Performance of Alternating Least Squares in a distributed approach using GraphLab and MapReduce

Elizabeth Veronica Vera Cervantes, Laura Vanessa Cruz Quispe, José Eduardo Ochoa Luna
National University of San Agustin
Arequipa, Peru
elizavvc@gmail.com, lcruzq@unsa.edu.pe, eduardo.ol@gmail.com

Abstract

Automated recommendation systems have been increasingly adopted by companies that aim to draw people attention about products and services on Internet. In this sense, development of distributed model abstractions such as MapReduce and GraphLab has brought new possibilities for recommendation research tasks due to allow us to perform Big Data analysis. Thus, this paper investigates the suitability of these two approaches for massive recommendation. In order to do so, the Alternating Least Squares (ALS), which is a Collaborative Filtering algorithm, has been tested using recommendation benchmark datasets. Results on RMSE show a preliminary comparative performance analysis.

1 Introduction

Data on the Internet is increasing, e-commerce sites, blogs and social networks spread the word about new products and services everyday. This social media information overwhelms any user, who has a given profile and therefore could not be interested in most of these offers (Koren et al., 2009).

In this sense, recommendation systems have gained momentum, because they "filter" products and services for users according to behavior patterns. Traditional approaches for automated recommendation range from Content-Based, Collaborative Filtering and Deep Learning systems (Adomavicius, 2005; Shi et al., 2014). However, to handle the current amount of available data we need to resort to frameworks for large-scale data processing.

Recently, the machine learning community has been increasingly interested in the task of managing Big Data with parallelism (Zhou et al., 2008; De Pessemier et al., 2011; Xianfeng Yang, 2014). However, parallel algorithms are extremely challenging and traditional approaches, despite of being powerful like MPI, rely on low levels of abstraction. On the other hand, distributed models such as GraphLab(Low et al., 2012) and MapReduce(Xiao and Xiao, 2014; Dean and Ghemawat, 2008) foster high levels of abstraction and, therefore, they are more intuitive. The aim of this paper is to investigate whether these distributed models are suitable for recommendation tasks. In order to do so we evaluate the Alternating Least Squares (ALS) algorithm, a parallel collaborative filtering approach(Koren et al., 2009; Schelter et al., 2013), in both GraphLab and MapReduce frameworks. We evaluate the performance on the MovieLens and Netflix datasets. According to preliminary results, GraphLab outperforms MapReduce in RMSE, when Lambda, iterations number and latent factor parameters are considered. Conversely, MapReduce gets a better execution time than GraphLab using the same parameters in MoveLens dataset. The paper is organized as follows. In Section 2, related work is described. Background is given in Section 3. Our proposal is showed in Section 4. Preliminary results are depicted in section 5. Finally Section 5, concludes the paper.

2 Related Work

Several distributed platforms have been used for studying performance of machine learning algorithms, for instance, a Matrix Factorization based on collaborative filtering over MapReduce model was proposed in (Xianfeng Yang, 2014; De Pessemier et al., 2011). In Low et al. (2012), some advantages and disadvantages of using GraphLab and MapReduce were described. For instance, MapReduce fails when there are computational dependencies on data, but it can be used to extract features from a massive collection.

In addition, MapReduce is targeted for large data centers, it is optimized for node-failure and disk-centric parallelism. Conversely, In GraphLab it is assumed that processors do not fail, and all data is stored in shared memory.

In Low et al. (2012), the Alternating Least Squares (ALS) algorithm was implemented over several platforms: GraphLab, Hadoop/MapReduce and MPI. Comparison results show that applications created using GraphLab outperformed equivalent Hadoop/MapReduce implementations by 20-60 times(Xianfeng Yang, 2014) .

Our work is most related to Low et al. (2012), but we focus on the evaluation of different configurations of ALS algorithm over GraphLab and MapReduce. Thus, we aim at obtaining optimal parameters that allow us to improve algorithm performance. Moreover, comparisons were based on RMSE and time execution values. The parameters considered are:

- Lambda, which is the regularization parameter in ALS
- The number of latent factors
- The number of iterations

3 Background

3.1 Recommendation Systems

A recommendation system aims at showing items of interest to a user, considering the context of where the items are being shown and to whom they are being shown (Alag, 2008).

Figure 1, depicts inputs and outputs of a common recommendation system.

