

Mahout Taste's Performance Review and Some Recommendations From iletken

Taste is an open-source recommendation library in which you can create a basic recommender easily.

We appreciate and support the efforts of Apache Foundation including Taste in their Mahout project. Mahout's goal is to build scalable apache licensed machine learning libraries. It is especially a great resource for developers who are willing to take a step into recommendation and machine learning technologies. In this post, we will try to describe some key features of Mahout's Taste implementation, and also discuss some problems associated with it. Because Taste has standard textbook algorithms, we will not get into accuracy details. Therefore our main focus will be on performance. **As a recommender system provider**, we are also struggling with scalability issues and we are investigating up to what extent a Hadoop implementation can help. Therefore, we went ahead and tested Mahout's latest release in order to answer this question:

Does Taste in Mahout somehow solve scalability issues in recommender systems?

Sadly, the answer was not a Yes. Taste is a great API which is very easy to develop with, however even with the Hadoop's scalability features, Taste is very limited. Taste's original implementation is not scalable and this implementation does not fit into Hadoop's map-reduce logic naturally. So, we recommend developers to play with Mahout and learn from it, but implement their own framework and recommendation algorithms for best accuracy and performance in the real world.

After providing a detailed review of Mahout-Taste, we will be describing some of the methodologies and technologies we rely on to overcome our performance and resource management problems with iletken. We will be recommending some basic implementation tips.

Question: Parallel Collaborative Filtering?

In order to talk about scaling in recommender systems, we must elaborate on some main concepts of recommender systems and discuss which ones are parallelizable. Let's keep in mind that Collaborative Filtering is not very scalable by its nature. On the other hand, search algorithms are mostly super-scalable.

Machine Training: Many CF algorithms calculate each user's preferences by iterating through all other users' preferences as in Matrix factorization and Boltzmann machines (both are not included in Taste). In such cases, it is not possible to fully parallelize these algorithms without losing accuracy. However, some simpler CF algorithms like "slope one" use a different approach and it's easy to parallelize them.

Let's remember slope one is not as accurate as Matrix Factorization and etc... For more information, please see benchmarking section.

Pre-computation: After the system is trained, preferences for each user are computed. This is done by iterating through users' history. Pre-computation is a must for recommender systems in order to produce real-time fast recommendations. In order to achieve real time speed, users' preferences must be stored in the memory. This can be parallelized with some effort.

Recommendation: When the system is trained and all necessary computations are done, providing the recommendations is easy and scalable.

What is Mahout Taste?

Taste is an open-source recommendation library. It includes only collaborative filtering algorithms. Recently, Taste has been added to the Apache's Mahout project.

Taste has many standard collaborative filtering algorithms for developers to work on.

What is good about Taste is that it can scale training session for slope one similarity calculations.

Yet, for the reasons described below, it might not be very possible to build high-scale, accurate, dynamic and fast recommendation over simple Taste.

Problem 1: Only Collaborative Filtering?

A complete recommender system should not only use collaborative filtering algorithms. There are many reasons for that starting with the issues related to rapidly emerging content like news. The question is: How do you recommend new content that quickly loses relevance with time? CF requires an item to be rated (or hit) by a number of users in order to give a recommendation for it. This makes using CF algorithms on News difficult.

On the contrary, Iletken's award winning algorithm solves this issue by using a hybrid of content based filtering (content analysis), collaborative filtering (behavior analysis) and social relevance proximity graphs (social/trust networks).

Problem 2: Inefficient Implementation & Accuracy

Netflix dataset is considered as the most simple and reliable dataset on the recommendation field. Building a recommender over Netflix dataset, potential preference/rating of each user for every movie can be predicted. Taste, by default has a data model for Netflix dataset, which means that Taste can read the Netflix data if 12 GiB of memory is available.

On the other hand, Taste, only allows slope one to be used on Netflix data model, because the data model does not implement many features required to use other recommendation techniques. This means that one cannot compute user or item similarities, and also due to the

implementation of Taste, which will be described in the next section, it is not easy to add these features.

With Taste's slope one implementation, it is possible to predict the rating of a user for an item using slope one method, which is considered not very accurate, compared to matrix factorization, restricted Boltzmann machines, or artificial neural networks.

2.1. Accuracy:

Accuracy is essential in recommender systems. Netflix is giving the Netflix Prize (\$ one million) to any team who increases their recommendation algorithm's accuracy by 10%. For the best accuracy, understanding the recommendation method, and deeply tweaking its algorithm is a must. For instance, slope one recommender, which is the recommender used in Mahout Taste for Netflix dataset provides results between -3% (0.98 RMSE) and 2% (0.928 RMSE) by tweaking it.

