DAT470/DIT065 Assignment 3

Xinyan Liu
liuxinya@chalmers.se

Leah Wanja Ndirangu
leahw@chalmers.se

2025-05-03

Problem 1

(a) Describe the operations that the programmer needs to define, using pseudocode. Include a clear description of the input and the output of each operation.

Below is the implementation of the `map` function in Python:

```
function mapper(_, line):
    # Ignore header row that contains the word "
        constellation"
    if "constellation" is in line (case-insensitive):
        return

    # Split the line into fields using a comma as the
        delimiter
    fields = split(line, ",")

    # Extract relevant fields
    constellation = fields[0]  # Constellation name
    star = fields[1]           # Star name
    mineral_value = float(fields[5])  # Mineral value (
        convert to float)

    # Combine star and constellation to form the star system
         name
    star_system = concatenate(star, " ", constellation)

    # Emit the key-value pair
    emit(star_system, integer(mineral_value))
```

Input:

A single line of text from the dataset (e.g., a row in `planets.csv`). Each line contains comma-separated fields such as:

- **Field 0:** Constellation (e.g., `"Centauri"`).
- **Field 1:** Star (e.g., `"Alpha"`).
- **Field 5:** Mineral value (RU) (e.g., `"3417"`).

Output:

- **Key:** The name of the star system, which is a combination of the `Star` and `Constellation` (e.g., `"Alpha Centauri"`).
- **Value:** The mineral value (converted to an integer) of the planet or moon (e.g., `3417`).

Below is the implementation of the `reducer` function in Python:

```
function reducer(self, key, values):
    name_list = ["Capella", "Alpha Cancri", "Gamma Sagittae"
        , "Beta Lyrae", "Alpha Geminorum"]
    total_value = sum(values)
    for name in name_list:
        if name in key:
            yield (key, int(total_value))
            break
```

Input:

- *key*: A string representing the constellation and star name of an astronomical object. For example, "Alpha Cancri", or "Gamma Sagittae".
- *values*: Mineral values associated with the given *key*, denoted as $\{v_1, v_2, \ldots, v_n\}$.

Output:

- $(name, total\_value)$: A tuple where
  - *name* is the name (constellation + stars, or each one of them) in `name_list` such that *name* is a substring of *key*.
  - $total\_value = \sum_{i=1}^{n} v_i$ is the total mineral values associated with *key*.

Motivation for Substring Checking

In practical astronomical datasets, the input *key* may not exactly match the standard names listed in `name_list`. For example, `name_list` may contain only "Capella", while the dataset contains the more specific "Prime Capella". Therefore, Substring checking ensures that:

- Even if only a partial name (e.g., "Capella") is present in `name_list`, it will still match the full name (e.g., "Prime Capella") in the dataset, and output all name and mineral values that match it.
- This approach prevents data loss due to naming inconsistencies and increases the robustness and convenience of the query.

(b) Draw a diagram that shows the data flow of the program, across the different operations. You may assume that initially the dataset is spread across different nodes and the results potentially end on different nodes. The diagram should clearly show when data is transferred between nodes.

The following picture 1 shows the data flow of the program. In the program, the input csv file may be divided into multiple splits, with many map workers and reduce workers as well as multiple outputs. Here only a portion of this is illustrated in the graph (Because the image is quite large, you need to zoom in to see it clearly). The graph is also attahed as flow1.svg.

