Massive Data Analysis
Term Project Report

Introduction

- Online summarization of movie reviews and scores using DGIM algorithm
- User can query the most popular or highest ranked movies in a specific time
- User can query information of a specific movie

Data Preprocess

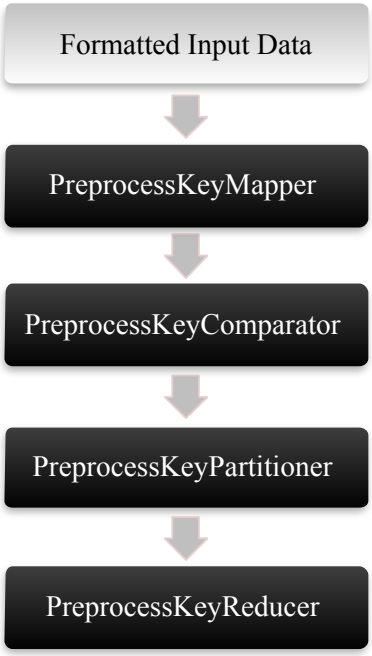
- When executing DGIM algorithm, data should come from the earliest to the latest as stream., so it's necessary to sort the reviews by timestamp and extract the information we need, such as timestamp, productId, userId, and score of each movie review. We employ movie review dataset from https://snap.stanford.edu/data/web-Movies.html, and part of the dataset is shown as below.

```
product/productId: B003AI2VGA
review/userId: A141HP4LYPWMSR
review/profileName: Brian E. Erland "Rainbow Sphinx"
review/helpfulness: 7/7
review/score: 3.0
review/time: 1182729600
review/summary: "There Is So Much Darkness Now ~ Come For The Miracle"
review/text: Synopsis: On the daily trek from Juarez, Mexico to El Paso, Texas an ever increasing
number of female workers are found raped and murdered in the surrounding desert. Investigative
reporter Karina Danes (Minnie Driver) arrives from Los Angeles to pursue the story and angers both
the local police and the factory owners who employee the undocumented aliens with her pointed
questions and relentless quest for the truth.<br /><br />Her story goes nationwide when a young
girl named Mariela (Ana Claudia Talancon) survives a vicious attack and walks out of the desert
crediting the Blessed Virgin for her rescue. Her story is further enhanced when the "Wounds of
Christ" (stigmata) appear in her palms. She also claims to have received a message of hope for the
Virgin Mary and soon a fanatical movement forms around her to fight against the evil that holds
such a stranglehold on the area.<br /><br />Critique: Possessing a lifelong fascination with such
esoteric matters as Catholic mysticism, miracles and the mysterious appearance of the stigmata, I
was immediately attracted to the '05 DVD release `Virgin of Juarez'. The film offers a rather
unique storyline blending current socio-political concerns, the constant flow of Mexican migrant
workers back and forth across the U.S./Mexican border and the traditional Catholic beliefs of the
Hispanic population. I must say I was quite surprised by the unexpected route taken by the plot and
the means and methods by which the heavenly message unfolds.<br /><br />`Virgin of Juarez' is not a
film that you would care to watch over and over again, but it was interesting enough to merit at
least one viewing. Minnie Driver delivers a solid performance and Ana Claudia Talancon is perfect
as the fragile and innocent visionary Mariela. Also starring Esai Morales and Angus Macfadyen (
Braveheart).
```

- After being preprocessed, we expect the data to be compressed as follows:
  1. The first attribute is the timestamp of the movie review
  2. The second attribute is the concatenation of {productId}_{userId}_{score}

```
872035200   6302967538_A37I5QIHD9UMPD_5.0
872035200   B00004CILW_A37I5QIHD9UMPD_5.0
872035200   6302763770_A37I5QIHD9UMPD_5.0
872035200   B00008V6YR_A37I5QIHD9UMPD_5.0
872294400   6302049040_A2XBTS97FERY2Q_5.0
872294400   B004J1A72C_A2XBTS97FERY2Q_5.0
872294400   B004J1A6WS_A2XBTS97FERY2Q_5.0
872294400   B0001Z4P2I_A2XBTS97FERY2Q_5.0
872467200   630549519X_A1QWWL8ZAUOOX4_5.0
872467200   B000089795_A1QWWL8ZAUOOX4_5.0
876700800   B000MQTI6O_A2JVVS49Y183CZ_5.0
876700800   6300271765_A2JVVS49Y183CZ_5.0
876700800   B003ODIUZ2_A2JVVS49Y183CZ_5.0
876700800   B000MQTI6Y_A2JVVS49Y183CZ_5.0
878860800   6303542050_A1DSDR71XQCI18_1.0
879206400   0792844882_A1SH28X6HM8BXB_4.0
879206400   B004TJ1GVK_A1SH28X6HM8BXB_4.0
879206400   B00020X88O_A1SH28X6HM8BXB_4.0
```

• Architecture of preprocessing MapReduce is shown as below.



