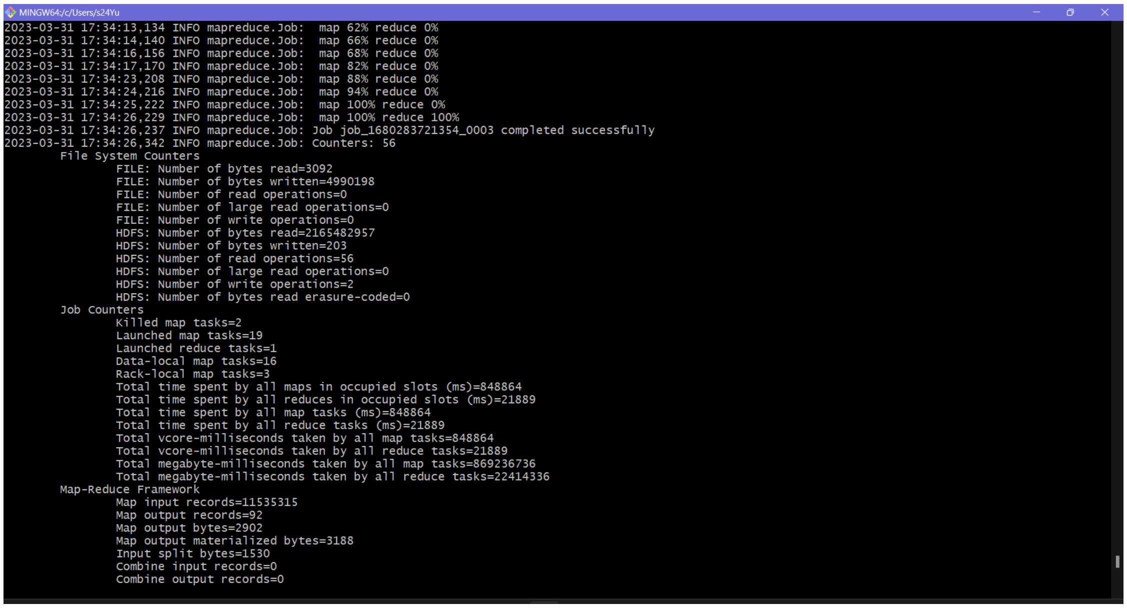

```
MINGW64:/c/Users/s24Yu
2023-03-31 17:34:13,134 INFO mapreduce.Job:  map 62% reduce 0%
2023-03-31 17:34:14,140 INFO mapreduce.Job:  map 66% reduce 0%
2023-03-31 17:34:16,156 INFO mapreduce.Job:  map 68% reduce 0%
2023-03-31 17:34:17,170 INFO mapreduce.Job:  map 82% reduce 0%
2023-03-31 17:34:23,208 INFO mapreduce.Job:  map 88% reduce 0%
2023-03-31 17:34:24,216 INFO mapreduce.Job:  map 94% reduce 0%
2023-03-31 17:34:25,222 INFO mapreduce.Job:  map 100% reduce 0%
2023-03-31 17:34:26,229 INFO mapreduce.Job:  map 100% reduce 100%
2023-03-31 17:34:26,237 INFO mapreduce.Job: Job job_1680283721354_0003 completed successfully
2023-03-31 17:34:26,342 INFO mapreduce.Job: Counters: 56
        File System Counters
                FILE: Number of bytes read=3092
                FILE: Number of bytes written=4990198
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=2165482957
                HDFS: Number of bytes written=203
                HDFS: Number of read operations=56
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Killed map tasks=2
                Launched map tasks=19
                Launched reduce tasks=1
                Data-local map tasks=16
                Rack-local map tasks=3
                Total time spent by all maps in occupied slots (ms)=848864
                Total time spent by all reduces in occupied slots (ms)=21889