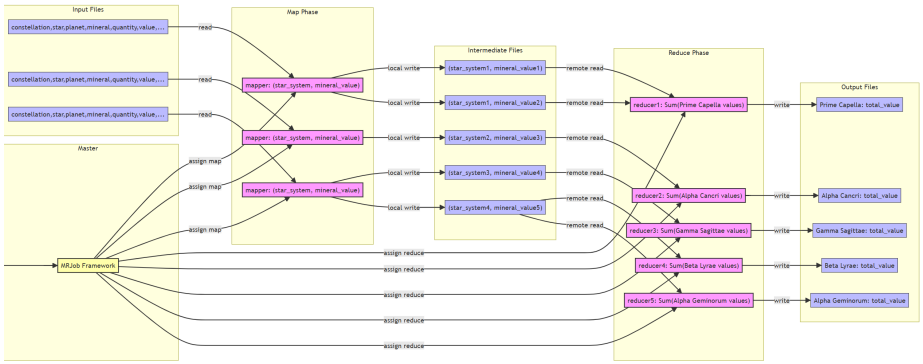


Figure 1: Data flows for the program

(c) Implement the program using MRJob. Use the file assignment3 problem1 -skeleton.py as your starting point. In your report, include a table that contains the total mineral values for the following systems: Capella, Alpha Cancri, Gamma Sagittae, Beta Lyrae, and Alpha Geminorum

The following table 1 shows our results:

Star System	**Mineral Value (RU)**
Beta Lyrae	2431
Alpha Geminorum	4257
Gamma Sagittae	2899
Alpha Cancri	2082
Capella	1248

Table 1: Mineral values for various star systems.

And the screenshot 2 below is our running result for this question

```
liuxinya@minerva:~/lab3/filecon$ tail -f stdout
Requirement already satisfied: mrjob in /data/users/liuxinya/lab3/lib/python3.12/site-package
Requirement already satisfied: PyYAML>=3.10 in /data/users/liuxinya/lab3/lib/python3.12/site-
2)
Requirement already satisfied: setuptools in /data/users/liuxinya/lab3/lib/python3.12/site-pa
No configs found; falling back on auto-configuration
No configs specified for local runner
Creating temp directory /tmp/assignment3_problem1_skeleton.liuxinya.20250429.144217.210891
Running step 1 of 1...
job output is in /tmp/assignment3_problem1_skeleton.liuxinya.20250429.144217.210891/output
Streaming final output from /tmp/assignment3_problem1_skeleton.liuxinya.20250429.144217.21089
"Beta Lyrae"    2431
"Alpha Geminorum"       4257
"Gamma Sagittae"        2899
"Alpha Cancri"  2082
"Prime Capella" 1248
Removing temp directory /tmp/assignment3_problem1_skeleton.liuxinya.20250429.144217.210891...
```

Figure 2: Running Result on the Terminal

Problem 2

(a) Create a diagram that shows the data flow in the new program. As before, assume the dataset is distributed across the nodes. However, in this case, the final end result, an array of k elements, must end up on a single node

The following picture 3 shows the data flow of the program. Except for the last part, the other parts are the same as the parts in Problem 1.(Because the image is quite large, you need to zoom in to see it clearly). The graph is also attached as flow2.svg.

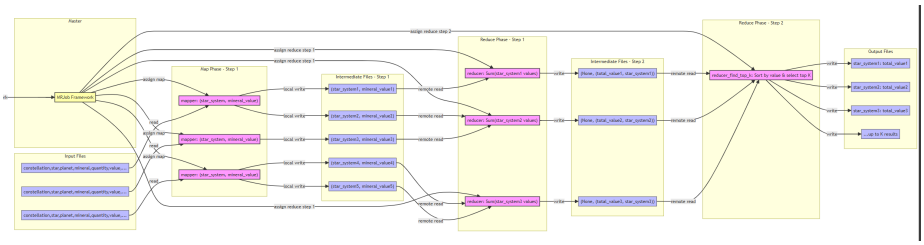


Figure 3: Data Flow

(b)Implement the new program using MRJob. You probably need more than one step. Include a table that contains the names and mineral values of the 10 most valuable systems in your report.

The table 2 below is our output for the names and mineral values of the 10 most valuable systems.