1. Before Preprocessing
   Since Mapper reads one line at a time by default, out data would be scattered and unstructured because a movie has 9 lines of information. We attempt to set Mapper to read 9 lines at a time configuring NLineInputFormat(), but failed due to unknown reason. We tried another way: we set Mapper to read lines according to specific delimiter(\n\n). Unexpectedly, we found out that the dataset is inconsistent. The third attribute of several data contains '\n\n' characters, which disturb the input of Mapper.

   Therefore, we are forced to do formatting in advance. We wrote a section of code to eliminate all '\n' characters, then split the data by "productId" and again insert '\n\n' between the split spots to separate information of each movie.

2. PreprocessMapper
   Mapper converts each review into key-value pair where key is timestamp and value is the concatenation of information we need.

3. PreprocessKeyComparator
   Sort each key-value pair received form Mappers by timestamp.

4. PreprocessPartitioner
   Divide the total time span into several intervals, and assign key-value pair to Reducer by its timestamp.

5. PreprocessReducer
   Output the key-value pairs.

Algorithm Design

- We define a Bucket class holding timestamp, size, and sum of score
- We use a hashmap to store windows of all movies, where key is the movie ID and value is a linked list representing the window of buckets of such movie
- For finding the most popular movies, we count the number of reviews of each movie within the query time interval. This is similar to counting 1's DGIM algorithm
- For calculating the average scores, we record the partial sum of scores of each movie within the time interval, and then divide the sum by the review count of such movie to obtain average score. This is similar to the partial sum extension of DGIM algorithm

- Counting Number of Review (DGIM)
  1. initialize(Date time) : initialize window and simulate data streaming using the preprocessed data as new incoming reviews

initialize(Date currentTime)

```
for each file F {
      for each line L in F {
            extract reviewTime, productId from L
            if reviewTime exceeds currentTime
                  break
            addReview(reviewTime, productId)
      }
}
```

  2. addReview(Date time, String productId) : add a new bucket of size 1 to the window of such movie

addReview(Date reviewTime, String productId)

```
bucket = new Bucket(reviewTime,1)
if window of productId exists
      extract window of productId
else
      create new window for productId
window.addFirst(bucket)
if difference of reviewTime and the timestamp of last bucket in window
      remove the last bucket in window
if needMerge(window.iterator)
      merge(productId, window.iterator)
```

3. needMerge(Iterator<Bucket> iter) : check whether buckets in the window need merging when adding a new review to a movie.

needMerge(Iterator<Bucket> iter)

```
get the first 3 buckets in window
if first.size() != second.size()
        no need to merge
else if second.size() == third.size()
        need to merge
```

4. merge(String productId, Iterator<Bucket> iter) : do merging if needed

merge(String productId, Iterator<Bucket> iter)

```
get the second and third buckets in window
add the size of third to second
remove third bucket
if needMerge(window.iterator)
        merge(productId, window.iterator)
```

5. query(Date time, String productId, long k) : get counts according to query and return the counts

6. getCount(String productId, long k) : for the queried movie, extract its window of buckets and add all sizes of buckets within the time interval. For the last bucket, deduct half of its size and return the final count.

getCount(String productId, long k)

```
for each bucket in window of productId {
        if currentTime-bucket.getTime() < k{
                size = bucket.size()
                count += size
        }
        else
                break
}
return count - size/2
```

7. getTrueCount(String productId, long k) : for the queried movie, add all sizes of data in the files within the time interval

- Calculating Average Score (DGIMAvg)
  1. initialize(Date time) : initialize window and simulate data streaming using the preprocessed data as new incoming reviews with scores

  2. addReview(Date time, String productId) : add a new bucket with score to the window of such movie

  3. needMerge(Iterator<Bucket> iter) : check whether buckets in the window need merging when adding a new review to a movie