                Total time spent by all map tasks (ms)=848864
                Total time spent by all reduce tasks (ms)=21889
                Total vcore-milliseconds taken by all map tasks=848864
                Total vcore-milliseconds taken by all reduce tasks=21889
                Total megabyte-milliseconds taken by all map tasks=869236736
                Total megabyte-milliseconds taken by all reduce tasks=22414336
        Map-Reduce Framework
                Map input records=11535315
                Map output records=92
                Map output bytes=2902
                Map output materialized bytes=3188
                Input split bytes=1530
                Combine input records=0
                Combine output records=0
```

Figure 25:

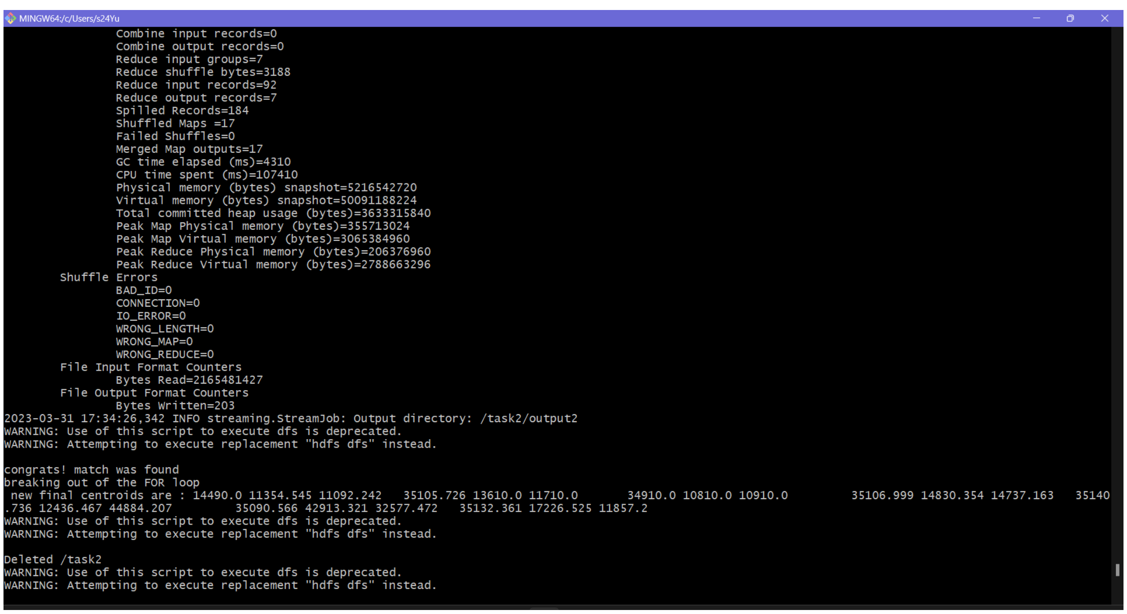
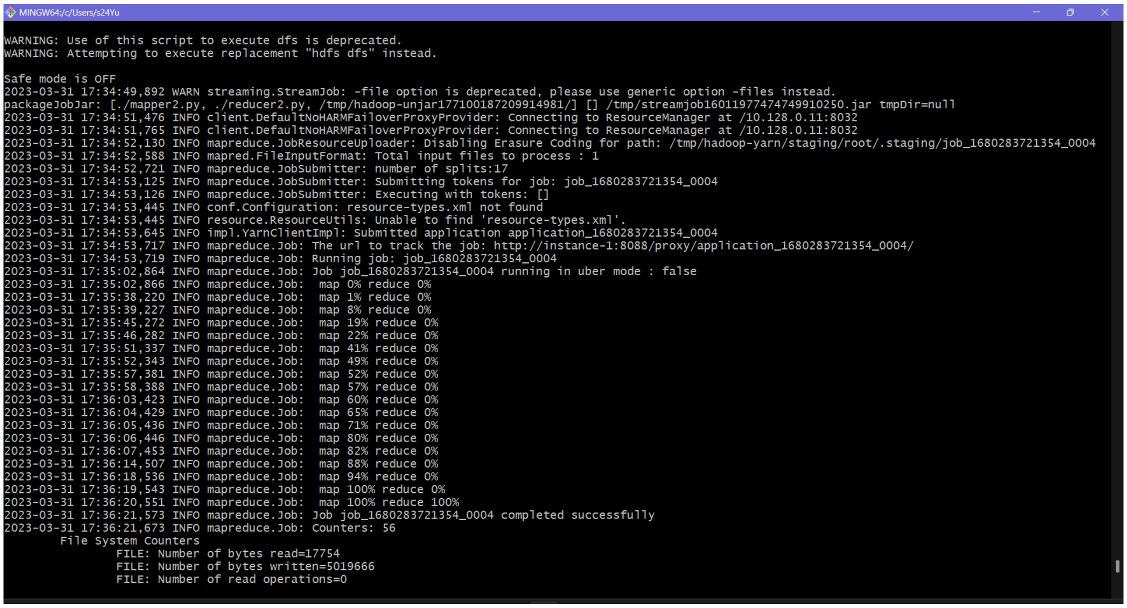
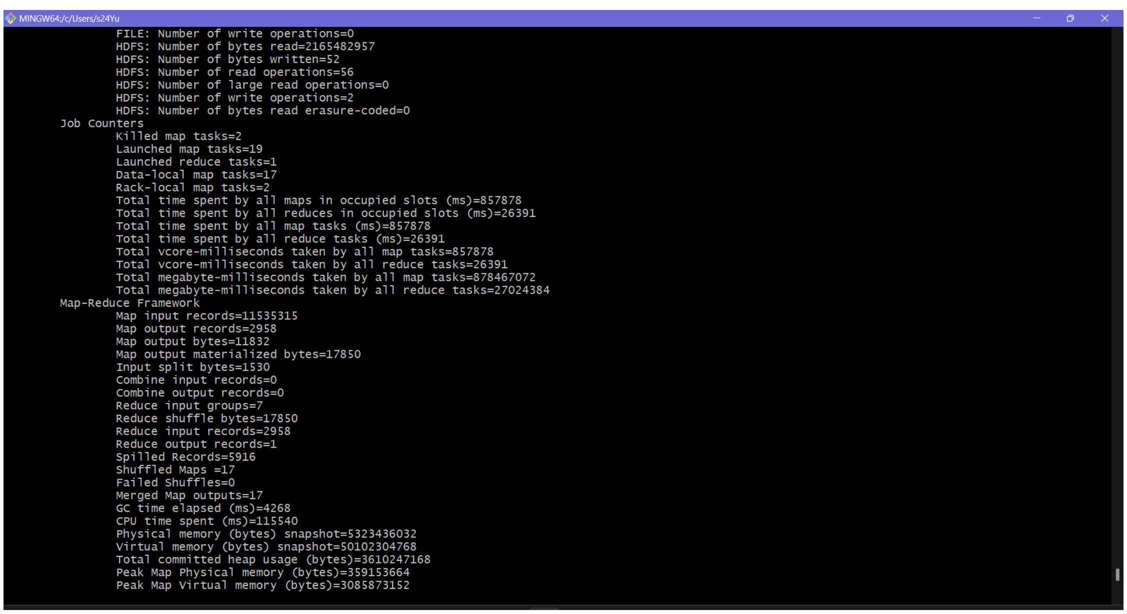

```
MINGW64:/c/Users/s24Yu
                Combine input records=0
                Combine output records=0
                Reduce input groups=7
                Reduce shuffle bytes=3188
                Reduce input records=92
                Reduce output records=7
                Spilled Records=184
                Shuffled Maps =17
                Failed Shuffles=0
                Merged Map outputs=17
                GC time elapsed (ms)=4310
