Big Data Assignment: Advanced Techniques and Configurations in Hadoop MapReduce

Task 0: Setting up Hadoop and HDFS

```
(base) mukundkomati@Mukunds-MacBook-Pro hands_on_hadoop2 % scp movie_new.xlsx rating_new.xlsx exouser@149.165.159.24:~/
exouser@149.165.159.24's password:
movie_new.xlsx                                                                100% 1260KB   6.2MB/s   00:00
rating_new.xlsx                                                               100%   37MB   2.8MB/s   00:13
(base) mukundkomati@Mukunds-MacBook-Pro hands_on_hadoop2 % scp movie_new.csv rating_new.csv exouser@149.165.159.24:~/
exouser@149.165.159.24's password:
movie_new.csv                                                                 100% 1672KB   3.7MB/s   00:00
rating_new.csv                                                                100%   48MB   2.8MB/s   00:16
```

```
exouser@bda-hadoop2:~$ jps
247744 NameNode
132641 Bootstrap
248295 SecondaryNameNode
247975 DataNode
249495 Jps
248694 ResourceManager
248920 NodeManager
```

```
exouser@bda-hadoop2:~$ hadoop fs -mkdir -p /assignment_data
exouser@bda-hadoop2:~$ hadoop fs -ls /
Found 1 items
drwxr-xr-x   - exouser supergroup          0 2024-11-23 22:11 /assignment_data
exouser@bda-hadoop2:~$ hdfs dfs -put movie_new.csv /assignment_data
exouser@bda-hadoop2:~$ hdfs dfs -put rating_new.csv /assignment_data
exouser@bda-hadoop2:~$ hadoop fs -ls /assignment_data
Found 2 items
-rw-r--r--   1 exouser supergroup    1712108 2024-11-23 22:11 /assignment_data/movie_new.csv
-rw-r--r--   1 exouser supergroup   50404502 2024-11-23 22:12 /assignment_data/rating_new.csv
```

Task 1: Data Exploration

```
exouser@bda-hadoop2:~$ hadoop fs -cat /assignment_data/movie_new.csv | head -n 10
movieId,title,genres,language,runTime
1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy,English,83
2,Jumanji (1995),Adventure|Children|Fantasy,Mandarin,103
3,Grumpier Old Men (1995),Comedy|Romance,English,168
4,Waiting to Exhale (1995),Comedy|Drama|Romance,French,91
5,Father of the Bride Part II (1995),Comedy,French,158
6,Heat (1995),Action|Crime|Thriller,German,174
7,Sabrina (1995),Comedy|Romance,Mandarin,149
8,Tom and Huck (1995),Adventure|Children,German,157
9,Sudden Death (1995),Action,Spanish,132
```

```
exouser@bda-hadoop2:~$ hadoop fs -cat /assignment_data/rating_new.csv | head -n 10
userId,movieId,rating,timestamp,ageGroup,device,region
1,2,3.5,4/2/05 23:53,35-44,Tablet,Europe
1,29,3.5,4/2/05 23:31,35-44,Tablet,Europe
1,32,3.5,4/2/05 23:33,35-44,Tablet,Europe
1,47,3.5,4/2/05 23:32,35-44,Tablet,Europe
1,50,3.5,4/2/05 23:29,35-44,Tablet,Europe
1,112,3.5,9/10/04 3:09,35-44,Tablet,Europe
1,151,4,9/10/04 3:08,35-44,Tablet,Europe
1,223,4,4/2/05 23:46,35-44,Tablet,Europe
1,253,4,4/2/05 23:35,35-44,Tablet,Europe
```

Data Exploration Summary

Dataset 1: Movies

- **Structure**: This dataset contains 5 columns: movieId (integer), title (string), genres (string), language (string), and runTime (integer).
- **Purpose**: It provides metadata about movies, including their unique ID, title, associated genres, primary language, and runtime.

Dataset 2: Ratings

- **Structure**: This dataset contains 7 columns: userId (integer), movieId (integer), rating (float), timestamp (string), ageGroup (string), device (string), and region (string).
- **Purpose**: It records user ratings for movies, along with demographic information about users (e.g., ageGroup, region) and additional context such as the device used to rate and the time of the rating.