  4. merge(String productId, Iterator<Bucket> iter) : do merging according to different conditions if needed

merge(String productId, Iterator<Bucket> iter)

```
get the second and third buckets in window
if second.size <=1
        third.size++
else{
        sum = second.sum + third.sum
        bound = pow(2, second.size+1)
        if sum <= bound {
                third.size++
                third.sum = sum
                remove second
        }else
                third.size++
}
```

  5. query(Date time, String productId, long k, ArrayList<Integer> counts) : get scores according to query and calculate the average scores using counts from DGIM

  6. getScore(String productId, long k) : for the queried movie, extract its window of buckets and add all scores of buckets within the time interval. For the last bucket, deduct half of its score and return the final score

  7. getTrueScore(String productId, long k) : for the queried movie, add all scores of data in the files within the time interval

Execution

- compile

```
$ sh compile.sh
———————————
rm -r class/*
javac -classpath hadoop-core-1.2.1.jar -d class preprocess/* dgim/*
jar -cvf TermProject.jar -C class/ .
```

- preprocess

```
$ sh execute.sh
————————————
hadoop dfs -rmr output
hadoop jar TermProject.jar preprocess.Preprocess input/movies_re.txt output
```

- execute DGIM query

```
$ sh DGIM.sh
————————————
hadoop jar TermProject.jar dgim.Main
```

- query example

enter a date since 1997/8/1 →

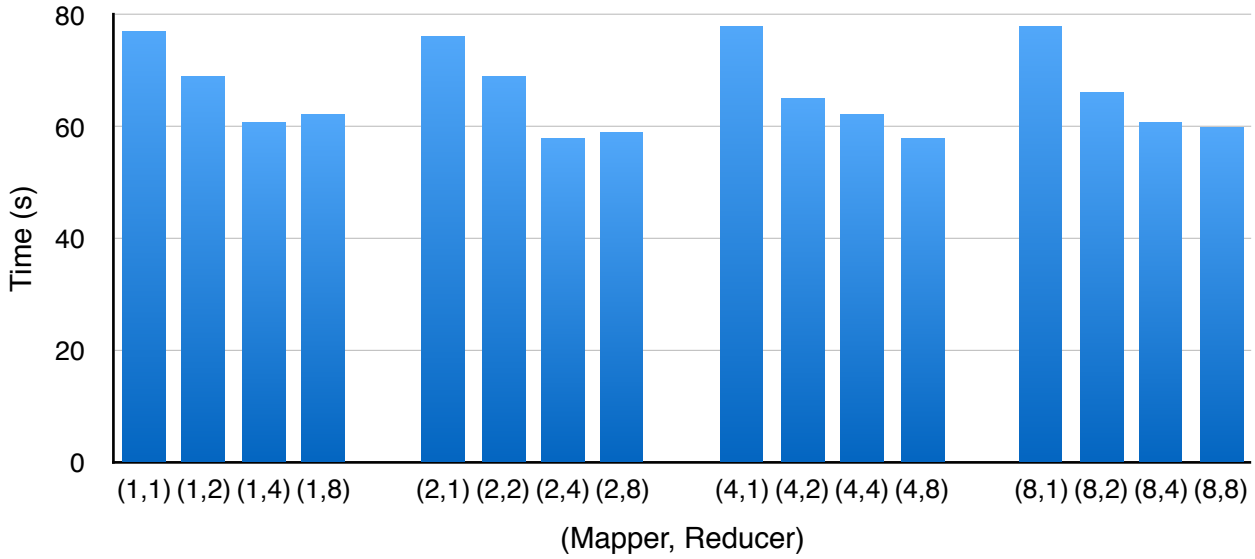
enter day to get top 10 list (k < 720) →