So, if accuracy is important, one needs to really understand the underlying algorithms, and customize them or create new ones to fit the exact requirements.

If this is true, the next step is: If you can already understand recommender algorithms and create your own methods, you will not need Mahout's standard textbook algorithms.

And finally, this suggests that Mahout is useful for basic and not highly accurate recommendations.

To give an example, a fine tuned matrix factorization method can achieve 0.87 RMSE (8.4 % improvement). It is clear that this is **way better** than slope one.

Problem 2.a) Memory Usage (on Netflix Dataset and Data Model)

As a brief summary you can read the description below and skip this part if you are not interested:

"Taste can only read 44.500.000 ratings on a computer with 4 GiB of ram. This is %40.45 of Netflix dataset. On the other hand, as iletken, we are able to read %100 of this dataset with only ~250mb of memory. Using an additional ~100mb of memory, iletken's system completes its training and becomes eligible to make recommendations for any user. Iletken has a movie recommender demo based on this data called "Moogoo". Send us an e-mail to check it out.

Details of Taste Implementation:

In Taste,

Ratings are held in a list of Preference objects

Each element of this list is a Preference object

Each Preference object has a User object, an Item object and a Rating value

So, for each rating in Netflix, 3 objects are instantiated. This results in being able to read only 44.500.000 ratings on a 4 GB machine. This is %40.45 of Netflix dataset.

This does not include memory used during computation of training and recommending parts. It is not even possible for a developer with a simple hardware resources to start working with the Netflix dataset on Mahout Taste.

Solution: Be Efficient, Use Your Own Implementation

iletken has a Netflix recommender that does the entire job, including training, computation and recommendation only with ~350mb of ram, using advanced matrix factorization techniques with far more better accuracy than any of Taste's open-source algorithms.

Key Points of iletken's Performance Implementation:

Here are the details of our Netflix Movie Recommender implementation apart from any accuracy issues. We hope these will help developers to achieve better performance.

Keep ratings on sparse matrices, old Yale is a good place to start.

Use bit vectors for item id's (14 bits) and rating values (3 bits) for memory efficiency,

With our own sparse matrix and bit vector implementations, which are implemented from scratch by iletken for maximum performance, we managed to get these results:

For 480189 users ($480189 + 1 \times 32/8 = 1920760$ byte) + for each users each item (110 million user-item pairs, each item is stored on 16bits, $110000000 \times 16/8 = 220000000$ byte) + for each users each items rating value (110 million user-item-rating triplets, each rating is stored on ~3.2bits, $110000000 \times 3.2/8 = 44000000$ byte) = 265920760 byte = 259688.24 KiB = 253.6018 MiB