Table 2: Stellar Data

Star Name	Value
Beta Scorpii	12680
Delta Tauri	12080
Alpha Ceti	11739
Alpha Centauri	11476
Delta Aurigae	11005
Beta Cephei	10447
Prime Zeeman	10421
Zeta Vulpeculae	9404
Beta Normae	9153
Delta Brahe	9100

And the screenshot 4 below is our running result for this question

```
liuxinya@minerva:~/lab3/filecon$ tail -f stdout2
Requirement already satisfied: mrjob in /data/users/liuxinya/lab3/lib/python3.12/site-packages (0.7.4)
Requirement already satisfied: PyYAML>=3.10 in /data/users/liuxinya/lab3/lib/python3.12/site-packages (from mrjob) (6.0.2)
Requirement already satisfied: setuptools in /data/users/liuxinya/lab3/lib/python3.12/site-packages (80.0.0)
No configs found; falling back on auto-configuration
No configs specified for local runner
Creating temp directory /tmp/assignment3_problem2.liuxinya.20250429.150248.550503
Running step 1 of 2...
Running step 2 of 2...
job output is in /tmp/assignment3_problem2.liuxinya.20250429.150248.550503/output
Streaming final output from /tmp/assignment3_problem2.liuxinya.20250429.150248.550503/output...
"Beta Scorpii"  12680
"Delta Tauri"   12080
"Alpha Ceti"    11739
"Alpha Centauri"        11476
"Delta Aurigae" 11005
"Beta Cephei"   10447
"Prime Zeeman"  10421
"Zeta Vulpeculae"       9404
"Beta Normae"   9153
"Delta Brahe"   9100
Removing temp directory /tmp/assignment3_problem2.liuxinya.20250429.150248.550503...
```

Figure 4: Running Result on the Terminal

Problem 3

(a) Describe a MapReduce algorithm that determines the Twitter ID of the person who follows the largest amount of users, the number of users this person follows, the average number of users followed, and the number of accounts that do not follow anyone. Describe the user-defined operations in pseudocode, and also provide a diagram that shows the flow of data.

Mapper

```
FUNCTION mapper(_, line)
    // Input: Key: line number (ignored), Value: String line
        in format "user_id: followed_user1 followed_user2
        ..."
    // Output: Key-Value pairs (user, 0) and (follow, 1) for
        each follower relationship

    user_follow = SPLIT line BY ':'
    user = STRIP user_follow[0]
    follows = SPLIT STRIP(user_follow[1]) BY ' '

    // Emit the user with a count of 0 to ensure all users
       are counted
    EMIT (user, 0)

    // Emit each followed user with a count of 1
    FOR follow IN follows DO
        EMIT (follow, 1)
    END FOR
END FUNCTION
```

Reducer

```
FUNCTION reducer(key, values)
    // Input: Key: user_id (string), Values: counts of how
       many times this person follow other
    // Output: Key: NULL, Value: (user_id, follower_count)
       tuple, follower_count means the total number of
       people that people is following

    // Sum the values to get follower count for this user
    follower_count = SUM(values)

    // Pass the user and their follower count to the next
       step
    EMIT (NULL, (key, follower_count))
END FUNCTION
```

Reducer2

```
FUNCTION reducer2(_, values)
    // Input: Key: NULL, Values: List of (user_id,
        follower_count) tuples for all users
    // Output: Final statistics as key-value pairs for most
        followers, average, and users with no followers

    user_list = LIST(values)

    // Find user with maximum followers
    max_pair = FIND_MAX(user_list, COMPARING follower_count)
    max_id = max_pair[0]
    max_count = max_pair[1]

    // Count users with no followers
    no_follow_count = COUNT(user IN user_list WHERE user[1]
        = 0)

    // Calculate average followers
    total_followers = SUM(user[1] FOR EACH user IN user_list
        )
    total_users = LENGTH(user_list)

    IF total_users > 0 THEN
        avg_followers = total_followers / total_users
    ELSE
        avg_followers = 0
    END IF

    // Emit final statistics
    EMIT ("most followers id", max_id)
    EMIT ("most followers", max_count)
    EMIT ("average followers", avg_followers)
    EMIT ("count no followers", no_follow_count)
END FUNCTION
```

The graph 5 below shows the flow of data.(Because the image is quite large, you need to zoom in to see it clearly). The graph is also attahed as flow3.svg.