                CPU time spent (ms)=107410
                Physical memory (bytes) snapshot=5216542720
                Virtual memory (bytes) snapshot=50091188224
                Total committed heap usage (bytes)=3633315840
                Peak Map Physical memory (bytes)=355713024
                Peak Map Virtual memory (bytes)=3065384960
                Peak Reduce Physical memory (bytes)=206376960
                Peak Reduce Virtual memory (bytes)=2788663296
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=2165481427
        File Output Format Counters
                Bytes Written=203
2023-03-31 17:34:26,342 INFO streaming.StreamJob: Output directory: /task2/output2
WARNING: Use of this script to execute dfs is deprecated.
WARNING: Attempting to execute replacement "hdfs dfs" instead.

congrats! match was found
breaking out of the FOR loop
 new final centroids are : 14490.0 11354.545 11092.242   35105.726 13610.0 11710.0       34910.0 10810.0 10910.0         35106.999 14830.354 14737.163   35140
.736 12436.467 44884.207         35090.566 42913.321 32577.472   35132.361 17226.525 11857.2
WARNING: Use of this script to execute dfs is deprecated.
WARNING: Attempting to execute replacement "hdfs dfs" instead.

Deleted /task2
WARNING: Use of this script to execute dfs is deprecated.
WARNING: Attempting to execute replacement "hdfs dfs" instead.