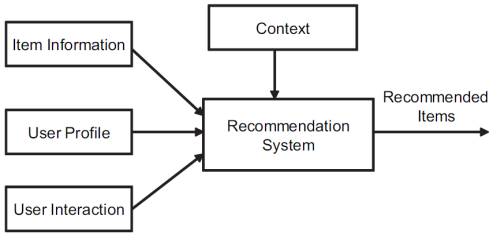


Figure 1: Inputs and Outputs of a Recommendation Engine

(Alag, 2008)

In Adomavicius (2005) three approaches for building a recommendation system are presented:

- Content-based Recommendation: Items similar to the ones he/she has preferred in the past, are recommended to the user.
- Collaborative Recommendation: Items that people with similar tastes and preferences liked in the past, are recommended to the user.
  - Collaborative Deep Learning: It is a recent kind of collaborative filtering using deep learning models Wang et al. (2014).
- Hybrid Approach: Recommendations are made using a combination of Content-based and Collaborative Recommendation methods.

3.2 Alternating Least Squares (ALS)

Alternating Least Squares (Low et al., 2012; Zhou et al., 2008; Koren et al., 2009) is an algorithm within the collaborative filtering paradigm. Input of ALS (in Figure 2) is a sparse user by items matrix R containing the rating of each user. The algorithm iteratively computes a low-rank matrix factorization $R = U \times V$ where U and V are d dimensional matrices. The loss function is defined as the squared error(Zhou et al., 2008), where the learning objective is to minimize the sum of the squared errors 1 between values predicted and real values of rantings.

$$(\hat{U}, \hat{V}) = \underset{U,V}{\operatorname{argmin}} \sum i, j \in R(r_{ij} - v_i^T u_j)^2 \quad (1)$$

Complexity and cost depend on the magnitude of the hidden variables d.

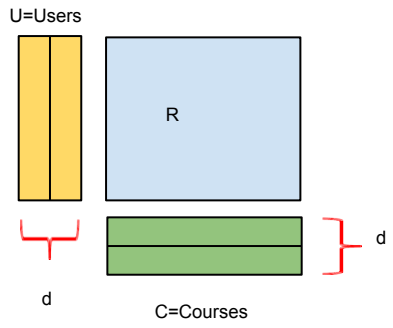


Figure 2: Matrix factorization of ALS

The ALS algorithm is computationally expensive, every iteration runs on $O(d^{-1}[Nr + (m +$

$n)r^2] + r^3)$, where m is length of items, and n is length of users (Schelter et al., 2013; Gemulla et al., 2011).

3.3 Alternating Least Squares on GraphLab

According to (Low et al., 2012; Gonzalez et al., 2012) ALS in GraphLab is implemented by using a bipartite two colorable graph and a chromatic synchronous engine with an edge consistency model for serializability.

Each vertex of the graph has a latent factor attached, that denotes a user or an item. Thus, they are linked to a column or a row in the matrix of ratings R. Each edge of the graph contains entry data (rating values), and the most recent error estimated by the algorithm. The goal of ALS algorithm is to discover values of latent parameters, such that non-zero entries in R can be predicted by the dot product of the row and column latent factor. ALS algorithm for GraphLab is implemented in the Gather-Apply-Scatter abstraction. ALS update considers adjacent vertices as X values and edges as observed y values, and then updates the current vertex value as a weight w:

$$y = X * w + noise \tag{2}$$

that is accomplished using the following equation:

$$w = inv(X' * X) * (X' * y) \tag{3}$$

In the Gather-Apply-Scatter model, the update is done as follows:

- Gather: it returns the tuple $(X' * X, X' * y)$
- Apply: it solves $inv(X' * X) * (X' * y)$
- Scatter: it schedules the update of adjacent vertices if this vertex has changed and the edge is not well predicted.

3.4 Alternating Least Squares on MapReduce

In Xianfeng Yang (2014; Zhou et al. (2008) MapReduce implementation is comprised by four tasks as shown in Figure 3. Each item in dataset is denoted as a triple (u, j, r). u denotes user, j is the label of item and r denotes corresponding rating. In the U-Update step, item matrix V is used as input and is sent to cluster nodes. Then, training rating R is used to compute user matrix U, including inputs as lambda parameter λ to regularization, number of latent factors. V-Update does the same as U-Update step, but its input is not an item matrix. On the contrary, it is a user matrix computed in U-Update step. Once U and V are learned, we can compute RMSE values using test dataset and estimated rating $\hat{r}$. So the Parallel ALS algorithm with Weighted--Regularization is as follows (Zhou et al., 2008): The objective function in

Algorithm 1 Alternating Least Square(ALS) with algorithm

1: Initialize V with random values between 0 and 1
2: Hold V constants, and solve U by minimizing the objective function.
3: Hold U constants, and solve M by minimizing the objective function.
4: **repeat** from step 2 and 3 until objective function converge.