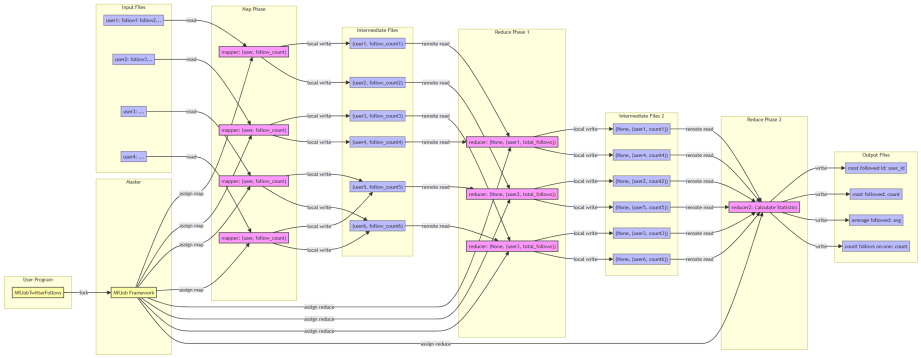


Figure 5: Flow of Data

(b) The file mrjob twitter follows.py contains the interface for a MRJob implementation that where you will need to fill in the blanks by implementing your algorithm. Implement the algorithm.

```
#!/usr/bin/env python3

from mrjob.job import MRJob
from mrjob.step import MRStep

class MRJobTwitterFollows(MRJob):
    # The final (key,value) pairs returned by the class
      should be
    #
    # yield ('most followed id', ???)
    # yield ('most followed', ???)
    # yield ('average followed', ???)
    # yield ('count follows no-one', ???)
    #
    # You will, of course, need to replace ??? with a
      suitable expression
    def mapper(self, _, line):
        user_follow = line.split(':')
        user = user_follow[0].strip()
        follow = user_follow[1].strip().split()
        yield (user, len(follow))

    def reducer(self, key, values):
        yield (None, (key, sum(values)))

    def reducer2(self, _, values):
        user_list = list(values)

        max_pair = max(user_list, key=lambda x: x[1],
            default=(None, -1))
```

```
        max_id, max_count = max_pair

        no_follow = sum(1 for v in user_list if v[1] == 0)

        total_follow = sum(v[1] for v in user_list)
        total_users = len(user_list)

        yield ('most followed id', max_id)
        yield ('most followed', max_count)
        yield ('average followed', total_follow /
            total_users if total_users else 0)
        yield ('count follows no-one', no_follow)

    def steps(self):
        return [
            MRStep(mapper=self.mapper,
                   reducer=self.reducer),
            MRStep(reducer=self.reducer2)
        ]


if __name__ == '__main__':
    MRJobTwitterFollows.run()
```

The picture 6 below shows our running result on 1k, 10k,100k datasets, which match the table in the manual.

```
liuxinya@minerva:~/lab3/filecon$ tail -f stdout
Requirement already satisfied: mrjob in /data/users/liuxinya/lab3/lib/python3.12/site-package
Requirement already satisfied: PyYAML>=3.10 in /data/users/liuxinya/lab3/lib/python3.12/site-
2)
Requirement already satisfied: setuptools in /data/users/liuxinya/lab3/lib/python3.12/site-pa
No configs found; falling back on auto-configuration
No configs specified for local runner
Creating temp directory /tmp/assignment3_problem1_skeleton.liuxinya.20250429.144217.210891
Running step 1 of 1...
job output is in /tmp/assignment3_problem1_skeleton.liuxinya.20250429.144217.210891/output
Streaming final output from /tmp/assignment3_problem1_skeleton.liuxinya.20250429.144217.21089
"Beta Lyrae"    2431
"Alpha Geminorum"       4257
"Gamma Sagittae"        2899
"Alpha Cancri"  2082
"Prime Capella" 1248
Removing temp directory /tmp/assignment3_problem1_skeleton.liuxinya.20250429.144217.210891...
```