```

Figure 26:



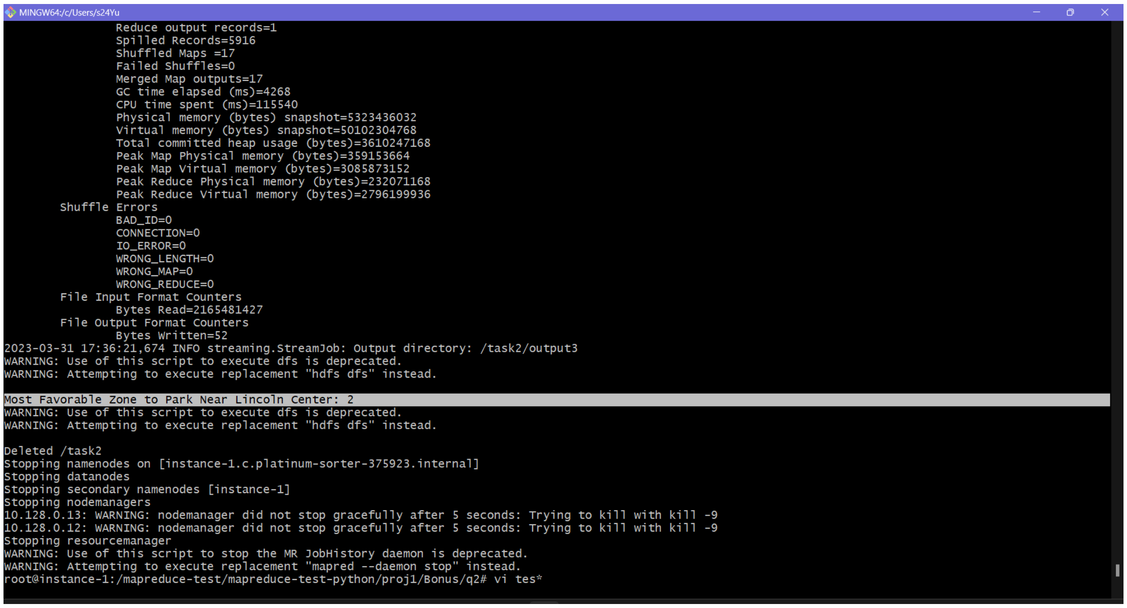
```
WARNING: Use of this script to execute dfs is deprecated.
WARNING: Attempting to execute replacement "hdfs dfs" instead.

Safe mode is OFF
2023-03-31 17:34:49,892 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [./mapper2.py, ./reducer2.py, /tmp/hadoop-unjar1771001872099144981/] [] /tmp/streamjob1601197747474991025O.jar tmpDir=null
2023-03-31 17:34:51,476 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /10.128.0.11:8032
2023-03-31 17:34:51,765 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /10.128.0.11:8032
2023-03-31 17:34:52,130 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1680283721354_0004
2023-03-31 17:34:52,588 INFO mapred.FileInputFormat: Total input files to process : 1
2023-03-31 17:34:52,721 INFO mapreduce.JobSubmitter: number of splits:17
2023-03-31 17:34:53,125 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1680283721354_0004
2023-03-31 17:34:53,126 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-03-31 17:34:53,445 INFO conf.Configuration: resource-types.xml not found
2023-03-31 17:34:53,445 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-03-31 17:34:53,645 INFO impl.YarnClientImpl: Submitted application application_1680283721354_0004
2023-03-31 17:34:53,717 INFO mapreduce.Job: The url to track the job: http://instance-1:8088/proxy/application_1680283721354_0004/
2023-03-31 17:34:53,719 INFO mapreduce.Job: Running job: job_1680283721354_0004
2023-03-31 17:35:02,864 INFO mapreduce.Job: Job job_1680283721354_0004 running in uber mode : false
2023-03-31 17:35:02,866 INFO mapreduce.Job:  map 0% reduce 0%
2023-03-31 17:35:38,220 INFO mapreduce.Job:  map 1% reduce 0%
2023-03-31 17:35:39,227 INFO mapreduce.Job:  map 8% reduce 0%
2023-03-31 17:35:45,272 INFO mapreduce.Job:  map 19% reduce 0%
2023-03-31 17:35:46,282 INFO mapreduce.Job:  map 22% reduce 0%
2023-03-31 17:35:51,337 INFO mapreduce.Job:  map 41% reduce 0%
2023-03-31 17:35:52,343 INFO mapreduce.Job:  map 49% reduce 0%
2023-03-31 17:35:57,381 INFO mapreduce.Job:  map 52% reduce 0%
2023-03-31 17:35:58,388 INFO mapreduce.Job:  map 57% reduce 0%
2023-03-31 17:36:03,423 INFO mapreduce.Job:  map 60% reduce 0%
2023-03-31 17:36:04,429 INFO mapreduce.Job:  map 65% reduce 0%
2023-03-31 17:36:05,436 INFO mapreduce.Job:  map 71% reduce 0%
2023-03-31 17:36:06,446 INFO mapreduce.Job:  map 80% reduce 0%
2023-03-31 17:36:07,453 INFO mapreduce.Job:  map 82% reduce 0%
2023-03-31 17:36:14,507 INFO mapreduce.Job:  map 88% reduce 0%
2023-03-31 17:36:18,536 INFO mapreduce.Job:  map 94% reduce 0%
2023-03-31 17:36:19,543 INFO mapreduce.Job:  map 100% reduce 0%
2023-03-31 17:36:20,551 INFO mapreduce.Job:  map 100% reduce 100%
2023-03-31 17:36:21,573 INFO mapreduce.Job: Job job_1680283721354_0004 completed successfully
2023-03-31 17:36:21,673 INFO mapreduce.Job: Counters: 56
        File System Counters
                FILE: Number of bytes read=17754
                FILE: Number of bytes written=5019666
                FILE: Number of read operations=0
