ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

"SISTEMA DE RECOMENDACIÓN DE PELÍCULAS"

INFORME DE MATERIA DE GRADUACIÓN

Previo a la obtención del título de:

INGENIERO EN COMPUTACIÓN
ESPECIALIZACIÓN SISTEMAS DE INFORMACIÓN
INGENIERO EN COMPUTACIÓN
ESPECIALIZACIÓN SISTEMAS TECNOLÓGICOS

PRESENTADO POR:

ANA VICTORIA KAM ORTIZ
LIGIA ELENA CALVA PAUCAR

GUAYAQUIL - ECUADOR

2009

AGRADECIMIENTO

A Dios por su infinita bondad.

A nuestras familias por su paciencia y constante apoyo.

A Ing. Cristina Abad por su esmerado esfuerzo al guiarnos en esta materia de graduación.

A nuestros compañeros de clase por su desinteresada y valiosa colaboración.

DEDICATORIA

A Dios, a nuestros padres, hermanos, familiares, profesores, amigos y compañeros.

TRIBUNAL DE GRADUACIÓN

Ing. Cristina Abad Robalino

DIRECTOR DE TESIS

Ing. Mónica Villavicencio Cabezas

PROFESOR DELEGADO DEL DECANO

DECLARACIÓN EXPRESA

“La responsabilidad del contenido de este Proyecto de Graduación, nos corresponde exclusivamente; y el patrimonio intelectual del mismo a la **ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**”.

(Reglamento de Graduación de la ESPOL).

Ana Victoria Kam Ortiz

Ligia Elena Calva Paucar

RESUMEN

Este proyecto describe un mecanismo escalable y distribuido de generar recomendaciones de ítems para cualquier usuario. Aunque el enfoque principal es para recomendaciones de películas, esta solución puede ser aplicada a recomendaciones de otros ítems.

Se describen todas las herramientas utilizadas, en especial las proporcionadas por los Amazon Web Services (AWS) [1], para la manipulación de una cantidad masiva de datos, pruebas y ejecución final de este proyecto.

Además se incluye un ejemplo donde se mide el nivel de exactitud de las recomendaciones generadas.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Dado el constante crecimiento de Internet y la sobrecarga de información que esto conlleva, la tarea de seleccionar aquello que realmente necesitan los usuarios, dentro de una cantidad masiva de datos, resulta un tanto difícil y poco precisa. En consecuencia, nacen los Sistemas de Recomendación para facilitar la selección de información de manera rápida [2].

Actualmente los Sistemas de Recomendación son utilizados para diversos objetivos, entre ellos: filtrado de noticias, búsqueda de personas afines en comunidades y recomendaciones de libros, películas y demás en tiendas en-línea.

El objetivo de un Sistema de Recomendación es generar sugerencias de nuevos ítems o predecir la utilidad de un ítem específico para un usuario particular [3].

Las dos entidades básicas de un Sistema de Recomendación son el usuario y el ítem o producto y está formado por los siguientes elementos [3]:

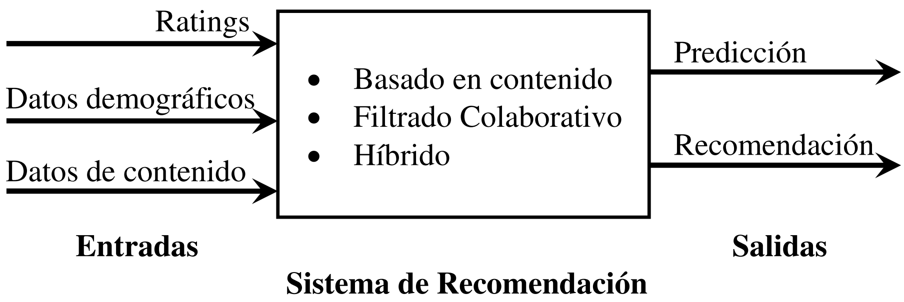

Figura 1. Esquema general de un Sistema de Recomendación.

Existe gran variedad de Sistemas de Recomendación de Películas en Internet, tales como Jinni, Taste Kid, Movielens, Netflix, entre otros. Todos ellos son diferentes, en características y resultados; algunos requieren poco o ningún ingreso de información antes de generar las recomendaciones, y otros hacen énfasis en los gustos de sus usuarios.

Objetivos

Los objetivos planteados en este proyecto de graduación fueron:

- Implementar un Sistema de Recomendación de Películas basado en el paradigma MapReduce para el procesamiento masivo de datos, utilizando el dataset de Netflix.
- Evaluar los resultados de las recomendaciones presentadas por el sistema para determinar su grado de exactitud con respecto al gusto del usuario.