Figure 6: Running Result on 1K 10K 100K dataset

(c) Measure the scalability of your algorithm on 1, 2, 4, . . . , 32 cores. Plot the empirical speedup as the function of cores. Use the 10M dataset for scalability experiments. In addition to the plot, report the single-core runtime on the dataset.

To plot the data:

```
import matplotlib.pyplot as plt
```

```
import re

filename = "/data/users/leahw/lab3/times_20250504_195223.txt
    "  # actual output filename

workers = []
times = []

with open(filename, 'r') as f:
    current_workers = None
    for line in f:
        if line.startswith("#"):
            continue  # Skip comment line
        if "Number of workers:" in line:
            match = re.search(r"Number of workers:\s*(\d+)",
                line)
            if match:
                current_workers = int(match.group(1))
        elif "Time elapsed:" in line:
            match = re.search(r"Time elapsed:\s*([0-9.]+)",
                line)
            if match and current_workers is not None:
                elapsed_time = float(match.group(1))
                workers.append(current_workers)
                times.append(elapsed_time)

# Sort by workers for clean plotting
workers, times = zip(*sorted(zip(workers, times)))

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(workers, times, marker='o')
plt.xlabel('Number of Workers')
plt.ylabel('Execution Time (seconds)')
plt.title('Execution Time vs Number of Workers')
plt.grid(True)
plt.xticks(workers)
plt.tight_layout()

# Save plot as PNG image
plt.savefig("execution_time_vs_workers.png")
print("Plot saved as 'execution_time_vs_workers.png'")
```

Bash script for execution:

```
#! /bin/bash

#SBATCH --cpus-per-task=4
#SBATCH --mem=16G
#SBATCH -t 00:30:00


source ./.venv/bin/activate
# pip3 install matplotlib

OUTPUT="times_$(date +%Y%m%d_%H%M%S).txt"

DATA=/data/courses/2025_dat470_dit066/twitter/twitter-2010_10M.txt

echo "#workers time" > $OUTPUT
for w in 1 2 4 8 16 32 64; do
    t=$(python3 mrjob_twitter_measure_followers.py -w $w $DATA 2>&1)
    echo "$w $t" >> $OUTPUT
done
```

Here is the plot for speed up with different numbers of workers, and **the single-core runtime**on the dataset is 10.429279327392578s.

Figure 7: Running of Data Flow

(d) Run your implementation (with a large number of cores) on the full dataset. Report the values (ID and number of accounts followed by the account that has follows most accounts, average number of accounts followed, and the number of accounts that follow no-one.

Results:

most followed id	813286
most followed	770155
average followed	35.25297882010159
count follows no-one	5963082

Problem 4

(a) Describe a MapReduce algorithm determines the Twitter ID of the person who has the largest amount of followers, the number of followers this person has, the average number of followers, and the number of accounts that do not have any followers. Describe the user-defined operations in pseudocode, and also provide a diagram that shows the flow of data. (2 pts)

Method 1: mapper

Algorithm 1 mapper

```
function MAPPER(_, line)
    Input: A line of text in format "user: follow1 follow2 ..."
    Output: Key-value pairs (user, 0) and (follow, 1) for each follower
    user_follow ← split(line, ":")
    user ← strip(user_follow[0])
    follows ← split(strip(user_follow[1]))
    yield (user, 0)        ▷ Ensure all users are counted, even with 0 followers
    for follow in follows do
        yield (follow, 1)                  ▷ Each follow represents one follower
    end for
end function
```

