A Continuous Dataflow Pipeline For Low Latency Recommendations

GE WU
A CONTINUOUS DATAFLOW PIPELINE FOR LOW LATENCY RECOMMENDATIONS

Degree Project in Information and Software Systems
Submitted for the degree of Master of Science in Software Engineering of Distributed Systems

Author: Ge Wu
Examiner: Vladimir Vlassov
Supervisors: Jens Agerberg, Paris Carbone
# Table of Contents

1. Abstract .................................................................................. 4
2. Contributions ........................................................................... 4
3. Introduction .............................................................................. 5
   3.1 Background ......................................................................... 5
   3.2 Problem ............................................................................. 6
   3.3 Purpose ............................................................................... 7
   3.4 Goal and Benefit ................................................................. 7
   3.5 Methods ............................................................................. 8
   3.6 Delimitations ...................................................................... 8
   3.7 Outline ............................................................................... 8
4. Recommendation Algorithms .................................................. 10
   4.1 Content Based filtering ...................................................... 10
      4.1.1 Characteristics of content based filtering .................... 12
   4.2 Collaborative filtering ....................................................... 12
      4.2.1 Memory based collaborative filtering ......................... 13
      4.2.2 Model based collaborative filtering ............................. 16
      4.2.3 Characteristics of Collaborative Filtering .................. 18
   4.3 Algorithm Summary .......................................................... 19
5. System Architecture ................................................................. 21
6. Methods .................................................................................... 24
   6.1 Mean Absolute Error ....................................................... 24
   6.2 Training error ..................................................................... 25
   6.3 Measurements .................................................................... 25
7. Implementation .......................................................................... 27
   7.1 Tools ................................................................................ 27
1 Abstract

The goal of building recommender system is to generate personalized recommendations to users. Recommender system has great value in multiple business verticals like video on demand, news, advertising and retailing. In order to recommend to each individual, large number of personal preference data need to be collected and processed. Processing big data usually takes long time. The long delays from data entered system to results being generated makes recommender systems can only benefit returning users. This project is an attempt to build a recommender system as service with low latency, to make it applicable for more scenarios.

In this paper, different recommendation algorithms, distributed computing frameworks are studied and compared to identify the most suitable design. Experiment results reviled the logarithmical relationship between recommendation quality and training data size in collaborative filtering. By applying the finding, a low latency recommendation workflow is achieved by reduce training data size and create parallel computing partitions with minimal cost of prediction quality. In this project the calculation time is successfully limited in 3 seconds (instead of 25 in control value) while maintaining 90% of the prediction quality.

2 Contributions

- An analysis of general item recommendation methods with a focus on collaborative filtering and existing approaches.
- A low latency streaming recommender system design and implementation that scales out in a data parallel environment.
- An experimental comparison that focuses on the tradeoff between data size and recommendation accuracy.
3 **Introduction**

Some large companies today (2015) use recommender systems to improve user experience, increase sales and product engagement. Some early applications got very encouraging results, for example: “Netflix have 2/3 of movie watches from recommended items, Google News recommendation generate 38% more click through. Amazon 35% sales generated from recommendation [1].”

Recommender system like many other machine-learning systems builds on big data, and more data means higher quality but also means longer it takes to train the system. Overnight training tasks are a norm among those early adopters of machine learning systems.

Statistics shows, in 2012, average session duration on the webpage is 2.8 minutes [2] with 5.4 page views. Over the years, people’s attention span is getting shorter, as much as 17% of web page views are less than 4 seconds in 2015 [3]. For small, medium size business or certain industries like news, where new users fill up a big proportion of their visitors, 2.8 minutes or five to six page views may be the only chance to convert new users to prospects.

*Innometrics AB* is a Swedish company builds its technologies to help businesses to understand their users better. Initiative of this project by *Innometrics AB* is to study the feasibility of building a flexible, low maintenance and high performance recommender system as service on top of Innometrics Profile Cloud™ (Profile Cloud).