CAPÍTULO 1

1. ANÁLISIS

1.1. Dataset Netflix

En este proyecto se utilizó el dataset que Netflix puso a disposición a los participantes del concurso The Netflix Prize[1]. Los archivos que utilizamos del dataset fueron los siguientes:

- Conjunto de archivos de puntuaciones por película.
- Archivo con nombres de las películas.

Cada archivo de puntuaciones por película tiene el siguiente formato:

- Id de la película (1-17770 secuencialmente).
- Id del usuario (1-2649429, no secuencialmente). Son 480189 usuarios.
- Puntuación de 1 a 5.
- Fecha de la puntuación en formato YYYY-MM-DD.

[1] http://www.netflixprize.com/

1.2. Pre-procesamiento del dataset

Esta base de datos consta de un archivo por cada película, razón por la cual fue necesario realizar un pre-procesamiento de los mismos para unificarlos de modo que puedan ser manejados por Hadoop de manera eficiente.

Como resultado del pre-procesamiento se obtuvo un conjunto de archivos con la siguiente estructura en cada línea:

- Id del usuario.
- Id de la película.
- Puntuación.

CAPÍTULO 2

2. DISEÑO

2.1. Introducción

A continuación una breve descripción de las herramientas, algoritmos y pasos utilizados en el diseño de la solución.

2.2. Herramientas utilizadas

2.2.1. Hadoop

Hadoop [4] es un framework desarrollado en Java, diseñado para correr aplicaciones en grandes clústeres de computadoras. Hadoop implementa el paradigma MapReduce [5] que es utilizado para procesar volúmenes grandes de datos en paralelo, dividiendo el trabajo en un conjunto de tareas independientes. Provee también un sistema de archivos distribuido, el Hadoop Distributed File System (HDFS), que almacena los datos en los nodos del clúster, proporcionando alto rendimiento para aplicaciones de procesamiento masivo de datos al explotar la localidad espacial de los datos durante el procesamiento de los mismos. El diseño del HDFS de Hadoop está basado en el Google File System [6] de Google.

En este proyecto se trabajó con Hadoop en su versión 0.18.3.

2.2.2. Amazon Elastic MapReduce

Amazon Elastic MapReduce [7] es un servicio Web que ejecuta programas MapReduce en Hadoop sobre instancias de Amazon EC2 (Elastic Compute Cloud) [8], utilizando Amazon S3 (Simple Storage Service) [9] como fuente de los datos a procesar y destino de los resultados.

2.2.3. MapReduce

El modelo de programación MapReduce [5] está definido como dos funciones: *Map* y *Reduce*.

Ambas funciones tratan con datos estructurados como par (clave, valor). *Map* toma un par de entrada (c1, v1) y produce una lista de pares (c2, v2) *intermedios* para cada par de entrada. Luego agrupa todos los pares con la misma clave intermedia de todas las listas, creando un grupo por cada clave intermedia diferente generada.

Map (c1,v1) → list(c2,v2)

La función Reduce recibe la clave intermedia y la lista de valores para esa clave. Luego junta todos estos valores para formar una lista de valores más pequeña. Típicamente cada llamada a Reduce produce un valor v2 o un valor vacío. Las salidas generadas por todas esas llamadas forman la lista resultante final.

Reduce (c2,list(v2)) → list(v2)

2.3. Algoritmos de filtrado colaborativo

2.3.1. Generalidades

El objetivo de un algoritmo de filtrado colaborativo es sugerir nuevos ítems o predecir la utilidad de cierto ítem para un usuario particular, basándose en sus elecciones anteriores y en las elecciones de otros usuarios con similar historial de valoraciones [10, 11].

En un escenario típico, existe una lista de m usuarios $U = \{u_1, u_2, \ldots, u_m\}$ y una lista de n ítems $I = \{i_1, i_2, \ldots, i_n\}$, donde cada usuario u_i tiene una lista de ítems I_{ui} que ha calificado. Esta calificación se puede expresar explícitamente como un rating, en este caso entre 1 y 5.

Existen dos formas básicas de algoritmos de filtrado colaborativo: basado en usuarios y basado en ítems [10].

2.3.2. Filtrado colaborativo basado en usuarios

Utilizan toda la base de datos de ítems y usuarios para generar predicciones. Emplean diferentes técnicas para encontrar un conjunto de usuarios con preferencias similares, llamado *vecindario*. Una vez obtenidos los vecindarios, se procede a generar una predicción o una lista de recomendaciones para el usuario deseado [10, 11, 12].

El vecindario se genera a partir de una matriz de similitud entre usuarios.

2.3.3. Filtrado colaborativo basado en ítems

A diferencia del basado en usuarios, la matriz de similitud se genera a partir de la similitud entre ítems. Para el cálculo de la similitud entre dos ítems se deben separar todos los usuarios que han calificado ambos ítems, de modo que la similitud se calcula sólo sobre ese grupo de usuarios. Luego del cálculo de la similitud, se procede a seleccionar los ítems más similares [10, 11].