Method 2: combiner

Algorithm 2 combiner

```
function COMBINER(key, values)
    Input: A user ID as key and a list of follower counts as values
    Output: Key-value pair (key, sum_of_values) where sum_of_values is
    the local sum of follower counts
    sum_of_values ← sum(values)
    yield (key, sum_of_values)
end function
```

Method 3: reducer

Algorithm 3 reducer

function REDUCER($key, values$)
    Input: A user ID as key and whether this person is followed by others in this line as values (0 or 1)
    Output: Key-value pair $(None, (user, follower\_count))$ "follower count" means how many times this person is followed by others in all dataset
    $follower\_count \leftarrow \text{sum}(values)$
    yield $(None, (key, follower\_count))$
end function

Method 4: reducer2

Algorithm 4 reducer2

function REDUCER2($\_, values$)
    Input: A key (None) and a list of $(user, follower\_count)$ pairs
    Output: Four key-value pairs with statistics about followers
    $user\_list \leftarrow \text{list}(values)$
    $max\_pair \leftarrow \max(user\_list, \text{key} = \lambda x : x[1], \text{default} = (None, -1))$
    $max\_id, max\_count \leftarrow max\_pair$
    $no\_follow \leftarrow \sum_{v \in user\_list} 1 \text{ if } v[1] = 0 \text{ else } 0$
    $total\_follow \leftarrow \sum_{v \in user\_list} v[1]$
    $total\_users \leftarrow \text{length}(user\_list)$
    yield $('most\ followers\ id', max\_id)$
    yield $('most\ followers', max\_count)$
    yield $('average\ followers', total\_follow/total\_users)$ ▷ If $total\_users > 0$
    yield $('count\ no\ followers', no\_follow)$
end function

The graph 5 below shows the flow of data.(Because the image is quite large, you need to zoom in to see it clearly). The graph is also attahed as flow4.svg.

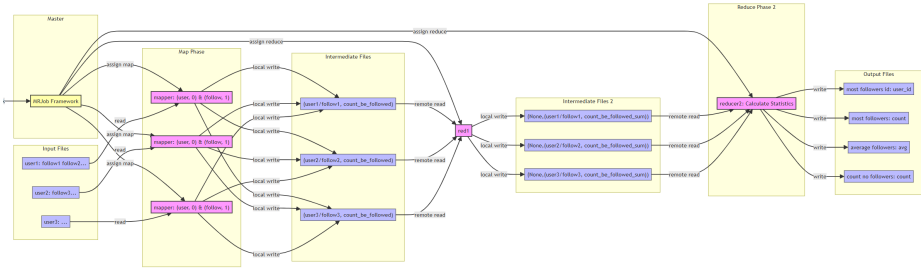


Figure 8: Running of Data Flow

(b) The file mrjob twitter followers.py contains the interface for a MRJob implementation that where you will need to fill in the blanks by implementing your algorithm. Implement the algorithm

```
 #!/usr/bin/env python3


from mrjob.job import MRJob
from mrjob.step import MRStep

class MRJobTwitterFollowers(MRJob):
    # The final (key,value) pairs returned by the class
        should be
    #
    # yield ('most followers id', ???)
    # yield ('most followers', ???)
    # yield ('average followers', ???)
    # yield ('count no followers', ???)
    #
    # You will, of course, need to replace ??? with a
        suitable expression

    def mapper(self,_,line):
        user_follow = line.split(':')
        user = user_follow[0].strip()
        follows = user_follow[1].strip().split()
        yield (user,0)
        for follow in follows:
            yield (follow,1)

    def reducer(self,key,values):
        yield (None, (key, sum(values)))

    def reducer2(self,_,values):
        user_list = list(values)

        max_pair = max(user_list, key=lambda x: x[1],
            default=(None, -1))
        max_id, max_count = max_pair

        no_follow = sum(1 for v in user_list if v[1] == 0)

        total_follow = sum(v[1] for v in user_list)
        total_users = len(user_list)

        yield ('most followers id', max_id)
        yield ('most followers', max_count)
        yield ('average followers', total_follow /
            total_users if total_users else 0)
        yield ('count no followers', no_follow)

    def steps(self):
```

```
        return [
            MRStep(mapper=self.mapper,
                   reducer=self.reducer),
            MRStep(reducer=self.reducer2)
        ]

if __name__ == '__main__':
    MRJobTwitterFollowers.run()
```