Key Relationship

Both datasets share the column movieId, which acts as a **primary key** in the movies dataset and a **foreign key** in the ratings dataset. This enables a join operation where the movies' metadata can be linked to their corresponding ratings, facilitating deeper analysis of user preferences and movie performance.

Task 2: MapReduce Join Operation

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


class MovieRatingJoin(MRJob):

    def mapper(self, _, line):
        # Skip the header lines
        if 'movieId' in line or 'userId' in line:
            return

        fields = line.strip().split(',')

        # Process movie dataset
        if len(fields) == 5:  # Movie dataset
            movie_id, title, genres, language, runtime = fields
            # Emit movie info with key as movie_id
            yield movie_id, ('MOVIE', title, genres, language, runtime)

        # Process rating dataset
        elif len(fields) == 7:  # Rating dataset
            user_id, movie_id, rating, timestamp, age_group, device, region = fields
            # Emit rating info with key as movie_id
            yield movie_id, ('RATING', user_id, rating, timestamp, age_group, device, region)

    def reducer(self, key, values):
        movie_data = None
        ratings = []

        # Separate movie and rating data
        for value in values:
            if value[0] == 'MOVIE':
                movie_data = value[1:]  # Movie information
            elif value[0] == 'RATING':
                ratings.append(value[1:])  # Ratings information

        # Emit joined data if movie_data exists
        if movie_data:
            for rating in ratings:
                yield key, movie_data + rating

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


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

```
exouser@bda-hadoop2:~/mrjobs$ python3 join_mrjob_v1.py -r hadoop hdfs:///assignment_data/movie_new.csv hdfs:///assignment_data/rating_ne
w.csv -o hdfs:///join_output/
No configs found; falling back on auto-configuration
No configs specified for hadoop runner
Looking for hadoop binary in /home/exouser/hadoop-3.4.0/bin...
Found hadoop binary: /home/exouser/hadoop-3.4.0/bin/hadoop
Using Hadoop version 3.4.0
Looking for Hadoop streaming jar in /home/exouser/hadoop-3.4.0...
Found Hadoop streaming jar: /home/exouser/hadoop-3.4.0/share/hadoop/tools/lib/hadoop-streaming-3.4.0.jar
Creating temp directory /tmp/join_mrjob_v1.exouser.20241125.165750.965161
uploading working dir files to hdfs:///user/exouser/tmp/mrjob/join_mrjob_v1.exouser.20241125.165750.965161/files/wd...
Copying other local files to hdfs:///user/exouser/tmp/mrjob/join_mrjob_v1.exouser.20241125.165750.965161/files/
Running step 1 of 1...
  packageJobJar: [/tmp/hadoop-unjar7363388847004294060/] [] /tmp/streamjob6269437408165060262.jar tmpDir=null
  Connecting to ResourceManager at /0.0.0.0:8032
  Connecting to ResourceManager at /0.0.0.0:8032
  Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/exouser/.staging/job_1732399787764_0057
  Total input files to process : 2
  number of splits:3
  Submitting tokens for job: job_1732399787764_0057
```

```
  Job job_1732399787764_0057 completed successfully
  Output directory: hdfs:///join_output/
```

```
exouser@bda-hadoop2:~/mrjobs$ hadoop fs -cat /join_output/part-00000 | head -n 3000
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "4005", "4", "3/28/06 19:47", "45-54", "TV", "South America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "4278", "4", "4/3/97 6:06", "25-34", "Mobile", "South America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "6171", "5", "8/31/14 19:04", "35-44", "Desktop", "Europe"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "6170", "5", "4/16/03 18:43", "18-24", "Mobile", "South America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "5031", "4.5", "9/21/07 0:29", "25-34", "Tablet", "Europe"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "4279", "4.5", "10/24/12 6:32", "18-24", "TV", "South America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "4137", "3.5", "11/11/05 15:22", "25-34", "Desktop", "Europe"]
```

```
"10"    ["GoldenEye (1995)", "Action|Adventure|Thriller", "Spanish", "156", "1806", "4", "6/28/07 2:39", "55+", "TV", "Europe"]
"10"    ["GoldenEye (1995)", "Action|Adventure|Thriller", "Spanish", "156", "430", "3", "11/7/06 9:27", "18-24", "Desktop", "Asia"]
"10"    ["GoldenEye (1995)", "Action|Adventure|Thriller", "Spanish", "156", "3412", "3", "6/13/96 17:51", "55+", "Desktop", "Asia"]
"10"    ["GoldenEye (1995)", "Action|Adventure|Thriller", "Spanish", "156", "2839", "3", "6/11/97 23:16", "45-54", "TV", "Africa"]
"10"    ["GoldenEye (1995)", "Action|Adventure|Thriller", "Spanish", "156", "159", "3", "12/16/96 17:52", "45-54", "Desktop", "North America"]
```