Existen diferentes formas de calcular la similitud entre dos ítems, entre estos:

- Coeficiente de Correlación de Pearson.
- Coseno.
- Coseno Ajustado.

Una vez separados los ítems más similares, se puede calcular las predicciones para los ítems del usuario deseado.

2.3.4. Algoritmo de filtrado colaborativo utilizado

En este proyecto se utilizó el algoritmo de filtrado colaborativo basado en ítems, ya que es más eficiente que el basado en usuarios para datasets de gran tamaño [12].

La similitud entre ítems es menos variable que la similitud entre usuarios, lo que permite pre-calcular la matriz de similitudes, haciendo que el proceso de generar recomendaciones sea mucho más rápido.

Para el cálculo de la similitud entre ítems se empleó la Correlación de Pearson por ser uno de los más recomendados [13].

2.4. Diseño de la solución

Se propuso como solución al cálculo de las recomendaciones cuatro pasos MapReduce, de los cuales el primero fue ejecutado independientemente y los otros tres en forma de cascada.

Paso 1: Conteo del número de puntuaciones para cada ítem.

La entrada para este paso es el resultado del pre-procesamiento explicado en la Introducción, donde cada línea de los archivos tiene el siguiente formato:

Id del usuario<espacio>Id de la Película<espacio>Puntuación

Map: Contó cada vez que un ítem fue calificado, generando un par con el ítem como clave y con valor inicial “1”.

Reduce: La función reduce sumó todos los valores emitidos para cada ítem, formando como salida un par con el ítem como clave y como valor el número de veces que fue calificado.

Paso 2: Agrupamiento de Puntuaciones por Usuario.

La entrada de este paso es la misma entrada del Paso 1.

Map: Generó como clave el Id del Usuario y como valor la dupla Id de la Película y la Puntuación de la misma.

Reduce: Se agruparon todas las películas y sus puntuaciones, por usuario.

Paso 3: Cálculo de la matriz de similitud entre Ítems.

La entrada de este paso fue la salida del Paso 2.

Map: Generó como salida un par con clave Id de la Película y la Puntuación; y como valor cuántas veces una película fue calificada por cualquier usuario y la suma de estas puntuaciones.

Reduce: Emitió como salida un par formado por la clave Id de una Película *X* y su Similitud; y el valor formado por Id de la Película *Y* y cuántas veces ambas películas (*X* y *Y*) fueron calificadas por un mismo usuario.

La Similitud fue calculada entre las películas *X* y *Y*, donde *X* es la película para la cual se desean encontrar ítems similares, y *Y* son todas las películas comparadas con *X*.

Cabe recordar que la similitud entre ítems se calcula sobre un par de ítems que han sido calificados por un mismo usuario.

Paso 4: Identificación de los 6 ítems más similares para cada película.

La entrada de este paso fue la salida del Paso 3.

Map: El mapper es básicamente una Función Identidad.

Reduce: El reducer seleccionó los 6 ítems con valores de similitud más altos.

Todos estos pasos están descritos en el siguiente gráfico.