```

Figure 27:



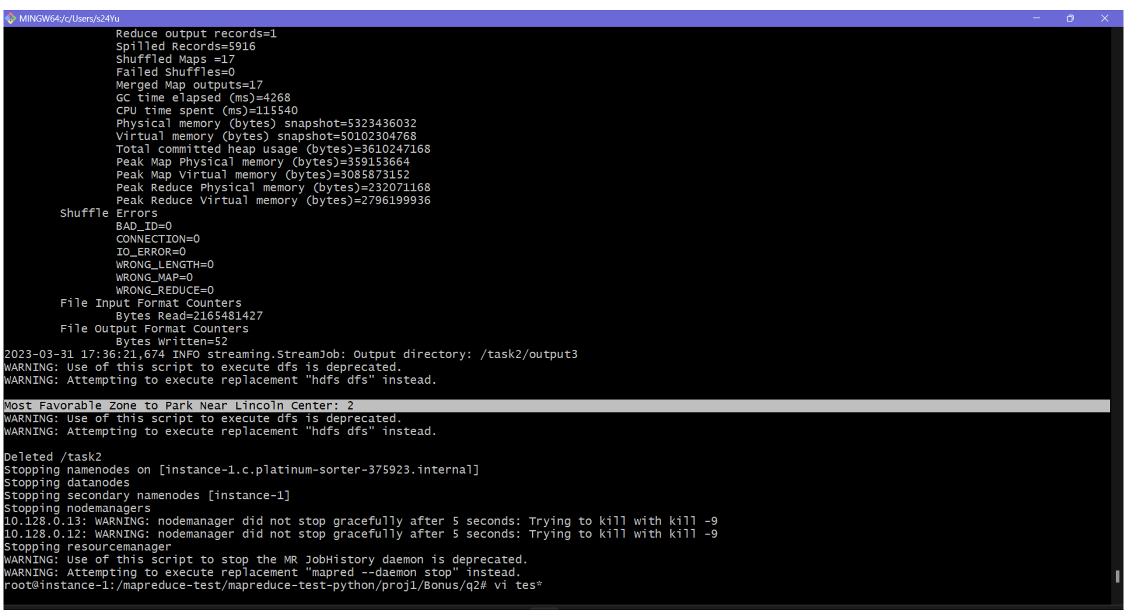
```
                FILE: Number of write operations=0
                HDFS: Number of bytes read=2165482957
                HDFS: Number of bytes written=52
                HDFS: Number of read operations=56
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Killed map tasks=2
                Launched map tasks=19
                Launched reduce tasks=1
                Data-local map tasks=17
                Rack-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=857878
                Total time spent by all reduces in occupied slots (ms)=26391
                Total time spent by all map tasks (ms)=857878
                Total time spent by all reduce tasks (ms)=26391
                Total vcore-milliseconds taken by all map tasks=857878
                Total vcore-milliseconds taken by all reduce tasks=26391
                Total megabyte-milliseconds taken by all map tasks=878467072
                Total megabyte-milliseconds taken by all reduce tasks=27024384
        Map-Reduce Framework
                Map input records=11535315
                Map output records=2958
                Map output bytes=11832
                Map output materialized bytes=17850
                Input split bytes=1530
                Combine input records=0
                Combine output records=0
                Reduce input groups=7
                Reduce shuffle bytes=17850
                Reduce input records=2958
                Reduce output records=1
                Spilled Records=5916
                Shuffled Maps =17
                Failed Shuffles=0
                Merged Map outputs=17
                GC time elapsed (ms)=4268
                CPU time spent (ms)=115540
                Physical memory (bytes) snapshot=5323436032
                Virtual memory (bytes) snapshot=50102304768
                Total committed heap usage (bytes)=3610247168
                Peak Map Physical memory (bytes)=359153664
                Peak Map Virtual memory (bytes)=3085873152
```

Figure 28:



```
                Reduce output records=1
                Spilled Records=5916
                Shuffled Maps =17
                Failed Shuffles=0
                Merged Map outputs=17
                GC time elapsed (ms)=4268
                CPU time spent (ms)=115540
                Physical memory (bytes) snapshot=5323436032
                Virtual memory (bytes) snapshot=50102304768
                Total committed heap usage (bytes)=3610247168
                Peak Map Physical memory (bytes)=359153664
                Peak Map Virtual memory (bytes)=3085873152
                Peak Reduce Physical memory (bytes)=232071168
                Peak Reduce Virtual memory (bytes)=2796199936
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=2165481427
        File Output Format Counters
                Bytes Written=52
2023-03-31 17:36:21,674 INFO streaming.StreamJob: Output directory: /task2/output3
WARNING: Use of this script to execute dfs is deprecated.