1 is obtained from equation 4, which is just linear regression with lambda regularization(λ), to avoid overfits it penalize large parameters.

4 Proposal

In this paper we evaluate several parameter configurations (lambda, number of latent factor, number of iterations) for ALS algorithm over GraphLab and MapReduce. Our aim is to obtain the best performance, over clusters of two and four machines, for the Movielens Dataset, and NetFlix Dataset (further details will be given in the next section). We evaluate performance according to RMSE and execution time values.

In order to implement ALS algorithm under the MapReduce Paradigm, the Mahout [1] API has been used. ALS algorithm for MapReduce (Zhou et al., 2008) is shown in 3. User and movie factors have been computed using equation 4. where n_{ui} and n_{vj} are the numbers of ratings of user i and item j respectively. When objective function showed in equation 4 does not change after further iterations, we attain the final step. Output is the predicted rating for each user/item pair.

[1] http://mahout.apache.org/

$$f(U,V) = \sum_{i,j \in I} (r_{ij} - u_i^T v_j)^2 + \lambda (\sum_i n_{ui} \| u_i \|^2 + \sum_j n_{vj} \| v_j \|^2) \quad (4)$$

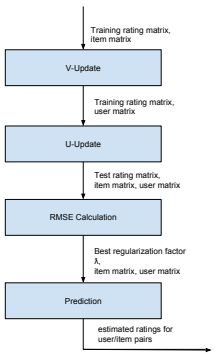


Figure 3: MapReduce ALS algorithms proposed (Zhou et al., 2008; Xianfeng Yang, 2014)

In order to evaluate ALS algorithm under GraphLab, the GraphLab API (Low et al., 2010) has been used. ALS algorithm for GraphLab (Low et al., 2012) is shown in Figure 4. User and movie factors have been computed using equation 5.

$$f[i] = \underset{w \in R^d}{\operatorname{argmin}} \sum_{j \in Neighbors(i)} (r_{ij} - w^T f[j]) \quad (5)$$

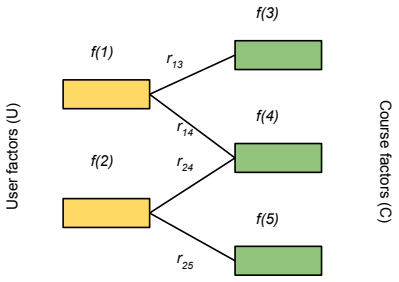


Figure 4: Matrix factorization of ALS using GraphLab

4.1 Movielens Dataset

MovieLens is a Web collaborative site that manages a recommender system for movies. This recommender system is based on a collaborative filtering algorithm developed by the GroupLens research group. The dataset is comprised by 6040 users, 3952 items and 100209 ratings for training. The data structure is: *user*, *item*, *rating*.

4.2 Netflix Dataset

We are using the Small Netflix Dataset. It is also a data-set for movie recommendation, it has 95526 users, 3561 items and 3298163 ratings. The structure of the data-set is: *user*, *item*, *rating*.

4.3 GraphLab Configuration

Setup of the GraphLab cluster is as follows. Two machines, one working as the master and the other as the worker node. The master machine operating system is Ubuntu 14.04, and its processor is Intel Core i3 CPU M 330@2.13GHzx4. The worker machine operating system is Ubuntu 13.10 of 64-bit, and its processor is Intel Core i3-2350M @2.30GHzx4. The cluster was configured using MPI(Message Passing Interface).

4.4 MapReduce Configuration

The setup is as follows. Four machines, three worker nodes and one master. The master machine operating system is Ubuntu 13.10 of 64-bit, and its processor is Intel Core i3-2350M @2.30GHzx4. Table.1 shows the configuration of the worker machines.

The cluster was configured using Hadoop, and

Machine	Operating System	Processor
1	Ubuntu 14.04	Intel Core i3 CPU M 330@2.13GHzx4
2	Ubuntu 13.10	Intel Core i3-2350M @2.30GHzx4
3	Ubuntu 14.04	Intel Core i7-4700MQ @2.40GHzx8

Table 1: Worker Machines Configuration

the HDFS(Hadoop Distributed File System). The ALS(Alternating Least Squares) algorithm implementation was taken from Mahout Library.