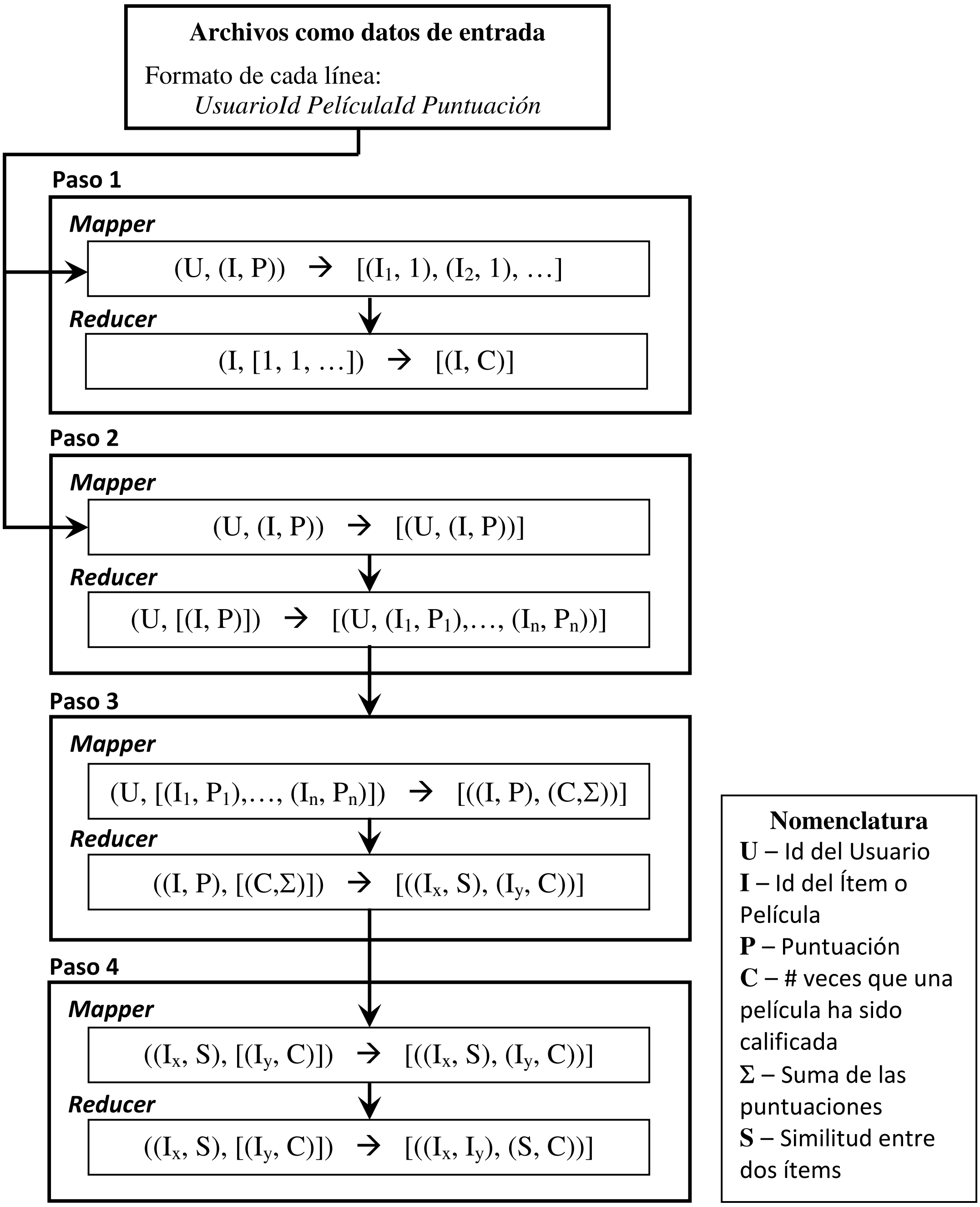

Figura 2. Flujo de proceso del modelo MapReduce empleado.

CAPÍTULO 3

3. IMPLEMENTACIÓN

3.1. Introducción

La implementación de este programa está basada en un tutorial [14] que utiliza una aplicación en Python, para encontrar los ítems similares para cada ítem de los archivos de entrada.

3.2. Código fuente

3.2.1. Archivo similarity.py

Previo al flujo de pasos MapReduce empleados, este programa contiene las funciones para el cálculo del Coeficiente de Correlación de Pearson.

```
import sys, os
import re
import csv
from math import sqrt, log, exp
from random import random
from collections import defaultdict

EPS = 20
CONF_LEVEL = 1.96

def fisher_z(x):
  z = .5*( log((1+x)/(1-x)) )
```

```
  return z

def inverse_fisher_z(x):
  zi = (exp(2*x)-1)/(exp(2*x)+1)
  return zi

def fisher_sigma(n):
  sigma = 1/(sqrt(n-3))
  return sigma

def pearsonr(sum_xx, sum_x, sum_xy, sum_y, sum_yy, n):
  r_num = n * sum_xy - sum_x*sum_y
  r_denom = sqrt((n *sum_xx - sum_x**2)*(n *sum_yy -
sum_y**2))
  if r_denom == 0.0:
    return 0.0
  else:
    return round(r_num / float(r_denom), 4)

def calc_similarity(sum_xx, sum_x, sum_xy, sum_y, sum_yy,
n, N):
    pearson = pearsonr(sum_xx, sum_x, sum_xy, sum_y,
sum_yy, n)
    if n < 4:
      return 0
    if pearson > .9999:
      pearson = .9999
    if pearson < -.9999:
      pearson = -.9999
    fisher_lower = inverse_fisher_z( fisher_z(pearson) -
fisher_sigma(n)*CONF_LEVEL)
    fisher_upper = inverse_fisher_z( fisher_z(pearson) +
fisher_sigma(n)*CONF_LEVEL)
    if pearson >= 0 and fisher_lower < 0:
      fisher_lower = 0
    if pearson <= 0 and fisher_lower > 0:
      fisher_lower = 0
    if fisher_lower*fisher_upper < 0:
      fisher_lower =
min(abs(fisher_lower),abs(fisher_upper))
    else:
      fisher_lower = min(fisher_lower,fisher_upper)

    return fisher_lower * (n/(float(n + EPS )))**2 *
(log(n)/log(N))
```

