Double Matrix Factorization Recommendation System

Hua Liu
Department of Electrical and Computer Engineering
Rutgers, the State University of New Jersey
hl678@scarletmail.rutgers.edu

Yao Lu
Department of Electrical and Computer Engineering
Rutgers, the State University of New Jersey
yl931@scarletmail.rutgers.edu

Jianqin Gao
Department of Electrical and Computer Engineering
Rutgers, the State University of New Jersey
jg1204@rutgers.edu

ABSTRACT

Recommendation system is mostly built using collaborative filtering or content-based filtering.The collaborative filtering will suffer the performance loss when the rating matrix is sparse, which is actually a common case in the real world. In this paper, we propose a method based on Alternating Least Square to alleviate the effect of the cold start problem when the rating matrix is very sparse.

Author Keywords

recommendation system; collaborative filtering;content-based; ALS; matrix factorization

INTRODUCTION

The motivation for collaborative filtering comes from the idea that people often get the best recommendations from someone with tastes similar to themselves.Collaborative filtering encompasses techniques for matching people with similar interests and making recommendations on this basis.Another common approach when designing recommendation systems is content-based filtering. Content-based filtering methods are based on a description of the item and a profile of the users preference. In a content-based recommendation system, keywords are used to describe the items and a user profile is built to indicate the type of item this user likes. In other words, these algorithms try to recommend items that are similar to those that a user liked in the past (or is examining in the present). In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended.

As it is shown in the Figure 1, we will have an initial rating matrix that contains the rates from the users to the movies they have watched.And what the recommendation system do is to fill the blank elements in this rating matrix.

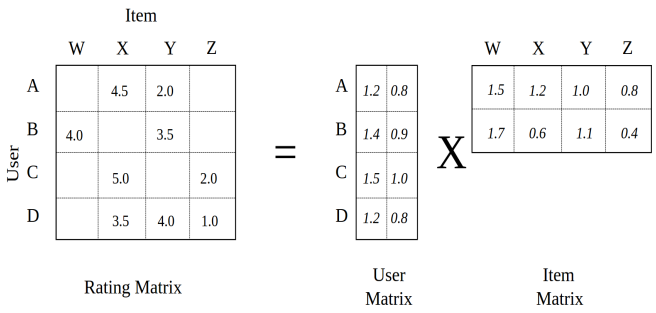

Figure 1. User-Item rating matrix

The built-in method in the recommendation system of Spark framework is Alternating Least Squares(ALS) algorithm. The ALS is one way of implementing collaborative filtering based on matrix factorization.

However, in the real world, most of the rating matrix is extremely sparse, which means that most of the entries in the matrix are unknown.The problem caused by the sparsity is that there are too few elements we can use in the matrix factorization and it will decrease the performance of the algorithm.Another problem that is commonly seen in the sparse matrix is cold start problem.The collaborative filtering makes the prediction for the rate value based on the user's preference, but in the sparse matrix, the user usually may only rate very few or even one movie. Therefore, it is hard for the recommendation system to predict the preference of the users without enough information.

The Figure 2 is the result of MSE(mean square error) vs. the sparsity of matrix.Here we change the sparsity of the matrix by changing the training ratio of the training data set with the fix ratio of test data set.As we can see, when we make the rating matrix more and more sparse(decrease the training ratio), the mean square error of the rating keeps increasing, which means that the performance of the algorithm will be

CHI'16, Sept 23–Dec 11, 2016, New Brunswick, the United States.

Figure 2. MSE vs. Sparsity of matrix

decreased while the matrix is getting more sparse.Here the algorithm we use to test is the original ALS algorithm implemented in the Spark mllib and the data set is from Movielens 1m(1 million ratings from 6000 users to 4000 movies).