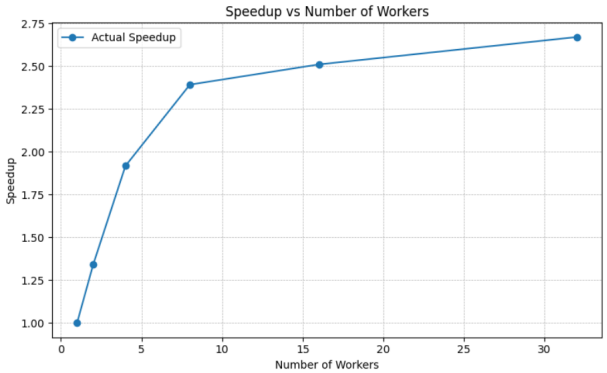
The picture 9 below shows our running result on 1k, 10k,100k datasets, which match the table in the manual.

```
liuxinya@minerva:~/lab3/filecon$ tail -f reuksr
Creating temp directory /tmp/mrjob_twitter_followers.liuxinya.20250503.221212.372113
Running step 1 of 2...
Running step 2 of 2...
job output is in /tmp/mrjob_twitter_followers.liuxinya.20250503.221212.372113/output
Streaming final output from /tmp/mrjob_twitter_followers.liuxinya.20250503.221212.372113/output...
"most followers id"     "26493932"
"most followers"        1
"average followers"     0.001
"count no followers"    999
Removing temp directory /tmp/mrjob_twitter_followers.liuxinya.20250503.221212.372113...
No configs found; falling back on auto-configuration
No configs specified for inline runner
Creating temp directory /tmp/mrjob_twitter_followers.liuxinya.20250503.221214.248614
Running step 1 of 2...
Running step 2 of 2...
job output is in /tmp/mrjob_twitter_followers.liuxinya.20250503.221214.248614/output
Streaming final output from /tmp/mrjob_twitter_followers.liuxinya.20250503.221214.248614/output...
"most followers id"     "12926542"
"most followers"        20
"average followers"     0.0163
"count no followers"    9904
Removing temp directory /tmp/mrjob_twitter_followers.liuxinya.20250503.221214.248614...
No configs found; falling back on auto-configuration
No configs specified for inline runner
Creating temp directory /tmp/mrjob_twitter_followers.liuxinya.20250503.221216.322944
Running step 1 of 2...
Running step 2 of 2...
job output is in /tmp/mrjob_twitter_followers.liuxinya.20250503.221216.322944/output
Streaming final output from /tmp/mrjob_twitter_followers.liuxinya.20250503.221216.322944/output...
"most followers id"     "24083587"
"most followers"        754
"average followers"     0.08054
"count no followers"    95958
Removing temp directory /tmp/mrjob_twitter_followers.liuxinya.20250503.221216.322944...
```

Figure 9: Running Result on 1K 10K 100K dataset

(c) Measure the scalability of your algorithm on 1, 2, 4, . . . , 32 cores. Plot the empirical speedup as the function of cores. Use the 10M dataset for scalability experiments. In addition to the plot, report the single-core runtime on the dataset.

The picture below is our speed up for this question, and the **single running time is 746.211**



(d) Run your implementation (with a large number of cores) on the full dataset. Report the values (ID and number of followers by the account that has most followers, average number of followers, and the number of accounts that have no followers.

most followers id	19058681
most followers	2997469 ers
average followers	35.25297882010159
count follows no-one	1548949