Pasos MapReduce:

Paso 1: Conteo del número de puntuaciones para cada ítem.

```
def mapper1(args):
  for line in sys.stdin:
    user, item, rating = line.strip().split()
    sys.stdout.write('LongValueSum:%s\t1\n' % item)

def reducer1(args):
  last_item, item_count = None, 0
  for line in sys.stdin:
    item = line.strip().split('\t')[0].split(':')[1]
    if last_item != item and last_item is not None:
      print '%s\t%s' % (last_item, item_count)
      last_item, item_count = None, 0
    last_item = item
    item_count += 1
  print '%s\t%s' % (last_item, item_count)
```

Paso 2: Agrupamiento de Puntuaciones por Usuario.

```
def mapper2(scale, args):
  for line in sys.stdin:
    user, item, rating = line.strip().split()
    if scale=='log':
      rating = int(log(int(rating))) + 1
    sys.stdout.write('%s,%s\t%s\n' % (user,item,rating))

def reducer2(args):
  last_user, item_ratings = None, []
  user_count, user_sum = 0, 0
  for line in sys.stdin:
    (user_item, rating) = line.strip().split("\t")
    user, item = user_item.split(',')
    if last_user != user and last_user is not None:
      rating_string = ' '.join(item_ratings)
      print "%s\t%s|%s|%s" % (last_user, user_count,
user_sum, rating_string)
      (user_count, user_sum, item_ratings) = (0, 0, [])
    last_user = user
    item_ratings.append('%s,%s' % (item,rating))
    user_count += 1
    user_sum += int(rating)
  rating_string = ' '.join(item_ratings)
  print "%s\t%s|%s|%s" % (last_user, user_count,
user_sum, rating_string)
```

Paso 3: Cálculo de la matriz de similitud entre Ítems.

Similar al concepto de *stop words*[2], se ha utilizado los stop users, donde se han excluido los usuarios que han calificado películas con mayor frecuencia pero cuidando que aquellas películas que han sido calificadas menos veces, se mantengan dentro de los datos a procesar. Para esto se valió del archivo generado como salida en el Paso 1: item_rating_counts.txt

```
def P(item, user_count, item_counts, cutoff):
  item_count = float(item_counts[item])
  if item_count > 0:
    return max(cutoff/float(user_count),
cutoff/item_count)
  return cutoff/float(user_count)

def mapper3(cutoff, itemcountfile, args):
  cutoff = int(cutoff)
  reader = csv.reader(open(itemcountfile, 'rb'),
delimiter='\t')
  item_counts = defaultdict(int)
  for item, count in reader:
    item_counts[item]=int(count)
  for line in sys.stdin:
    (user, vals) = line.strip().split("\t")
    (user_count, user_sum, item_ratings) =
vals.split('|')
    items = [x.split(',') for x in item_ratings.split()
if \
    random() < P(x.split(',')[0], user_count,
item_counts, cutoff)]
    for i, y in enumerate(items):
      for x in items[:i]:
```

[2]*Stop words* son palabras que no aportan gran significancia a la hora de realizar búsquedas dado que son extremadamente comunes en el vocabulario de cada lengua. *Fuente:* http://en.wikipedia.org/wiki/Stop_words

```
        sys.stdout.write("%s %s\t%s %s\n" % (x[0], y[0],
x[1], y[1]))


def reducer3(N, args):
  N = int(N)
  (last_item_pair, similarity) = (None, 0)
  (sum_xx, sum_xy, sum_yy, sum_x, sum_y, n) = (0, 0, 0,
0, 0, 0)
  for line in sys.stdin:
    (item_pair, coratings) = line.strip().split("\t")
    (x, y) = map(int,coratings.split())
    if last_item_pair != item_pair and last_item_pair is
not None:
      similarity = calc_similarity(sum_xx, sum_x, sum_xy,
sum_y, sum_yy, n, N)
      print "%s %s %s %s" % (item_x, similarity, item_y,
n)
      print "%s %s %s %s" % (item_y, similarity, item_x,
n)
      (sum_xx, sum_x, sum_xy, sum_y, sum_yy, n) = (0, 0,
0, 0, 0, 0)
    last_item_pair = item_pair
    item_x, item_y = last_item_pair.split()
    sum_xx += x*x
    sum_xy += x*y
    sum_yy += y*y
    sum_y += y
    sum_x += x
    n += 1
  similarity = calc_similarity(sum_xx, sum_x, sum_xy,
sum_y, sum_yy, n, N)
  print "%s %s %s %s" % (item_x, similarity, item_y, n)
  print "%s %s %s %s" % (item_y, similarity, item_x, n)
```

Paso 4: Identificación de los 6 ítems más similares para cada película.