5 Experimental Results

This section shows experimental results conducted on MovieLens data set aforementioned. Experimental setting parameters are described in

Parameters	Value
Lambda	0.01 - 0.09
# Latent factors	10-50
# Iterations	2-30

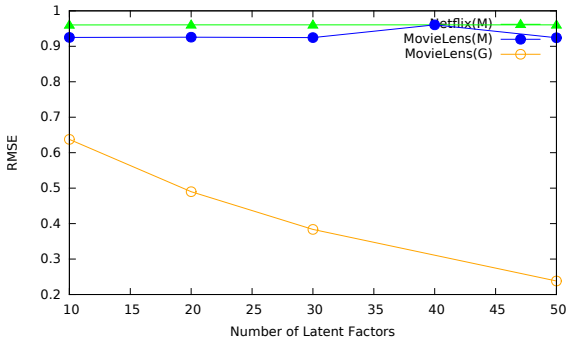
Table 2: Parameters used for ALS algorithms

Table 2. Latent factors have been increased for each test in 10 step size, Lambda has been increased in 0.01 step size, and Number of iterations in 1 step size. Results are showed in Figures 5,6,7.

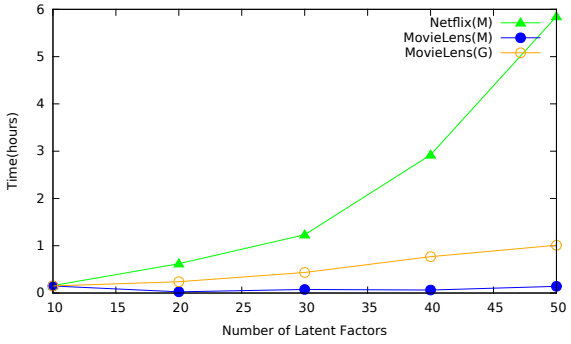
In Figure 5a, RMSE values for MapReduce do not change even if we increase the number of latent factors thus, RMSE values on MapReduce are independent on the number of latent factors. RMSE for MapReduce converges around 0.95. Conversely, RMSE values for GraphLab decreases while the number of latent factors increases. When the number of latent factors was 50, RMSE value reaches around 0.25. However, GraphLab spends more time than MapReduce, Figure 5b depicts MapReduce times almost as an horizontal line for MovieLens dataset, the line of execution time for Netflix dataset is much steeper. Between Graphlab and MapReduce lines representing Movilens dataset execution, Graphlab line is more pronounced.

Figure 6a depicts GraphLab and MapReduce performance according the Lambda parameter. While Lambda increases, RMSE decreases accordingly, i.e., if a greater value of Lambda is used then algorithm accuracy tends to be better. We also notice that Graphlab has lower values of RMSE compared to MapReduce. GraphLab RMSE values are around 0.5, and MapReduce RMSE values are around 1. Figure 6b illustrates a better execution time of MapReduce compare to GraphLab over Movielens dataset. However, now the execution time for GraphLab decreases, while the value of Lambda increases. Figure 6b also shows that It takes longer to process the data from netflix than Movielens.

In Figure 7a we notice that the value of RMSE is almost invariant to the increase of iterations for MapReduce execution, given that the number of iterations are small,nevertheless we notice clearly that RMSE value for GraphLab decreases as the number of iterations increases. RMSE value for Graphlab converges around 0.55. Figure 7b shows that MapReduce execution time over Movielens dataset is good, however it increases a lot for Netflix dataset. Graphlab execution time increases as the number of iterations grows.

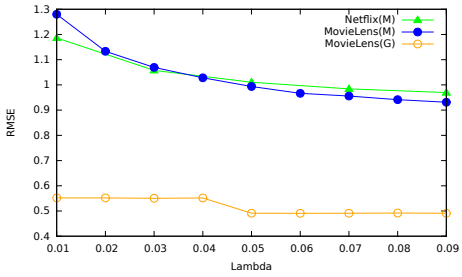


(a) Number of latent factors Vs. RMSE

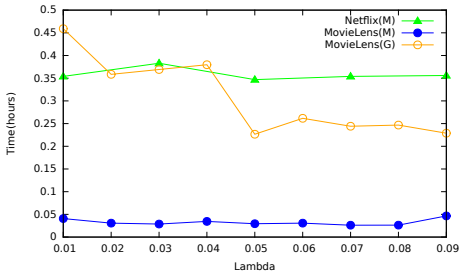


(b) Number of latent factors Vs. Time

Figure 5: Performance of MapReduce and GraphLab when number of features in ALS algorithms is increased.