RELATED WORK

There are several methods that have been proposed to deal with the sparsity problem. Most popular approaches proposed include dimensionality reduction of the user-item matrix, application of associative retrieval technique in the bipartite graph of items and users, item-based similarity instead of user-based similarity, and content-boosted collaborative filtering. The dimensionality reduction approach addresses the sparsity problem by removing unrepresentative or insignificant users or items so as to condense the user-item matrix. More advanced techniques to achieve dimensionality reduction have been proposed as well. Examples include statistical techniques such as Principle Component Analysis (PCA)[4] and information retrieval techniques such as Latent Semantic Indexing (LSI)[1][3][5].Content-based collaborative filtering approaches take advantage of additional information regarding items as well as a metric to compute meaningful similarities among them.

There is a method[6] proposed in a paper is that the author define a transitive properties between users in the context of a social network. In the Figure 3, suppose that users S, N have rated item I1 and users N, T have rated I2. Classic Collaborative Filtering will associate user S with user N and user N with user T, but not user S with user T. However, a more sophisticated approach that incorporates transitive interactions would recognize the associative relationship between user S and user T and infer this indirect association. To deal with this problem, they adopt a method of inferring trust between users that are not directly associated to each other. Thus, in the example, it is possible to infer trust between the source user S and the target user T through the intermediate user N. According to this process, trust is propagated in the network and associations between users are built, even if they have no co-rated item.

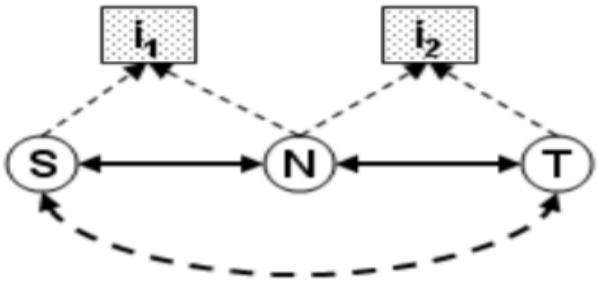

Figure 3. Trust Inferences

Another solution to the sparsity problem is hybrid recommendation system, which will combine two or more recommendation algorithms to overcome the weakness of each algorithm.The weighted hybrid technique combines different algorithms with different weights.[2]The core part of it is how to decide the weight for each algorithm.

There is also one more method that is similar to our implementation.A hybrid approach to recommendation system based on matrix factorization.They combine the original rating matrix with all extracted feature information in a single model. Inflating the original rating matrix with content-based features is expected to improve the performance for rating prediction.Actually it combines the collaborative filtering and content-based filtering by adding more information based on the characteristics of the items during the matrix factorization stage.What it differs from our method is that use the additional information to construct larger matrix but we use the information separately.

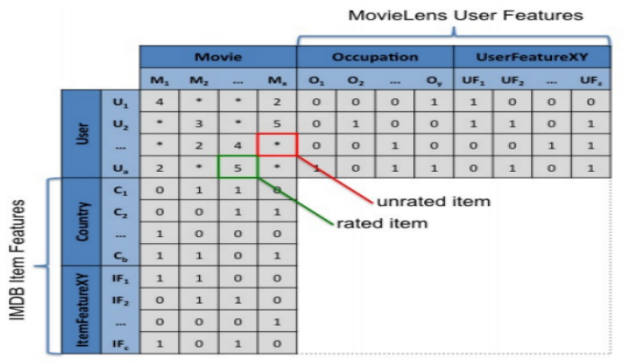

Figure 4. Rating matrix after combination

CONTRIBUTION

To solve the sparsity problem in matrix factorization based recommendation system, we create a hybrid matrix factorization based recommendation system and test the system with training matrix of variate sparsity levels. In addition, we deploy the system to amazon Elastic Map-reduce (EMR) cloud platform to test the parallelism and its suitability for cloud computing which generally has a bigger cost for communication between distributed processors.

The main character of our system compared to others is that we use a second, content information considered, matrix factorization based system to optimize part of the prediction from the first matrix factorization based system. Specifically, with Alternating Least Squares (ALS) as our adopted factorization algorithm, in our first direct ALS based system, we train the ALS model with 1 million user-movie-rating records. While in the second ALS based system, in order to improve the prediction result for users who only rated a extremely small amount of movies, and for movies that has only been rated by a small amount of users, we take consideration of movie genera information.