MapReduce Workflow for the Join Operation

This MapReduce job is designed to perform a reduce-side join between two datasets: movies and ratings. The datasets are joined using the movieId column, which serves as the key for both datasets. The workflow consists of the following steps:

1. Mapper Class Design

Functionality:

- The mapper processes both datasets and emits intermediate key-value pairs where the movieId is the key.
- The value includes a tag (MOVIE or RATING) to identify the origin of the record and the relevant details.

Input:

- movies dataset: Lines containing movie metadata (movieId, title, genres, language, runtime).
- ratings dataset: Lines containing rating data (userId, movieId, rating, timestamp, ageGroup, device, region).

Output:

- Key: movieId
- Value:
  - For movies: ('MOVIE', title, genres, language, runtime)
  - For ratings: ('RATING', userId, rating, timestamp, ageGroup, device, region)

2. Reducer Class Design

Functionality:

- The reducer groups all intermediate values by the movieId key.
- It separates movie metadata from ratings.
- If movie metadata exists for the given movieId, it performs an inner join with the ratings data.
- For each rating, it combines the movie metadata with the rating information.

Input:

- Key: movieId
- Values:
  - A list of tuples, either:
    - ('MOVIE', title, genres, language, runtime) or
    - ('RATING', userId, rating, timestamp, ageGroup, device, region)

Output:

- Key: movieId
- Value: A combined record of movie metadata and rating information.

Interaction Between Mapper and Reducer

1.Mapper Phase:

1. Each mapper processes a chunk of the input files (either movies or ratings).
2. It emits key-value pairs with movieId as the key and the tagged value as the payload.

2.Shuffle and Sort Phase:

1. The MapReduce framework groups all intermediate key-value pairs by the movieId key.
2. Values are sorted and sent to the corresponding reducer.

3.Reducer Phase:

1. The reducer processes all values grouped by movieId.
2. It separates movie metadata from ratings data and performs an inner join.
3. The joined data is emitted as the final output.

Task 3: Secondary Sorting

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


class MovieRatingSecondarySort(MRJob):
    SORT_VALUES = True  # Ensures that values are sorted during the shuffle phase

    def mapper(self, _, line):
        # Skip the header lines
        if 'movieId' in line or 'userId' in line:
            return

        fields = line.strip().split(',')

        # Process movie dataset
        if len(fields) == 5:  # Movie dataset
            movie_id, title, genres, language, runtime = fields
            # Emit movie info with a fixed priority of 0 to ensure it precedes ratings
            yield movie_id, ('0', 'MOVIE', title, genres, language, runtime)

        # Process rating dataset
        elif len(fields) == 7:  # Rating dataset
            user_id, movie_id, rating, timestamp, age_group, device, region = fields
            # Emit ratings with negative priority (to sort ratings in descending order by rating)
            yield movie_id, (rating, 'RATING', user_id, rating, timestamp, age_group, device, region)


#     def reducer(self, key, values):
#         # Yield each key-value pair exactly as received
#         for value in values:
#             yield key, value

    def reducer(self, key, values):
        movie_data = None
        ratings = []

        # Separate movie and rating data
        for value in values:
            if value[1] == 'MOVIE':
                movie_data = value[2:]  # Movie information
            elif value[1] == 'RATING':
                ratings.append(value[2:])  # Ratings information

        # Emit joined data if movie_data exists
        if movie_data:
            for rating in ratings:
                yield key, movie_data + rating