enter productId and day to query specific movie (k < 720) →

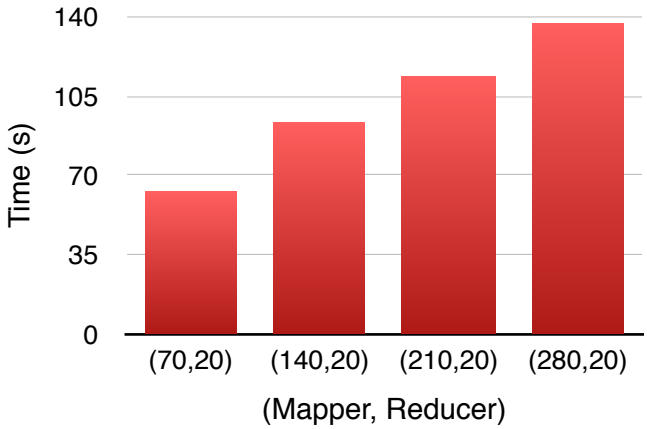
```
Specify current time: (1999/2/9 8)1999/2/9 8
Load input to Tue Feb 09 08:00:00 CST 1999
Enter k (day): (500) or productId and k (day): (6302967538 500)
100
Top 10 movies by avg. score:
rank0: 0767805739(7.5)
rank1: 0767805763(7.5)
rank2: 078062162X(7.5)
rank3: 0780622251(7.5)
rank4: 0780623967(7.5)
rank5: 0782009123(7.5)
rank6: 0967418518(7.5)
rank7: 156501345X(7.5)
rank8: 6300181553(7.5)
rank9: 6300247236(7.5)
Top 10 movies by the number of reviews
rank0: 0767802659(7.0)
rank1: 0767811038(7.0)
rank2: 6304913176(7.0)
rank3: B00004RF0E(7.0)
rank4: B0000648WZ(7.0)
rank5: B00008EY5W(7.0)
rank6: B0009MWEN0(7.0)
rank7: B000MF4O82(7.0)
rank8: B000OVLBHG(7.0)
rank9: B000UAFDP2(7.0)
Enter k (day): (500) or productId and k (day): (6302967538 500)
6302967538 100
DGIM/ approximate:1 true: 1
Error: 0.0
DGIMAvg/ approximate:7.5 true: 5.0
Enter k (day): (500) or productId and k (day): (6302967538 500)
```

Experiment Results

• Time v.s Number of Mapper and Reducer



- Observation :
  1. When the number of Reducer increases, the execution time decreases.
  2. Number of Mapper doesn't affect the performance of preprocessing. In fact, the actual number of Mapper depends on data locality and it's automatically configured. In the case of our dataset, data-local map tasks are fixed to 70, so if we set 10 map tasks, it will still launch 70 map tasks. If we set more map tasks, say 140, there will be 68 rack-local map tasks and 72 data-local map tasks. When number of map tasks exceeds 68, additional communication between racks are required, which actually slows the whole process down. Such result is shown as below.



• Time v.s Size of dataset
  - Since we only use MapReduce in preprocessing, where dataset with larger size trivially requires more time processing, so we omit this experiment.

Verification of DGIM Error Bound

```
Specify current time: (1999/2/9 8)2005/10/10 10
Load input to Mon Oct 10 10:00:00 CST 2005
Enter k (day): (500) or productId and k (day): (6302967538 500)
6302967538 500
DGIM/ approximate:2 true: 2
Error: 0.0
DGIMAvg/ approximate:3.25 true: 4.5
```

```
Specify current time: (1999/2/9 8)2008/10/10 10
Load input to Fri Oct 10 10:00:00 CST 2008
Enter k (day): (500) or productId and k (day): (6302967538 500)
5556167281 350
DGIM/ approximate:20 true: 23
Error: 0.13043478260869565
DGIMAvg/ approximate:5.6 true: 4.869565217391305
```

```
Specify current time: (1999/2/9 8)2008/10/10 10
Load input to Fri Oct 10 10:00:00 CST 2008
Enter k (day): (500) or productId and k (day): (6302967538 500)
5556167281 700
DGIM/ approximate:56 true: 52
Error: 0.07692307692307693
DGIMAvg/ approximate:5.214285714285714 true: 4.9423076923076925
```

```
Specify current time: (1999/2/9 8)2010/10/10 10
Load input to Sun Oct 10 10:00:00 CST 2010
Enter k (day): (500) or productId and k (day): (6302967538 500)
5556167281 350
DGIM/ approximate:26 true: 29
Error: 0.10344827586206896
DGIMAvg/ approximate:5.538461538461538 true: 4.758620689655173
```

```
Specify current time: (1999/2/9 8)2010/10/10 10
Load input to Sun Oct 10 10:00:00 CST 2010
Enter k (day): (500) or productId and k (day): (6302967538 500)
5556167281 700
DGIM/ approximate:62 true: 69
Error: 0.10144927536231885
DGIMAvg/ approximate:4.258064516129032 true: 4.782608695652174
```