As is shown in section Result, the second user-tag ALS based system outperforms the direct ALS system in accuracy, for movies that have been rated for a relatively small amount of users, and for users who have rated only a few movies.

IMPLEMENTATION

The implementation of our double ALS based recommendation system consists of the implementation of a user-movie-rating based direct ALS training model and another user-tag-rating based indirect ALS training model. The second ALS model is trying to capture the relationship between a user and a tag. And it uses this information to predict the ratings for a user-movie pair by first prediction the rating for the user towards each of the tag inside of the movie, and take the average rating of all tags as the final predicted rating for the movie.

The reason that the second indirect ALS user-tag system theoretically outperforms the first one for users and items with sparse records is because of an observation: for user who rated only a small amount of movie, it is better to predict his or her rating by the content of the movie instead of finding similar users or train user feature matrix which has too little training data to get the real feature of the user.

The whole process can be divided into three phases: training and testing direct user-movie ALS model, training and testing indirect user-tag ALS, and testing the influence of sparsity in both systems.

ALS Algorithm

The basic procedure of Matrix Factorization is to firstly decompose the large user-item matrix into a lower dimensional user feature matrix P and a item feature matrix Q such that the mean square error of the existing entries inside of the recomposed Matrix, using P multiplied by Q, is minimum. Then to estimate the user u's rating for a specific movie i, we multiply the uth P_u and Q_i according to the Equation 1:

$$r_{ui} = p_u^T q_i \tag{1}$$

Those factors (i.e. P, Q) are learned by minimizing the quadratic loss function in Equation 2:

$$min_{q,p} \sum (r_{ui} - p_u^t q^i)^2 \tag{2}$$

To minimize the quadratic loss function in Equation 2, there are several optional algorithms. For example, in an SGD (Stochastic Gradient descent) approach, after calculating the error for each training (user,item,rating) pair.The parameters vector is updated by a factor in the opposite direction of the gradient.

Alternating Least Squares (ALS) represents a different approach to optimizing the loss function. In SGD, it's a non-convex iterative optimization problem. While in ALS, he key insight is that you can turn the non-convex optimization problem in Equation 2 into an "easy" quadratic problem if you fix either p_u or q_i. ALS fixes each one of those alternatively. When one is fixed, the other one is computed, and vice versa.

System Design

- Randomly partition the user-movie-rating records into training data and testing data.
- Train direct user-movie ALS in Spark, using the library ALS.scala come with Spark source code. And calculate the Mean Square Error (MSE) using testing data.
- Implement indirect user-tag ALS, as is shown in Figure 5:

  (1) As is shown in block "UMR2UTR", transform RDD(user,movie,rating) into RDD(user,tag,rating) using the information of (movie,tags). A rating record of movie i with n tags would be mapped to n records of RDD(user,tag,rating).