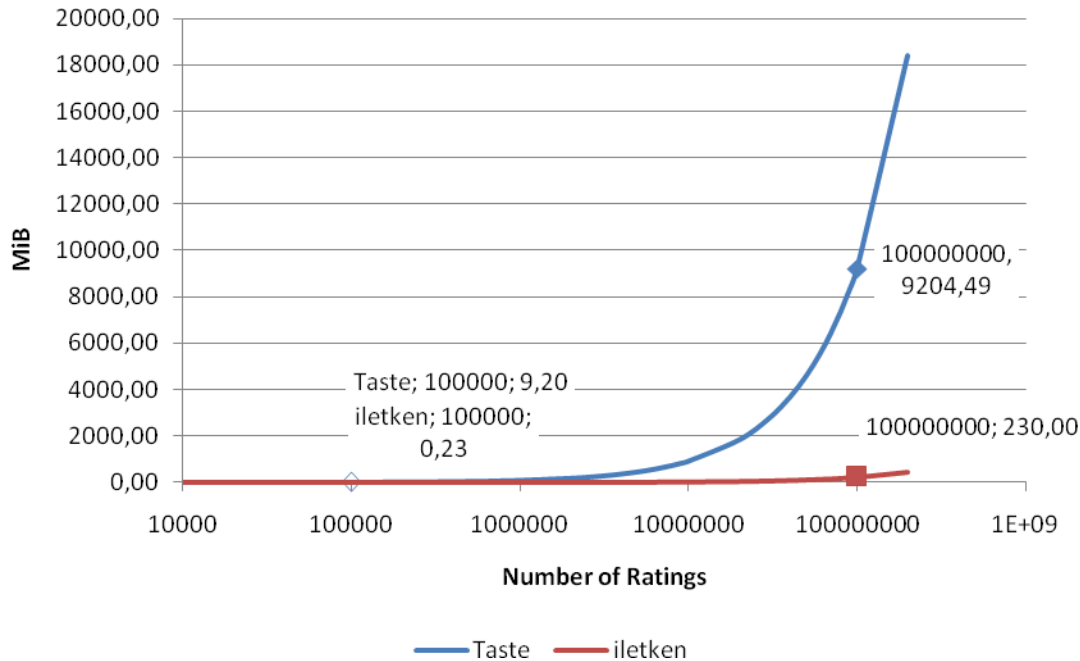


Figure 1. Comparison of memory requirement (Scaled at Log Base 10)

Problem 3: Scaling with Mahout?

In this section, we will be talking about Mahout’s Hadoop implementation and argue that it doesn’t have a very scalable implementation. If you are not interested, you can skip to the disadvantages part at the end of this section.

Mahout only uses slope one method with the scalability option, and this method computes average difference values between items for recommendations. Mahout has implemented two Hadoop jobs for computing these difference values. These jobs are run on each Hadoop node in parallel; and the end result is written on to another file presumably in HDFS. However this is not the end of the computation. When the average calculation step is completed, you will need to read this intermediate file and create a slope one recommender out of it.

However **Mahout doesn’t have a scalable recommender implementation**. You must read that entire file on every machine, and in order to produce fast results, you must read it into the memory. However, the size of the file is not small. Since it includes an average difference value for each item-item pair, it has a size in the order of $O(I^2)$. Therefore, for 20,000 items, this is roughly 4 GiB, and for 50,000 items it becomes a whopping amount of 28 GiB (Figure 2). Actually this value is larger since Mahout writes items and difference values as serialized objects. Here it is assumed that 96 bits are enough to hold an item-item-rating triplet.

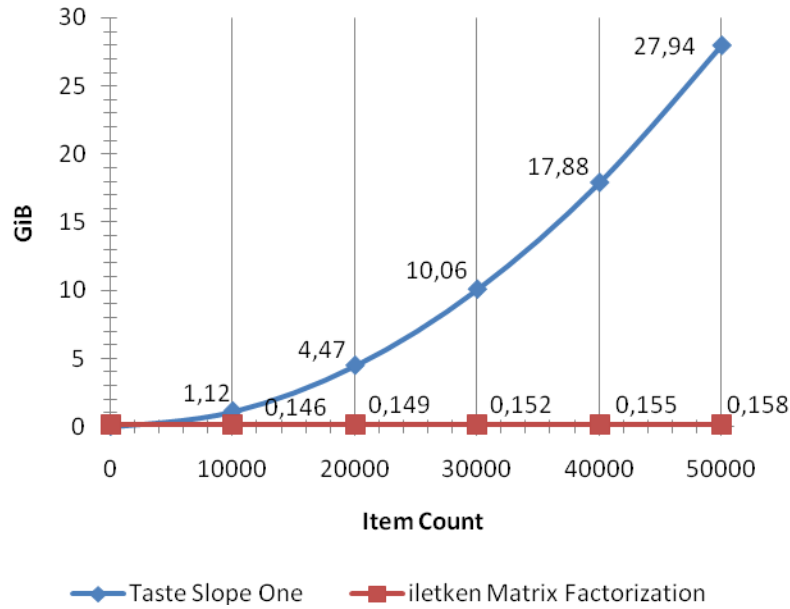


Figure 2. Comparison of memory requirements as item count changes

Therefore, to create a recommender for 50000 items that produces fast results, you need to have a server with at least 30 GiB of ram. And still, this server would not make recommendations on the scale of milliseconds. This is because Mahout has to compute users' preference for every item (i.e. 50000 items) using users' item-rating history, and then sort by preference to recommend top K items. Without sorting, this is $O(I*P)$ where P is count of item-rating pairs in user history (Figure 3). Obviously this is not a fast way to recommend items to a user; those results have to be pre-computed.