WARNING: Attempting to execute replacement "hdfs dfs" instead.

Most Favorable Zone to Park Near Lincoln Center: 2
WARNING: Use of this script to execute dfs is deprecated.
WARNING: Attempting to execute replacement "hdfs dfs" instead.

Deleted /task2
Stopping namenodes on [instance-1.c.platinum-sorter-375923.internal]
Stopping datanodes
Stopping secondary namenodes [instance-1]
Stopping nodemanagers
10.128.0.13: WARNING: nodemanager did not stop gracefully after 5 seconds: Trying to kill with kill -9
10.128.0.12: WARNING: nodemanager did not stop gracefully after 5 seconds: Trying to kill with kill -9
Stopping resourcemanager
WARNING: Use of this script to stop the MR JobHistory daemon is deprecated.
WARNING: Attempting to execute replacement "mapred --daemon stop" instead.
root@instance-1:/mapreduce-test/mapreduce-test-python/proj1/Bonus/q2# vi tes*
```

Figure 29:



```
                Reduce output records=1
                Spilled Records=5916
                Shuffled Maps =17
                Failed Shuffles=0
                Merged Map outputs=17
                GC time elapsed (ms)=4268
                CPU time spent (ms)=115540
                Physical memory (bytes) snapshot=5323436032
                Virtual memory (bytes) snapshot=50102304768
                Total committed heap usage (bytes)=3610247168
                Peak Map Physical memory (bytes)=359153664
                Peak Map Virtual memory (bytes)=3085873152
                Peak Reduce Physical memory (bytes)=232071168
                Peak Reduce Virtual memory (bytes)=2796199936
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=2165481427
        File Output Format Counters
                Bytes Written=52
2023-03-31 17:36:21,674 INFO streaming.StreamJob: Output directory: /task2/output3
WARNING: Use of this script to execute dfs is deprecated.
WARNING: Attempting to execute replacement "hdfs dfs" instead.

Most Favorable Zone to Park Near Lincoln Center: 2
WARNING: Use of this script to execute dfs is deprecated.
WARNING: Attempting to execute replacement "hdfs dfs" instead.

Deleted /task2
Stopping namenodes on [instance-1.c.platinum-sorter-375923.internal]
Stopping datanodes
Stopping secondary namenodes [instance-1]
Stopping nodemanagers
10.128.0.13: WARNING: nodemanager did not stop gracefully after 5 seconds: Trying to kill with kill -9
10.128.0.12: WARNING: nodemanager did not stop gracefully after 5 seconds: Trying to kill with kill -9
Stopping resourcemanager
WARNING: Use of this script to stop the MR JobHistory daemon is deprecated.
WARNING: Attempting to execute replacement "mapred --daemon stop" instead.
root@instance-1:/mapreduce-test/mapreduce-test-python/proj1/Bonus/q2# vi tes*
```

Figure 30:

We provided the answer and explanation comprehensively for each question. Nowadays, Mapreduce is a popular tool for big data processing. Mapreduce is particularly useful for tasks that involve processing a large amount of unstructured data, such as text documents. Log files and web page. It can be used for a wide range of applications, including data mining, machine learning and more. However, it still has some limitations and restrictions. We should carefully consider these limitations when designing and implementing MapReduce-based solutions and be aware of the potential trade-offs in using this framework.