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


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

```
exouser@bda-hadoop2:~/mrjobs$ python3 conf_mrjob_v1.py -r hadoop hdfs:///assignment_data/movie_new.csv hdfs:///assignment_data/rating_new.csv -o hdfs:///conf_output/
No configs found; falling back on auto-configuration
No configs specified for hadoop runner
Looking for hadoop binary in /home/exouser/hadoop-3.4.0/bin...
Found hadoop binary: /home/exouser/hadoop-3.4.0/bin/hadoop
Using Hadoop version 3.4.0
Looking for Hadoop streaming jar in /home/exouser/hadoop-3.4.0...
Found Hadoop streaming jar: /home/exouser/hadoop-3.4.0/share/hadoop/tools/lib/hadoop-streaming-3.4.0.jar
Creating temp directory /tmp/conf_mrjob_v1.exouser.20241125.173452.869816
uploading working dir files to hdfs:///user/exouser/tmp/mrjob/conf_mrjob_v1.exouser.20241125.173452.869816/files/wd...
Copying other local files to hdfs:///user/exouser/tmp/mrjob/conf_mrjob_v1.exouser.20241125.173452.869816/files/
Running step 1 of 1...
  packageJobJar: [/tmp/hadoop-unjar5240527260795664027/] [] /tmp/streamjob747054420746753385.jar tmpDir=null
  Connecting to ResourceManager at /0.0.0.0:8032
  Connecting to ResourceManager at /0.0.0.0:8032
  Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/exouser/.staging/job_1732399787764_0059
  Total input files to process : 2
  number of splits:3
  Submitting tokens for job: job_1732399787764_0059
```

```
  Job job_1732399787764_0059 completed successfully
  Output directory: hdfs:///conf_output/
```

```
exouser@bda-hadoop2:~$ hadoop fs -cat /conf_output/part-00000 | head -n 1000
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "2056", "0.5", "3/22/05 23:32", "18-24", "Desktop", "Europe"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "2084", "0.5", "1/23/10 8:05", "55+", "Mobile", "Africa"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "2202", "0.5", "1/3/07 10:09", "25-34", "Desktop", "South America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "2362", "0.5", "2/29/04 0:10", "35-44", "Desktop", "North America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "4869", "0.5", "1/15/07 8:40", "18-24", "TV", "Europe"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "5256", "0.5", "5/22/04 2:20", "25-34", "TV", "North America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "5444", "0.5", "5/17/06 16:08", "45-54", "Mobile", "North America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "6734", "0.5", "2/24/05 14:15", "45-54", "Tablet", "South America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "1247", "1", "6/11/97 14:48", "25-34", "Desktop", "North America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "1662", "1", "11/5/08 23:34", "35-44", "Tablet", "Europe"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "2281", "1", "3/24/00 15:45", "25-34", "Mobile", "South America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "2361", "1", "7/29/07 8:50", "55+", "Tablet", "South America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "269", "1", "11/13/97 15:12", "55+", "TV", "Asia"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "3052", "1", "7/15/97 18:48", "55+", "Mobile", "South America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "3277", "1", "8/15/07 17:27", "18-24", "Desktop", "Europe"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "3300", "1", "7/4/00 3:37", "35-44", "TV", "Africa"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "3450", "1", "11/27/03 23:47", "45-54", "Desktop", "South America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "3596", "1", "11/21/10 4:00", "35-44", "TV", "Africa"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "4217", "1", "1/22/10 6:25", "45-54", "Tablet", "Asia"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "47", "1", "4/17/09 3:53", "18-24", "Tablet", "South America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "514", "1", "12/4/99 22:35", "55+", "Desktop", "Africa"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "5347", "1", "9/1/12 19:37", "18-24", "TV", "Asia"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "5625", "1", "7/4/97 8:49", "18-24", "Tablet", "North America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "6040", "1", "6/21/97 11:24", "35-44", "TV", "South America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "6066", "1", "10/3/01 16:17", "55+", "Desktop", "Europe"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "6514", "1", "10/16/07 20:56", "18-24", "Desktop", "Europe"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "6561", "1", "11/15/08 6:02", "35-44", "Tablet", "Asia"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "763", "1", "2/28/07 19:23", "18-24", "TV", "North America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "115", "1.5", "11/9/10 9:19", "45-54", "Mobile", "Asia"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "1374", "1.5", "4/3/06 23:59", "25-34", "Desktop", "Europe"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "1377", "1.5", "3/22/05 11:44", "18-24", "TV", "Asia"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "1616", "1.5", "1/7/09 12:55", "55+", "Tablet", "South America"]
"1"     ["Toy Story (1995)", "Adventure|Animation|Children|Comedy|Fantasy", "English", "83", "2004", "1.5", "1/26/09 0:32", "55+", "Mobile", "Asia"]
```

Explanation of Adjustments for Secondary Sorting

Mapper Adjustments

Purpose:

•Emit records with a custom key-value pair to facilitate sorting during the shuffle phase.