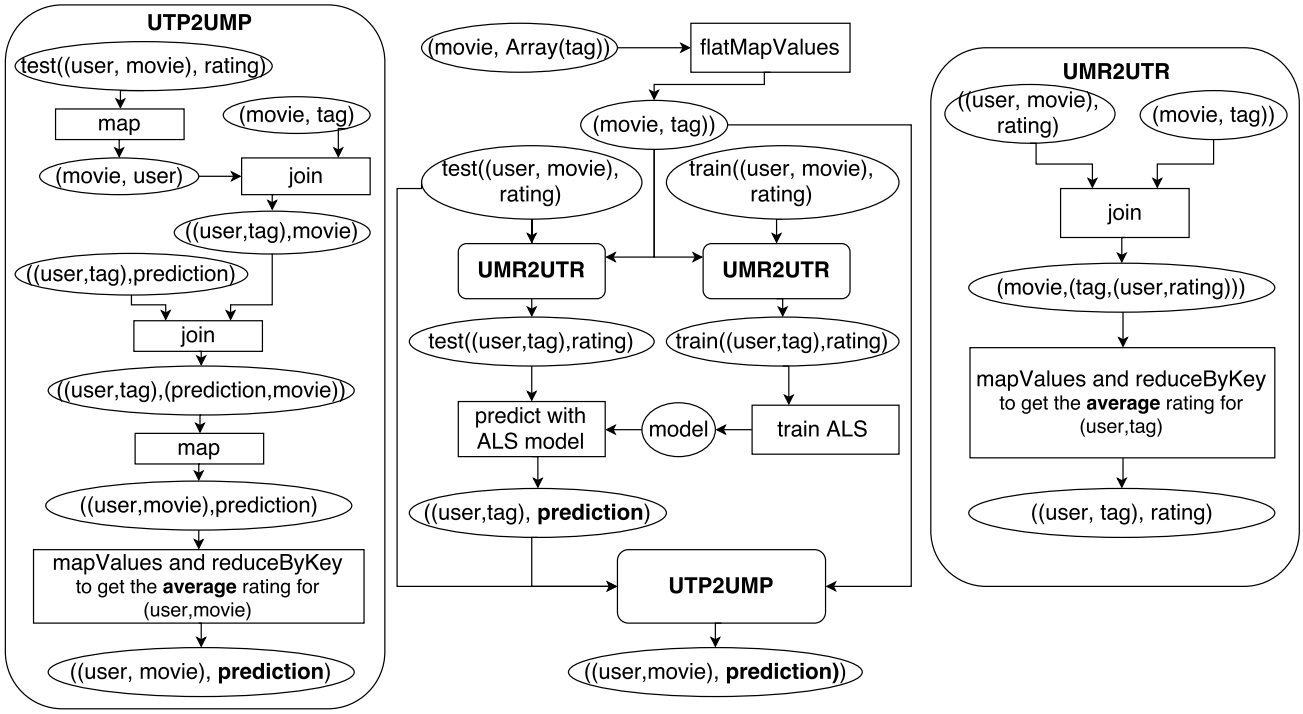

Figure 5. RDD Flow for indirect user-tag ALS based recommendation system

(2) Use RDD(user,tag,rating) to train a ALS model. And use it to predict the rating of a user to a tag.

(3) As is shown in block "UTP2UMP", sum up user As prediction for all the tags of a movie B, and take the average as the final predicted rating for user A to movie B.

- For all the testing data, group the prediction error by 'count(movie)' and 'count(user)', where 'count(movie)' means 'the number of rating a movie get in total' and 'count(user) means 'the number of ratings a user get in total'.

Sparsity Testing System

Actually for each item in the rating matrix, they have different sparsity between each other as some items may have many rates while others may have very few rates. To specify how sparse the item is, we take the number of rates to an item as the indicator of the sparsity of the item, that is the count.We classify all these ratings of different movies according to the count of the movie in order to observe the performance of the algorithm when facing movies that have different sparsity, especially the case that movies only have very few rates.Every time the test data is generated by randomly splitting from the original data set, but in order to guarantee the consistency of test data we should keep it unchanged.So in our future work, we will keep the test data fix.

train:test	Direct ALS	Indirect ALS	Final Indirect ALS
0.3:0.2	0.8643	0.6854	1.0673
0.4:0.2	0.8381	0.6902	1.0750
0.5:0.2	0.8269	0.6232	1.0527
0.6:0.2	0.8065	0.6489	1.0520
0.7:0.2	0.7935	0.6606	1.0541
0.8:0.2	0.7934	0.6360	1.0489

Table 1. MSE for original ALS model, indirect user-tag ALS model, and the final indirect ALS system's user-movie prediction

RESULTS

Local Result

Firstly to evaluate the performance of our Double Latent Factor Learning based Recommendation system, we run the program locally together with the original ALS to compare the result.

The experiment starting at the condition that ratio of training set to test is 8:2, the result is shown in Figure 6: When we decrease the training ratio, which means the rating matrix becomes more sparse. As we can see in the Table 1, the original ALS algorithm will be effected by the sparsity more seriously, while our indirect ALS algorithm MSE does not increase as the rating matrix is getting more and more sparse.