Este resultado es luego cargado en MySQL.

```
def mapper4(min_coratings, args):
  for line in sys.stdin:
    item_x, similarity, item_y, n = line.strip().split()
    if int(n) >= int(min_coratings):
      sys.stdout.write('%s,%s\t%s %s\n' % (item_x,
```

```
      str(1.0-float(similarity)), item_y, n))

def reducer4(K, item_name_file, args):
  name_reader = csv.reader(open(item_name_file, 'rb'),
delimiter='\t')
  item_names = {}
  for itemid, name in name_reader:
    item_names[itemid] = name
  last_item, item_count = None, 0
  for line in sys.stdin:
    item_sim, vals = line.strip().split('\t')
    item_y, n = vals.split()
    item, similarity = item_sim.split(',')
    if last_item != item and last_item is not None:
      last_item, item_count = None, 0
    last_item = item
    item_count += 1
    if item_count == 1:
      print "\t".join([item, item, '1', '0',
item_names[item]])
    if item_count <= int(K):
      print "\t".join([item, item_y,
      str(round(1.0 - float(similarity),4)), n,
item_names[item_y]])
```

3.2.2. Archivo netflix_jobflow.json

Archivo utilizado para ejecutar todos los pasos MapReduce descritos anteriormente.

```
        [
          {
            "Name": "MR Paso 1: Conteo del número de
puntuaciones para cada item.",
            "ActionOnFailure": "TERMINATE_JOB_FLOW",
            "HadoopJarStep": {
              "Jar":
"/home/hadoop/contrib/streaming/hadoop-0.18-
streaming.jar",
              "Args": [
                 "-input",
"s3n://pruebanetflix/netflix/input/",
                 "-output",
"s3n://pruebanetflix/netflix/item-counts/",
                 "-mapper",    "python similarity.py
mapper1",
```

```
                 "-reducer",   "python similarity.py
reducer1",
                 "-cacheFile",
"s3n://elasticmapreduce/samples/similarity/similarity.py#
similarity.py",
                 "-jobconf",   "mapred.map.tasks=34",
                 "-jobconf",   "mapred.reduce.tasks=1",
                 "-jobconf",
"mapred.compress.map.output=true"
              ]
            }
          },
          {
            "Name": "MR Paso 2: Agrupamiento de
puntuaciones por usuario",
            "ActionOnFailure": "TERMINATE_JOB_FLOW",
            "HadoopJarStep": {
              "Jar":
"/home/hadoop/contrib/streaming/hadoop-0.18-
streaming.jar",
              "Args": [
                 "-input",
"s3n://pruebanetflix/netflix/input/",
                 "-output",
"hdfs:///home/hadoop/output2/",
                 "-mapper",    "python similarity.py
mapper2 real",
                 "-reducer",   "python similarity.py
reducer2",
                 "-cacheFile",    "s3n://similarity-
example/similarity.py#similarity.py",
                 "-jobconf",   "mapred.map.tasks=136",
                 "-jobconf",   "mapred.reduce.tasks=68",
                 "-partitioner",
"org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner",
                 "-jobconf",
"map.output.key.field.separator=,",
                 "-jobconf",
"num.key.fields.for.partition=1",
                 "-jobconf",
"mapred.compress.map.output=true"
              ]
            }
          },
          {
            "Name": "MR Paso 3: Calculo de la matriz de
similitud entre ítems.",
            "ActionOnFailure": "TERMINATE_JOB_FLOW",
            "HadoopJarStep": {
```

```
              "Jar":
"/home/hadoop/contrib/streaming/hadoop-0.18-
streaming.jar",
              "Args": [
                 "-input",
"hdfs:///home/hadoop/output2/",
                 "-output",
"hdfs:///home/hadoop/output3/",
                 "-mapper",    "python similarity.py
mapper3 96 item_rating_counts.txt",
                 "-reducer",   "python similarity.py
reducer3 480189",
                 "-cacheFile",
"s3n://elasticmapreduce/samples/similarity/similarity.py#
similarity.py",
                 "-cacheFile",
"s3n://pruebanetflix/netflix/item-counts/part-
00000#item_rating_counts.txt",
                 "-jobconf",   "mapred.map.tasks=136",
                 "-jobconf",   "mapred.reduce.tasks=68",
                 "-jobconf",
"mapred.compress.map.output=true"
              ]
            }
          },
          {
            "Name": "MR Paso 4: Identificacion de los 6
items más similares para cada película.",
            "ActionOnFailure": "TERMINATE_JOB_FLOW",
            "HadoopJarStep": {
              "Jar":
"/home/hadoop/contrib/streaming/hadoop-0.18-
streaming.jar",
              "Args": [
                 "-input",
"hdfs:///home/hadoop/output3/",
                 "-output",
"s3n://pruebanetflix/netflix/output-large-50/",
                 "-mapper",    "python similarity.py
mapper4 16",
                 "-reducer",   "python similarity.py
reducer4 25 reformatted_movie_titles.txt",
                 "-cacheFile",
"s3n://pruebanetflix/netflix/reformatted_movie_titles.txt
#reformatted_movie_titles.txt",
                 "-cacheFile",
"s3n://elasticmapreduce/samples/similarity/similarity.py#
similarity.py",
                 "-jobconf",   "mapred.map.tasks=136",
```

```
                 "-jobconf",   "mapred.reduce.tasks=68",
                 "-partitioner",
"org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner",
                 "-jobconf",
"map.output.key.field.separator=,",
                 "-jobconf",
"num.key.fields.for.partition=1",
                 "-jobconf",
"mapred.compress.map.output=true"
              ]
            }
          }
        ]
```