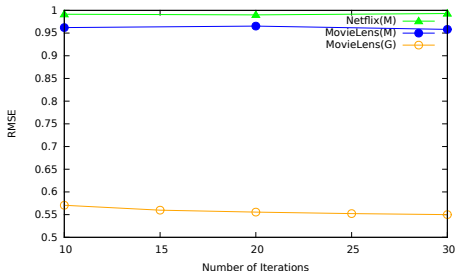


(a) Lambda Vs. RMSE

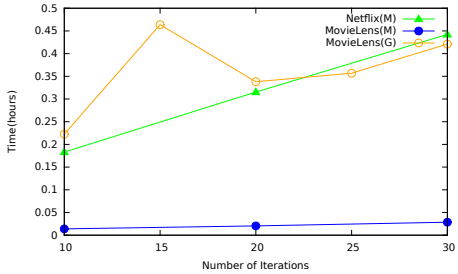


(b) Lambda Vs. Time

Figure 6: Performance of MapReduce and GraphLab when Lambda values in ALS algorithms are increased.



(a) Number of Iterations Vs. RMSE



(b) Number of Iterations Vs. RMSE

Figure 7: Performance of MapReduce and GraphLab when the number of iterations in ALS algorithms is increased.

6 Conclusion

We evaluated the Alternating Least Squares (ALS) algorithm, a parallel collaborative filtering in both GraphLab and MapReduce frameworks. Experiments were run over the MovieLens and Netflix datasets. The RMSE between MapReduce execution in NetFlix dataset and Movielens dataset in all the experiments was similar, but the execution time was longer in Netflix dataset.
Looking at the executions over Moviliens dataset, we can say, that even though GraphLab only ran in two machines and MapReduce in 4 machines, the first one outperformed the second one in RMSE. Considering lambda value variation, Figure.6a,the number of iterations Figure.7a, and the number of latent factors Figure.5a, GraphLab performed better (RMSE) than MapReduce. In all previous three cases MapReduce was faster than GraphLab, obviously by the difference between the number of machines in their configuration.

Thus, when scalability and distribution are evaluated, MapReduce performs better, because ALS does not require data dependency for computing. Moreover, it took less execution time when more latent factors were added. In this work we only used two nodes, however GraphLab demonstrated best results with few nodes.

In conclusion, GraphLab performed better when RMSE was considered but, there are open issues with shared-memory. GraphLab is also better for computing recommendations in real time. However, for more sophisticated computations MapReduce performs better so far as to an offline environment and all data is used.

References

Tuzhilin A. Adomavicius, G. 2005. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17:734 – 749.

Satnam Alag. 2008. *Collective Intelligence in Action*.

Toon De Pessemier, Kris Vanhecke, Simon Dooms, and Luc Martens. 2011. Content-based recommendation algorithms on the hadoop mapreduce framework. In *7th international conference on Web Information Systems and Technologies, Proceedings*, pages 237–240. Ghent University, Department of Information technology.

Jeffrey Dean and Sanjay Ghemawat. 2008. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January.

Rainer Gemulla, Erik Nijkamp, Peter J Haas, and Yannis Sismanis. 2011. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 69–77. ACM.

Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 17–30, Berkeley, CA, USA. USENIX Association.

Y Koren, R Bell, and C Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. 2010. Graphlab: A new parallel framework for machine learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, July.

Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. 2012. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727.

Sebastian Schelter, Christoph Boden, Martin Schenck, Alexander Alexandrov, and Volker Markl. 2013.

Distributed matrix factorization with mapreduce using a series of broadcast-joins. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 281–284, New York, NY, USA. ACM.

Yue Shi, Martha Larson, and Alan Hanjalic. 2014. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Comput. Surv.*, 47(1):3:1–3:45, May.

Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2014. Collaborative deep learning for recommender systems. *CoRR*, abs/1409.2944.

Pengfei Liu Xianfeng Yang. 2014. Collaborative filtering recommendation using matrix factorization: A mapreduce implementation. *International Journal of Grid and Distributed Computing*.

Zhifeng Xiao and Yang Xiao. 2014. Achieving accountable mapreduce in cloud computing. *Future Gener. Comput. Syst.*, 30:1–13, January.

Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. 2008. Large-scale parallel collaborative filtering for the netflix prize. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*, AAIM '08, pages 337–348, Berlin, Heidelberg. Springer-Verlag.