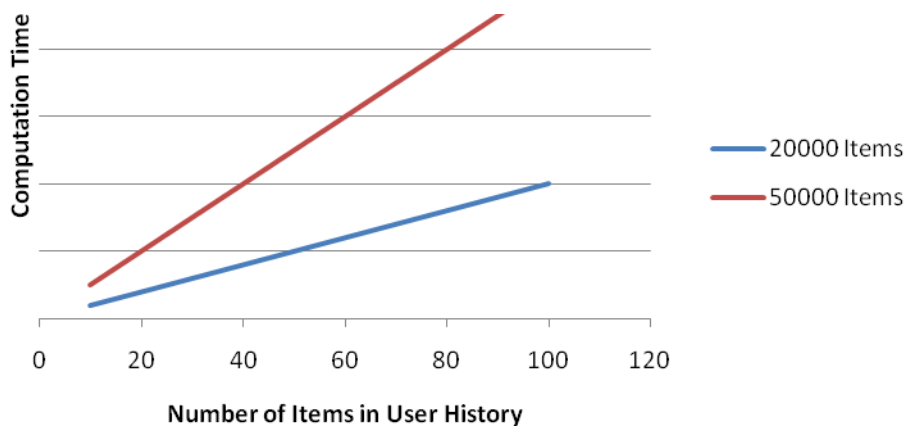


Figure 3. Computation time as number of items in user history changes

Mahout has a Hadoop job implementation just for this pre-computation. What Mahout does at this stage is to copy the 28 GiB file (for 50000 items) to each Hadoop node, instantiate a recommender at those nodes, then compute and write recommendations for each user on the node.

This has 3 disadvantages:

- 1- For 50000 items, you need to have N machines with at least 28 GiB of memory for each, where N is the number of Hadoop nodes. Considering that Hadoop promises scalability with commodity hardware, 28 GiB of memory becomes an issue.
- 2- As your items increase, memory requirements in Taste increase exponentially. Since the recommender class implementation is not scalable, this requirement cannot be solved by adding more servers in the current implementation. Adding more servers and adding more ram to each server on Hadoop is needed. This is not very scalable in our sense.
- 3- Network transfer: The whole recommender needs to be transferred to each node on every re-training. Considering Taste has a non efficient recommender implementation, this will result in numerous network transfers. For 100 nodes and 50000 items transferring ~2.7 TiB for every re-train is required (Figure 4).

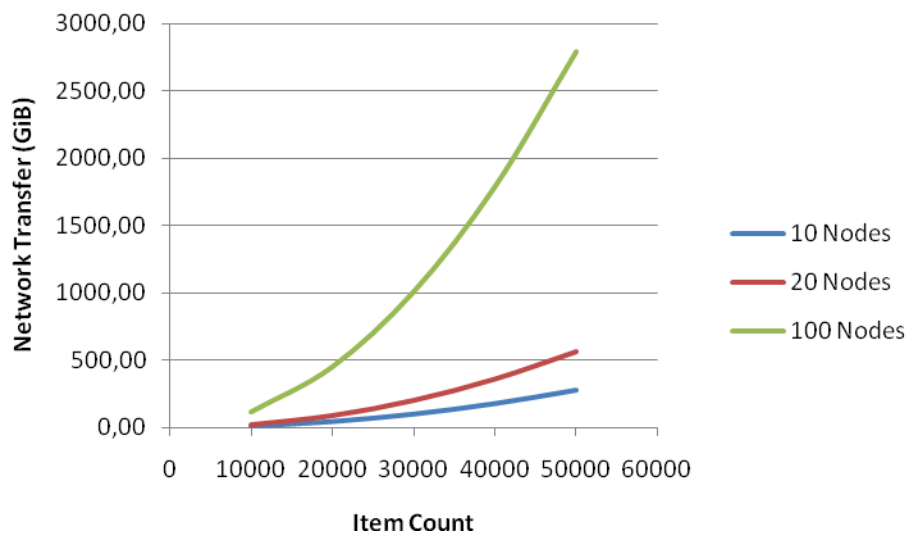


Figure 4. Network transfer as item and node count changes

To sum up: If providing recommendations is an important part of your business focus, build a custom built recommender. If not, Mahout is a good place to start.

This is the end of our Mahout review. Please don't hesitate to contact us for any reason.

Summary

Pros:

- Mahout manages to scale training session with slope one method and hadoop implementation
- Mahout is Open Source

Cons:

- Does not scale as expected
- Mahout scalability is achieved by only using a slope one method.
- Standard Slope one recommender is not very accurate (Netflix test: Success -3% (0.98 RMSE)) compared to other algorithms(not included in Mahout) such as Matrix Factorization(Netflix test: Success: 8.4% (0.87 RMSE))
- Inefficient implementation
- High memory & resource consumption
- Only Collaborative filtering
- Only standart algorithms.

iletken R&D Team

Bariş Can DAYLIK - M.Deniz OKTAR

deniz.oktar@iletken-project.com , baris.daylik@iletken-project.com

www.iletken-project.com