3.2.3. Ejecución del programa

Primero se cargó los datos de entrada, la carpeta *netflix-data*, en el repositorio *pruebanetflix* creado en Amazon S3 [9].

Luego se corrió el JobFlow en el cliente Ruby CLI [15] proporcionado por Amazon para ejecutar trabajos en Elastic MapReduce [7].

CAPÍTULO 4

4. ANÁLISIS DE RESULTADOS

4.1. Introducción

Con el objetivo de probar la exactitud los resultados obtenidos con la matriz de similitudes, se procedió a efectuar un proceso por el cual se estimaron las puntuaciones para una película en particular y así compararlas con las reales.

Las puntuaciones estimadas se basaron en la matriz de similitudes y para calcularlas se utilizó el método de la Suma con pesos [11].

4.2. Resultados obtenidos

Para el presente análisis se utilizó la película “Ghost”. La matriz de similitudes calculada se encuentra en la Tabla I.

Película 1	Película 2	Similitud
Ghost	Pretty Woman	0.3458
Ghost	Sister Act	0.3364
Ghost	Dirty Dancing	0.3226
Ghost	Titanic	0.283
Ghost	What Women Want	0.2751
Ghost	Mrs. Doubtfire	0.2637

Tabla I. Ghost y sus 6 películas más similares.

Una vez obtenida la matriz de similitudes, se realizaron los cálculos para determinar las puntuaciones estimadas para las 6 películas con las similitudes más altas.

Los cálculos presentaron los resultados detallados en la Tabla II.

Película	Puntuación real	Puntuación estimada
Pretty Woman	5	4,08250
Sister Act	3	3,56380
Dirty Dancing	5	4,14410
Titanic	5	4,07080
What Women Want	3	3,48420
Mrs. Doubtfire	4	3,56380

Tabla III. Puntuaciones Reales vs. Estimadas.

Adicionalmente, para realizar la evaluación del rendimiento de los algoritmos implementados se utilizó el error medio absoluto (MAE) [11], que presentó los resultados detallados en la Tabla III.

Puntuación real	Puntuación estimada	Diferencia
5	4,0825	0,9175
3	3,5638	-0,5638
5	4,1441	0,8559
5	4,0708	0,9292
3	3,4842	-0,4842
4	3,5638	0,4362

Tabla III. Diferencia entre las Puntuaciones reales y estimadas.

El valor calculado del parámetro MAE para las 6 películas de ejemplo fue 0,34847.

El valor del parámetro MAE se puede disminuir utilizando para los cálculos las 24 películas con las más altas similitudes, tal como se muestra en la Tabla IV.

Película	Similitud	Puntuación real	Puntuación estimada
Pretty Woman	0,3458	5	4,0825
Sister Act	0,3364	3	3,5638
Dirty Dancing	0,3226	5	4,1441
Titanic	0,283	5	4,0708
What Women Want	0,2751	3	3,4842
Mrs. Doubtfire	0,2637	4	3,5638
Stepmom	0,2557	4	3,84640
Miss Congeniality	0,2527	4	3,60420
Top Gun	0,2446	5	4,13070
Double Jeopardy	0,2407	4	3,72890
Air Force One	0,2349	4	3,99820
Pearl Harbor	0,2317	4	3,79570
Steel Magnolias	0,2274	5	4,20580
The Net	0,2267	4	3,73680
Beaches	0,2259	3	3,89300
Don't Say a Word	0,2247	3	3,50180
Along Came a Spider	0,2233	3	3,59400
Erin Brockovich	0,2212	4	3,98260
Runaway Bride	0,2211	4	3,66790
The General's Daughter	0,2209	4	3,74100
Twister	0,2194	4	3,73780
Sleepless in Seattle	0,2193	5	4,20580
Three Men and a Baby	0,2109	4	3,52670

Tabla IIV. Similitudes, puntuaciones reales y estimadas para las 24 películas más similares.