Implementation (using SORT_VALUES = True for secondary sorting by value):

•For movies:

- Emit a fixed priority of 0 to ensure movie data is processed first.
- yield movie_id, ('0', 'MOVIE', title, genres, language, runtime)

•For ratings:

- Emit the rating (to sort ratings in ascending order).
- yield movie_id, (rating, 'RATING', user_id, rating, timestamp, age_group, device, region)

•Shuffle and Sort Phase :

•Data is grouped by movie_id and sorted in ascending order based on the rating value before being sent to the reducer.

Reducer Adjustments

Purpose:

•Process data in a pre-sorted manner (movies first, followed by ascending ratings).

•Join movie metadata with ratings data.

Implementation:

•Separate movie and rating data:

- Identify records with the 'MOVIE' tag and extract movie information.
- Collect records with the 'RATING' tag into a list.

•Emit joined results:

- For each rating, append the movie metadata.
- For rating in ratings: yield key, movie_data + rating

Key Benefits

•Ensures movie metadata is processed first for each movieId.

•Simplifies the reducer logic by relying on sorted data from the shuffle phase.

•Outputs ratings in ascending order for each movie, enabling better data organization.

Task 4: Custom Partitioner

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


class MovieRatingSecondarySortPartitioned(MRJob):
    SORT_VALUES = True  # Ensures that values are sorted during the shuffle phase

    def mapper(self, _, line):
        # Skip the header lines
        if 'movieId' in line or 'userId' in line:
            return

        fields = line.strip().split(',')

        # Process movie dataset
        if len(fields) == 5:  # Movie dataset
            movie_id, title, genres, language, runtime = fields
            # Emit movie info with a fixed priority of 0 to ensure it precedes ratings
            yield movie_id, ('0', 'MOVIE', title, genres, language, runtime)

        # Process rating dataset
        elif len(fields) == 7:  # Rating dataset
            user_id, movie_id, rating, timestamp, age_group, device, region = fields
            # Emit ratings with negative priority (to sort ratings in descending order by rating)
            yield movie_id, (rating, 'RATING', user_id, rating, timestamp, age_group, device, region)

    def reducer_join(self, key, values):
        movie_data = None
        ratings = []

        # Separate movie and rating data
        for value in values:
            if value[1] == 'MOVIE':
                movie_data = value[2:]  # Movie information
            elif value[1] == 'RATING':
                ratings.append(value[2:])  # Ratings information

        # Emit joined data if movie_data exists
        if movie_data:
            for rating in ratings:
                # Emit the device as the new key for partitioning
                yield rating[5], movie_data + rating  # Using region as the key

    def reducer_partitioned(self, key, values):
        # Simply yield all the data grouped by the partition key (e.g., device)
        for value in values:
            yield key, value

    def steps(self):
        return [
            MRStep(
                mapper=self.mapper,
                reducer=self.reducer_join,
            ),
            MRStep(
                reducer=self.reducer_partitioned,
                jobconf={
                    'mapreduce.job.reduces': 5,
                    'mapreduce.partition.keypartitioner.options': '-k1,1'
                },
            ),
        ]