In Figure 6, we explored the influence of sparsity on the two ALS based system. The x-axis represents the count of the

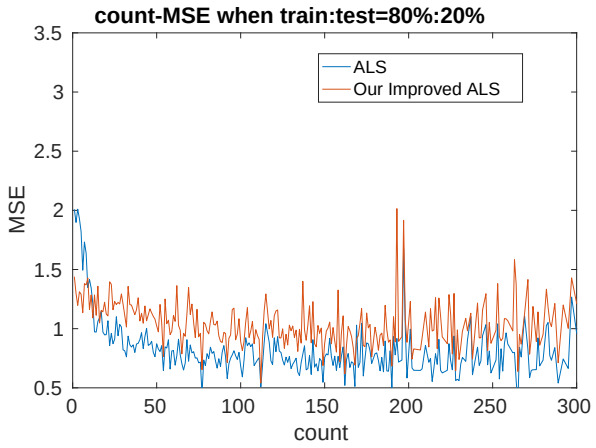

Figure 6. relationship between MSE and movie count at ratio 80%:20%

movie, which is the amount of test data, how many comments are recorded for a movie. the y-axis means the MSE(Mean Square Error), a kind of method measuring the deviation derived in the test data. Although our improved ALS is beaten for most time, its feature has already appeared: when the count is very small, our result is more precise than original ALS. Considering the problem we want to figure out, it's reasonable that apply a more sparse input will zoom up our performance. We change the ratio to 5 to 5 and record the experiment result. The out put data is indicated in Figure 7:

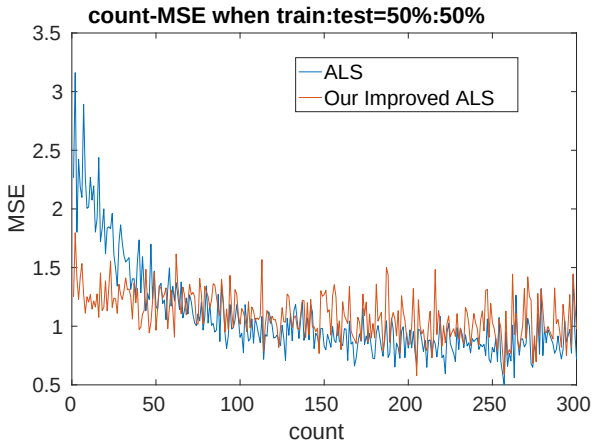

Figure 7. relationship between MSE and movie count at ratio 50%:50%

From above we could easily conclude the performance of our improved ALS during the situation that training set is equal to test set. When the number of count is less than 50, our algorithm has much better MSE than the original ALS, which means that the cold start problem we mentioned before got well solved. For the part after this point, our algorithm lacks in the precise but in general the MSE of both algorithm is quite similar.

We discover that our algorithm get highly developed when the input data(training set) got really sparse. Moreover, we took more steps changing the ratio between training to test set, the result is remarkably, As (train:test) get lower (i.e. training matrix gets more sparse), the cross points for the two gets shifted right, which means more part in a data set get enhanced prediction using our algorithm.

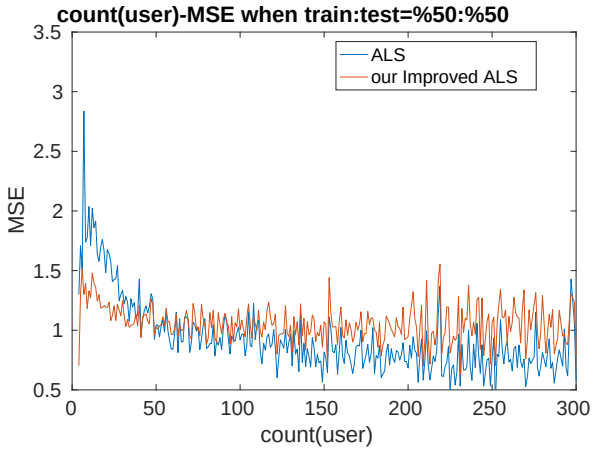

Figure 8. relationship between MSE and user count at ratio 50%:50%

What's more, in our implementation, to make our advantage more convincing, we do one more operation by replacing the count for movies with count for users, for example, if count is 20 this means that the user comments 20 movies. The result is shown in Figure 8: where the conclusion is almost compatible with our design, fir those who comments few movies, our Double Latent Factor Learning based Recommender provides a more acceptable recommendation for them.

