Algorithm design -

Collaborative-based Relevance Graph

# Overview

We are designing an algorithm to calculate item-item similarity for all the movies based on the collaborative-based approach.

The input data is all events in the user viewing history data. These events are used to build a movie-user matrix of binary value whose number of rows equals to the number of movies and number of columns equals to the number of users. Each value at (row i, column j) presents if user j has watched (or recorded, purchased, etc.) on a movie i. Each row in the matrix is a user view vector of a movie. The similarity between two movies <m1, m2> are calculated as cosine of two user view vectors of the two movies. The algorithm to calculate movie similarity using collaborative-based approach is modeled in Figure 1. We have developed an efficient method to handle the creation of the matrix and cosine calculation for the input data set of up to 5000 movies and 100000 users.

The actual input data can contain billions user viewing history events of 22M users watching hundred thousand movies. The intermediate data of unique <movie, user> pair and the movie-user matrix scale up to 400x compared to our base-line (5000 movies x 100k users) sample data as showed in Table 1. The output movie similarity also consumes large amount of memory. Because the consumed memory as well as the computation exceeds the capacity of a single node, a parallel algorithm is needed to solve this scaling up issue.

|  |  |  |  |
| --- | --- | --- | --- |
| *Movies x Users* | *Input data* | *User-movie matrix* | *Movie-movie matrix* |
| 5K x 1M | 15M unique <movie, user> pairs | 57MB | 100Mb |
| 100K x 22M | 6.6B unique <movie, user> pairs | 25GB | 40GB |

Table Estimation of resources as scaling-up



Figure Algorithm to calculate movie similarity with Collaborative-based approach



Figure Parallel algorithm to calculate movie similarity with Collaborative-based approach with RevoR on MapReduce

# Parallel algorithm for movie similarity calculation

We propose a parallel algorithm to calculate movie similarity. The algorithm can be implemented on RevoR and run on Hadoop MapReduce environment. The algorithm run in multiple steps as below:

**Step 1:** Use Hive to query the raw UVH, clean up the data, then produce a table of all unique <movie, user> pairs indicating the user has watched/purchased/recorded/… the movie. The table is then grouped by movie and divided into multiple chunks. A constraint is that no data related to one movie be divided into different chunks.

**Step 2:** Construct a part of the movie-user matrix from each data chunk created in Step 1. Each part contains user viewing vectors of multiple movies and no movies is separated into two or more chunks.

**Step 3:** Execute multiple parallel jobs to calculate cosine similarity for each pair of movie-user matrix’s parts.

**Step 4:** Aggregate the cosine results from different jobs to construct movie-movie similarity matrix.

**Step 5:** Extract the top similar movies and convert the result into json format.

# Discussions on parallelism and optimization

* Step 1 & 2: create the matrix on local or distributed chunks of <movie, user> tables on different nodes to create the matrix where the table is stored? This is depending on the amount of data
* Step 3: A node is assign a job to calculate similarity for Mi and Mj, it would be more efficient to assign the node to perform calculation for (Mi, Mk) or (Mj, Mk).
* Any more optimizations?

# Task and Schedule

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Task | Status (by May 28) |
| Week 1 | April 20 – 24 | * Build a sample dataset of 5K movies x 100k users | 100% |
| Week 2 | May 11 – 15 | * First version of the implementation * Run on sample dataset and evaluate the results | 100% |
| Week 3 | May 18 – 22 | * Optimize the matrix construction * Optimize the cosine calculation | 100% |
| Week 4-5 | May 23 – Jun 5 | * Develop the parallel algorithm to run on RevoR/ HadoopMR | 30% |
| Week 6-7 | Jun 8 -19 | * Evaluate the parallel algorithm * Tune the algorithm | 0% |