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

```
exouser@bda-hadoop2:~/mrjobs$ python3 part_mrjob_v1.py -r hadoop hdfs:///assignment_data/movie_new.csv hdfs:///assignment_data/rating_new.csv -o hdfs:///part_output/
No configs found; falling back on auto-configuration
No configs specified for hadoop runner
Looking for hadoop binary in /home/exouser/hadoop-3.4.0/bin...
Found hadoop binary: /home/exouser/hadoop-3.4.0/bin/hadoop
Using Hadoop version 3.4.0
Looking for Hadoop streaming jar in /home/exouser/hadoop-3.4.0...
Found Hadoop streaming jar: /home/exouser/hadoop-3.4.0/share/hadoop/tools/lib/hadoop-streaming-3.4.0.jar
Creating temp directory /tmp/part_mrjob_v1.exouser.20241125.223356.245804
uploading working dir files to hdfs:///user/exouser/tmp/mrjob/part_mrjob_v1.exouser.20241125.223356.245804/files/wd...
Copying other local files to hdfs:///user/exouser/tmp/mrjob/part_mrjob_v1.exouser.20241125.223356.245804/files/
Running step 1 of 2...
  packageJobJar: [/tmp/hadoop-unjar49298437601431 3974/] [] /tmp/streamjob6575596608826556273.jar tmpDir=null
  Connecting to ResourceManager at /0.0.0.0:8032
  Connecting to ResourceManager at /0.0.0.0:8032
  Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/exouser/.staging/job_1732399787764_0086
  Total input files to process : 2
  number of splits:3
  Submitting tokens for job: job_1732399787764_0086
```

Both the chained jobs are successfully completed

```
  Job job_1732399787764_0086 completed successfully
  Output directory: hdfs:///user/exouser/tmp/mrjob/part_mrjob_v1.exouser.20241125.223356.245804/step-output/0000
```

```
  Job job_1732399787764_0087 completed successfully
  Output directory: hdfs:///part_output/
```

Five reducer outputs

```
exouser@bda-hadoop2:~/mrjobs$ hadoop fs -ls /part_output | awk '{print $NF}'
items
/part_output/_SUCCESS
/part_output/part-00000
/part_output/part-00001
/part_output/part-00002
/part_output/part-00003
/part_output/part-00004
```

First Three Rows of Part File 0 (Containing Only Values from the Africa Region)

```
exouser@bda-hadoop2:~/mrjobs$ hadoop fs -cat /part_output/part-00001 | head -n 3
"Africa"        ["$5 a Day (2008)", "Comedy|Drama", "German", "126", "2348", "3", "9/20/10 22:35", "45-54", "Mobile", "Africa"]
"Africa"        ["$5 a Day (2008)", "Comedy|Drama", "German", "126", "4335", "3", "1/6/11 23:53", "55+", "Tablet", "Africa"]
"Africa"        ["$5 a Day (2008)", "Comedy|Drama", "German", "126", "4415", "3", "12/25/10 13:10", "45-54", "TV", "Africa"]
```

First Three Rows of Part File 1 (Containing Only Values from the Asia Region)

```
exouser@bda-hadoop2:~/mrjobs$ hadoop fs -cat /part_output/part-00002 | head -n 3
"Europe"        ["$5 a Day (2008)", "Comedy|Drama", "German", "126", "5129", "4", "9/20/10 3:02", "25-34", "TV", "Europe"]
"Europe"        ["'71 (2014)", "Action|Drama|Thriller|War", "English", "160", "3858", "3", "12/20/14 2:00", "18-24", "Tablet", "Europe"]
"Europe"        ["'Hellboy': The Seeds of Creation (2004)", "Action|Adventure|Comedy|Documentary|Fantasy", "Spanish", "154", "540", "2.5", "2/5/13 21:28", "55+", "TV", "Euro
pe"]
```

Similar outputs were observed for the remaining part files.

Partitioning

The code uses **region** as the key for partitioning data across reducers. In the reducer_join step, the region field from the rating dataset is emitted as the key, ensuring all records for the same region are grouped together during partitioning. The default Hadoop partitioner then distributes these keys across reducers using a hash function:

Reducer = (hashCode(region) mod numReducers)

The mapreduce.job.reduces property is set to 5, creating 5 reducers to handle the data.

Efficiency

Partitioning by region provides:

1.Balanced Workload: If regions are evenly distributed, which is the case in this dataset, reducers process similar amounts of data, optimizing load.

2.Localized Processing: Data for each region is handled independently, simplifying analysis.

3.Scalability: Supports scaling by increasing reducers for larger datasets or more regions.

However, if data is skewed (e.g., one region dominates), some reducers may be overloaded, requiring preprocessing or a custom partitioner for better balance.