Deploy to AWS

To fulfill the implementation of our project on the cloud computing framework, we deploy our experiment into the amazon web service, considering all factors, we finally select Amazon Elastic MapReduce(EMR) as our cloud implementation platform. Compared with a more traditional web service EC2, EMR takes its advantage at convenient and effeciency. It provides a managed Hadoop framework to process large amount of data across dynamically scalable Amazon EC2 instances. In our deployment, we change the number of instances to test our Spark application. Here is the result shown in Figure 9:

A clear observation from the result is that the relationship between running efficiency and number of executors is quite notable, as indicated, we started at 2 executors condition and time decreases non-linearly when we plus new executor, This is easy-understanding because new executors get participated

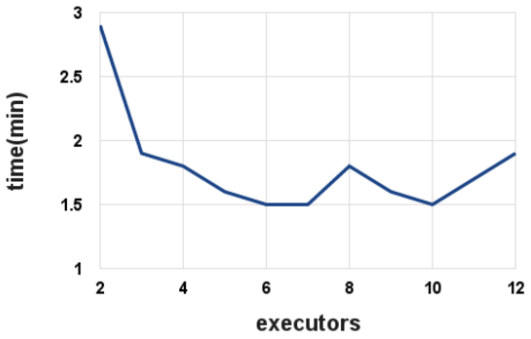

Figure 9. Execution time regarding number of executors

in the process to do calculation. As the ALS algorithm is not a simple linear-programming project, the improvement is not strictly follow the numerical expectation.

However, during our experiment, we detect the phenomenon that when the number of executors is over 7 which means there are over 7 slaves instance besides the master instance, the execution time just become to fluctuate.

The reason to explain the data-wrinkle can be concluded into two points:

1. The communication cost, as described, the number of executors means the total number of nodes in the cluster, when you have large number of slave nodes managed by the master node, the communication cost must be considered as one of main factor affecting efficiency, during the process of computing data, different work node has different priority which finally made up the work flow, which means communication is everywhere to ensure the correct execution sequence and avoid the reading or writing clash. For general enterprise use, they have billions of input data to process, not like presented in our experiment, so probably 8 executors is absolutely enough for our task and over-use of executor cause the unnecessary for communication cost.

2. The algorithm cost, another explanation derives from the algorithm itself, in our algorithm, we didn't quantitatively modify the data-partition, instead, we let spark automatic split the input data, when deploying the application into the cloud web service, the most scientific action is to re-size the number of instances to make it adaptable to the algorithm design, for example, the Scala lib function distribute the data into 8 blocks, during the EMR process, if more than executors are issued by user, there will be spare executor existed, they are not involved into the work but will occupy the time and space, like the definition and initialization of the variations. As the result, the fluctuation happens regarding the current algorithm and data size.

CONCLUSION

In this paper, we propose a new method in order to make the recommendation system performs better when there exists cold start problem in a sparse matrix and it has been proved to performs better than the original ALS algorithm that is implemented in the Spark mllib module.By using the tag information of the movies, we can predict the users preference based on the content of the movies when there are almost no information of the users we can use. Although our algorithm performs better under the certain circumstance, the original ALS algorithm can still beat our method when the matrix is not that sparse. Therefore, the current possible optimized method we think about is to combine these two methods in order to overcome the weakness of each algorithm.And the combination of these methods can be included in our future work.

FUTURE WORK

In the future, we plan to get our project developed from four aspects.

First of all, it's a regret in our research that we simply use one movie database as our input, due to the time limit and hardware problem, for cloud computing application, it's target is usually defined as mass data containing complex computation and interaction, so the result based on our benchmark, about 1MB size, is not convincing enough to strongly prove our conclusion, despite the topic we study. We consider the precisely instead of capacity, it seems there's no relationship while applying our algorithm on big or small data, but due to the sparsity of the input matrix may vary in different database, it's scientific and significant to duplicate our experiment on different scale database.