El valor calculado del MAE para las 24 películas fue 0,18187. Esto indica que las estimaciones de las calificaciones (predicciones) basadas en la matriz de similitudes son bastante buenas y dan resultado.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

1. Los sistemas de recomendación al ser ampliamente utilizados actualmente, reclaman el uso de tecnologías que vayan acorde a los tiempos. Es por esto que el uso de MapReduce cubre satisfactoriamente esta necesidad.

2. Existen diversas maneras de generar recomendaciones; en este proyecto se utilizó un algoritmo de Filtrado Colaborativo basado en Ítems, usando los ratings o puntuaciones que los usuarios han dado a las películas. Pero se podría utilizar además ciertas características de las películas como categoría, actores, etc., para obtener mejores resultados; es decir, recomendaciones más exactas.

3. Al evaluar los resultados del algoritmo de Filtrado Colaborativo mediante el Error Medio Absoluto, se demostró que las predicciones calculadas son bastante cercanas a las puntuaciones reales dadas por los usuarios.

RECOMENDACIONES

1. Dado que los usuarios no siempre proporcionan una calificación a las películas, es altamente probable que se presenten problemas de dispersión en la información; para esto se recomienda emplear técnicas de reducción de dispersión en el proceso de generar recomendaciones.

2. Para pre-calcular la matriz de similitudes entre ítems, es recomendable el uso de un computador con buena capacidad de procesamiento y memoria, puesto que dicho cálculo demanda un amplio uso de estos recursos.

3. Adicionalmente a la calificación, incluir en el análisis datos sobre las preferencias de los usuarios (sexo, edad, ocupación, etc.) e información sobre las películas (cómo el género), esto ayudaría a formular predicciones más exactas.

BIBLIOGRAFÍA

1. Amazon Web Services. http://aws.amazon.com/. Último acceso septiembre 2, 2009.

2. Lathia, N. Computing Recommendations with Collaborative Filtering. Chapter 2 in "Collaborative and Social Information Retrieval and Access: Techniques for Improved User Modeling". July 2008.

3. Vozalis, E., Margaritis, K. G. “Analysis of Recommender Systems' Algorithms”. In Proceedings of the 6th Hellenic European Conference on Computer Mathematics and its Applications (HERCMA-2003). Athens, Greece.

4. Hadoop. http://hadoop.apache.org/. Último acceso agosto 28, 2009.

5. Dean, J. y Ghemawat, S. “MapReduce: Simplified Data Processing on Large Clusters“. In Proceedings of the Sixth Symposium on Operating System Design and Implementation (OSDI 2004). San Francisco, CA-EE.UU.. Diciembre, 2004.

6. Ghemawat, S., Gobioff, H., y Leung, S. “The Google File System“. In Proceedings of the 19th ACM Symposium on Operating Systems Principles. Lake George, NY-EE.UU. Octubre, 2003.

7. Amazon Elastic MapReduce Developer Guide. http://docs.amazonwebservices.com/ElasticMapReduce/latest/DeveloperGuide/. Último acceso septiembre 2, 2009.

8. Amazon Elastic Compute Cloud (EC2) Developer Guide. http://docs.amazonwebservices.com/AWSEC2/latest/DeveloperGuide/. Último acceso septiembre 2, 2009.

9. Amazon Simple Storage Service (S3) Developer Guide. http://docs.amazonwebservices.com/AmazonS3/latest/. Último acceso septiembre 2, 2009.

10. Sarwar, B., Karypis, G., Konstan, J., y Riedl, J. "Item-based Collaborative Filtering Recommendation Algorithms". In Proceedings of the 10th International World Wide Web Conference. Hong Kong, Hong Kong. May 1-5, 2001.

11. Galán, S. "Filtrado colaborativo y Sistemas de Recomendación". Leganés, Madrid, España. Enero 2007.

12. Karypis, G. "Evaluation of Item-Based Top-N Recommendation Algorithms". In CIKM'01: Proceedings of the tenth international conference on Information and knowledge management. pp. 247-254. Atlanta, Georgia, USA. 2001.

13. Papagelis, M., Plexousakis, D. "Qualitative Analysis of User-Based and Item-Based Prediction Algorithms for Recommendation Agents". In CIA 2004: Proceedings of the 8th International Workshop of Cooperative Information Agents. pp. 152-166. Erfurt, Germany. September 27-29, 2004.

14. Finding Similar Items with Amazon Elastic MapReduce, Python, and Hadoop Streaming.http://developer.amazonwebservices.com/connect/entry.jspa?externalID=2294. Último acceso septiembre 2, 2009.

15. Amazon Elastic MapReduce Ruby Client. http://developer.amazonwebservices.com/connect/entry.jspa?externalID=2264. Último acceso septiembre 2, 2009.