Explanation of Custom Partitioner Design

•**Key Selection:** region is chosen as it provides a logical grouping for geographical analysis.

•**Partitioner Use:** The default partitioner is leveraged, hashing region to evenly distribute keys across reducers.

•**Steps:**

- Emit region as the key in reducer_join.
- Use mapreduce.job.reduces to configure 5 reducers.
- Validate even distribution across reducers.

Conclusion

The partitioning strategy by region aligns with the requirements by distributing data logically and efficiently. While effective for balanced datasets, further customization might be needed to handle skewed distributions.

Task 5: Testing

```python
import unittest
from join_mrjob_v1 import MovieRatingJoin  # Import the MRJob class from the correct file


class TestMovieRatingJoin(unittest.TestCase):

    def setUp(self):
        """Set up the MRJob instance."""
        self.mr_job = MovieRatingJoin()

    def test_mapper_movies(self):
        """Test the mapper for the movies dataset."""
        input_line = "1,Toy Story,Adventure|Comedy,English,83"
        expected_output = [
            ("1", ("MOVIE", "Toy Story", "Adventure|Comedy", "English", "83"))
        ]
        mapper_output = list(self.mr_job.mapper(None, input_line))
        self.assertEqual(mapper_output, expected_output)

    def test_mapper_ratings(self):
        """Test the mapper for the ratings dataset."""
        input_line = "1,1,4.5,2024-11-23,18-24,Tablet,Europe"
        expected_output = [
            ("1", ("RATING", "1", "4.5", "2024-11-23", "18-24", "Tablet", "Europe"))
        ]
        mapper_output = list(self.mr_job.mapper(None, input_line))
        self.assertEqual(mapper_output, expected_output)

    def test_reducer_join(self):
        """Test the reducer for joining movie and rating data."""
        key = "1"
        values = [
            ("MOVIE", "Toy Story", "Adventure|Comedy", "English", "83"),
            ("RATING", "1", "4.5", "2024-11-23", "18-24", "Tablet", "Europe"),
            ("RATING", "2", "3.5", "2024-11-22", "25-34", "Phone", "Asia"),
        ]
        expected_output = [
            ("1", ("Toy Story", "Adventure|Comedy", "English", "83", "1", "4.5", "2024-11-23", "18-24", "Tablet", "Europe")),
            ("1", ("Toy Story", "Adventure|Comedy", "English", "83", "2", "3.5", "2024-11-22", "25-34", "Phone", "Asia")),
        ]
        reducer_output = list(self.mr_job.reducer(key, iter(values)))
        self.assertEqual(reducer_output, expected_output)

    def test_reducer_no_movie(self):
        """Test the reducer when no movie data is present."""
        key = "1"
        values = [
            ("RATING", "1", "4.5", "2024-11-23", "18-24", "Tablet", "Europe"),
        ]
        expected_output = []  # No movie data means no output
        reducer_output = list(self.mr_job.reducer(key, iter(values)))
        self.assertEqual(reducer_output, expected_output)

    def test_reducer_no_ratings(self):
        """Test the reducer when no ratings data is present."""
        key = "1"
        values = [
            ("MOVIE", "Toy Story", "Adventure|Comedy", "English", "83"),
        ]
        expected_output = []  # Movie without ratings means no join output
        reducer_output = list(self.mr_job.reducer(key, iter(values)))
        self.assertEqual(reducer_output, expected_output)


if __name__ == "__main__":
    unittest.main()
```

```
exouser@bda-hadoop2:~/mrjobs$ python3 test_cases.py
.....
----------------------------------------------------------------------
Ran 5 tests in 0.017s

OK
```

Summary of Testing

1. Unit Testing

Unit tests were implemented for the MovieRatingJoin MapReduce job to validate the correctness of the inner join operation. The test cases included:

- **Mapper Tests:**
  - Tested for both movies and ratings datasets to ensure proper key-value pairs are emitted.
- **Reducer Tests:**
  - **Successful Join:** Validated the correctness of the join operation by matching movie data with multiple ratings.
  - **No Movie Data:** Confirmed no output is emitted when no movie data is present for a key.
  - **No Ratings Data:** Confirmed no output is emitted when no ratings data exists for a key.