Secondly, we analysis the whole process presented by the EMR console, the figure is listed in Figure 10:

The figures shows the details around the function execution in a specific task, the width of rectangle represents the time cost, it's observable that the count function occupies large percentages of total execution time, which means there's still space left to be improved by minimizing the use of count function. What's more, it's essential to Come up with a formula to calculate the switch point, in form of Sparsity(count of input matrix). Since we have achieved the foundation of advantages of our algorithm in this paper, to optimize the performance to the most extend, it's challenging to find the switch point regarding certain input data, for the former part before the switch

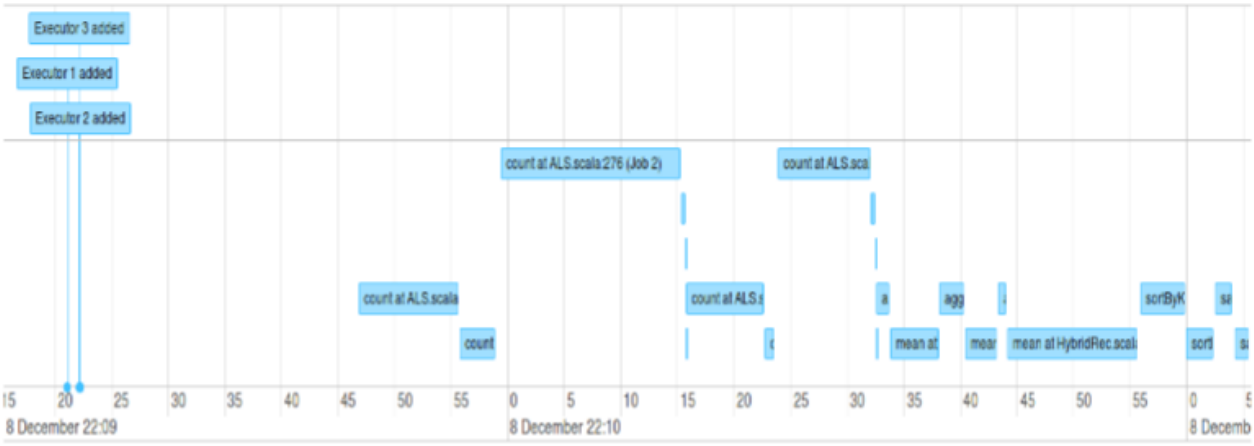

Figure 10. Execution time regarding number of executors

point of the data, we employ our improved ALS algorithm, besides, for the remaining part of data, we adopt the original ALS, thus the combination result will have better expectation than ever before, the bottleneck is to judge the sparsity of the data and tier them apart.

In the end, considering the reality of a user, who has his personal preference for different movies, however, there's a possibility that his taste migrates as the time, like, one prefers western movies before but recently Japanese movie attracts him bit more, the change reflects on the tag and time, which ca be concluded as time-stamp. Trying to collect such information and design a developed ALS, we may need more pairs of input matrix and computation, but it's a valuable direction of further work, worth discovering.

ACKNOWLEDGMENT

We would like to thank Professor Dehnavi for her instruction and support. Our data set is based on MovieLens, besides we appreciate all the developers who contribute to the cloud computing service and engineers who struggle with the problems.

REFERENCES

1. Billsus, D., and Pazzani, M. J. Learning collaborative information filters. In *Icml*, vol. 98 (1998), 46–54.
2. Burke, R. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction 12*, 4 (2002), 331–370.
3. Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. Indexing by latent semantic analysis. *Journal of the American society for information science 41*, 6 (1990), 391.
4. Goldberg, K., Roeder, T., Gupta, D., and Perkins, C. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval 4*, 2 (2001), 133–151.
5. Hofmann, T. Collaborative filtering via gaussian probabilistic latent semantic analysis. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, ACM (2003), 259–266.
6. Papagelis, M., Plexousakis, D., and Kutsuras, T. Alleviating the sparsity problem of collaborative filtering using trust inferences. In *International Conference on Trust Management*, Springer (2005), 224–239.