2. Inner Join Correctness

- The reducer ensures an **inner join** by only emitting output when both movie data and ratings data exist for a given movieId. Keys with missing data in either dataset are excluded from the output.

3. Results and Issues

- **Results:** All tests passed without errors, confirming the implementation's correctness.
- **Issues:** No bugs or significant issues were encountered during testing. The logic handled all edge cases as expected.

Bonus: Explain, in your own words, why Hadoop MapReduce is suitable for handling large-scale data processing. Discuss the advantages and potential challenges of using Hadoop for real-world big data applications, with considerations around scalability, fault tolerance, and data distribution. Minimum one page (font size 12, 1.5 spacing)

Hadoop MapReduce's distributed and parallel computing capabilities make it ideal for handling large volumes of data. It is perfect for effectively analyzing massive datasets because it breaks up large computing jobs into smaller pieces, performs them concurrently over numerous nodes, and aggregates the results.

Advantages

1.Scalability:
Hadoop can easily manage increasing data volumes because of its capacity to extend horizontally by adding more nodes to the cluster.
Example: A global e-commerce platform processes petabytes of transaction logs to identify seasonal trends. As the volume of data increases, Hadoop can scale horizontally by adding new nodes to the cluster to handle the workload.

2.Fault Tolerance:
Hadoop replicates data across nodes and is integrated with HDFS. Reliable execution is ensured by reassigning work to other nodes in the event of a node failure.
Example: While analyzing clickstream data, a node in the Hadoop cluster crashes. The job continues uninterrupted because Hadoop replicates the data across nodes, and the work is reassigned to another node.

3.Data Locality:
By processing tasks on the nodes that house the data, performance is enhanced and data transport expenses are decreased.
Example: A telecom company processes call detail records (CDRs) for billing purposes. By running the analysis on the nodes where the data is stored, Hadoop reduces the need to transfer large datasets over the network, speeding up the process.

4.Parallelism:
Data processing is accelerated when tasks are carried out concurrently on several machines.
Example: A weather forecasting organization runs simulations on terabytes of sensor data collected from satellites. Tasks are divided across multiple nodes, significantly reducing the overall computation time.

5.Cost-Effectiveness:
When compared to older systems, using commodity hardware lowers infrastructure expenses.
Example: A startup chooses Hadoop to process log files on commodity hardware instead of investing in expensive proprietary solutions, keeping costs low while handling large data volumes.

Challenges

1.Complexity of Development:

Creating MapReducing employment requires knowledge of distributed systems and programming.

Example: A data engineer without prior experience in distributed systems finds it difficult to write MapReduce jobs to analyze sales data, as it requires understanding of both programming and the Hadoop ecosystem.

2.High Latency:

Hadoop is not appropriate for real-time analytics since batch processing entails considerable latency.

Example: An ad-tech company wants to monitor user engagement in real-time to optimize ad placements. Hadoop's batch processing model is too slow for such real-time requirements, making it unsuitable for this task.

3.Limitations of Iterative Processing:

Because HDFS reads and writes occur often, tasks like machine learning that need to retrieve datasets repeatedly have a significant overhead.

Example: Training a machine learning model on large datasets using iterative algorithms (e.g., gradient descent) is inefficient because Hadoop writes data to HDFS after each iteration, significantly slowing down the process.

4.Resource Management:

Inefficient use of resources might result from misconfigured systems or uneven data distribution.

Example: A Hadoop cluster is misconfigured, causing one node to handle a disproportionately large chunk of data. This leads to inefficiencies, with some nodes idle while others are overloaded.

5.Storage Overhead:

Replication of data for fault tolerance raises storage needs, which affects cost effectiveness.

Example: A financial institution processes sensitive data and uses Hadoop's fault-tolerance mechanism with a default replication factor of 3. This triples the storage requirement, increasing hardware costs.

In conclusion

Hadoop MapReduce is a dependable and scalable method for batch-oriented, large-scale data processing. For use cases needing real-time or interactive processing, however, supplementary tools like Apache Spark are necessary because of its high latency and inefficiency for iterative operations. Hadoop is still a key technology for extracting knowledge from enormous datasets, despite its drawbacks.