### 3.1 Background

The Innometrics Profile Cloud is a behavior-tracking platform. Profile Cloud collect behaviors as events from multiple online channels like web, mobile as well as Internet of Things (IoT). A profile in Profile Cloud is a
collection of events wrapped by sessions and a set of profile attributes. Each event contains an event definition id, unique event id and event data. The event definition id identifies the event type. On a website an event type can be: “product view”, “product added to cart” or “cart checkout”. Events are triggered and sent into profile cloud in real-time. Innometrics Profile Cloud connected with an integration platform, which listen and distribute events from Profile Cloud to corresponding apps. App on the integration platform can be configured to listen to one or multiple event types. If a listened event is triggered, a message with event details will be delivered to the app.

This system expected to be built as an application in the Innometrics integration platform, so it can take advantage of flexible and real-time training data. Training data will be a set of events with rating and related item. The ratings can explicitly expressed by user or visitor, or inexplicitly set by Innometrics customer via event configuration.

In this report, the training data comes from GroupLens Research [4], an open set of data from movie lens website by University of Minnesota. The data is widely used as control samples to develop and verify recommender systems.

3.2 Problem

Machine learning systems today are usually built in-house, and highly customized to certain use cases for the company. Unlike those technology giants, a lot of companies do not have recommender system because of the cost, complexity and expertise needed. There is only handful of machine learning as service providers and mostly batch. The project is aimed to provide a recommender system as service with following properties:

- Customizable scoring system
• Horizontally scalable
• Generically available for all item types
• Easy to adapt (low maintenance, cheap to run)
• Low latency (from training data to results)

Benefited by Innometrics Profile Cloud and connected integration platform, this project can be built with fully customized scoring system for each customer with low latency training data delivering.

However to have the rest of the properties we need to find:

• A recommendation algorithm that requires least maintenance.
• A system architecture that works with streaming data and scales horizontally.

3.3 Purpose

The purpose of this project is to compare recommendation algorithms and software to build a scalable, low latency recommender system. This written material will illustrate the characteristics of recommendation algorithms, scalable distributed computing frameworks, some implementation details and discussion of the results. The result of this project should able to answer if the requirement is possible to meet, and with what trade off.

3.4 Goal and Benefit

The goal of this project is to find and draft out a plan for Innometrics to implement a recommender system; develop a prototype application with properties mentioned previously.
The system design, results and result evaluations will benefit Innometrics as well as other organizations and individuals who want to build a similar system.

3.5 Methods

In this project, quantitative research methodology is used combined with theory study. Experiments and measurements are focused on making large amount of recommendation calculations and visualize the relationship between calculation time, prediction quality and cluster size. By observing the relationship we should able to answer: Is the relationship between speed and quality predictable using the algorithm decided? If it is predictable, is there is an optimal point between them? Can it scales out by scale the cluster?

3.6 Delimitations

Limited by time and agreement with Innometrics customer, live data cannot be used in this study, it limits the possibility of result verification, and some results may only be understood on theoretical level.

3.7 Outline

This report includes content as following:

- Theory study of different forms of recommendation algorithms. Discuss the differences between algorithms regarding the problem the project wants to resolve.
- System design and implementation based on the characteristic of the selected algorithm.
• Results and findings relate to the goal of this project.
• Finally, the conclusion and future works.
4 Recommendation Algorithms

Machine learning algorithms dictate many facets of the system, most importantly: what is the content of training data and what are the key factors decide quality and speed.

Recommendation in machine learning can be seen as filter out irrelevant information for the querying user. Mainly there are two designs of recommendation algorithms: Content based filtering and collaborative filtering. Both content based filtering and collaborative filtering are based on how human beings would make recommendations. This part of the paper discusses the differences of the two approaches and compares the pros and cons to find one most suitable to achieve the goal.

4.1 Content Based filtering

In content based filtering, the recommendations are given based on a user’s historic rating and description of the contents [5]. In another word: if you like TV A and B, and they are both full HD TVs, then you will probably like TV C, because it is also a full HD TV. In practice, content-based filtering needs to keep historic ratings for each user and have features mapped out to each item. The recommendation is based on preference expressed previously by the user.

A typical content based recommender system needs two data sources: content oriented data and user oriented data. Content oriented data contains features and values to each feature for every piece of content. User oriented data includes ratings expressed by user to some of the contents. In those two kinds of data, content features usually come from product specification sheet, it needs an extra external system to generate and import into recommender system. Less human intervention and less sub system is a huge advantage in machine learning, it means less cost for maintenance. One of the effort to automate generation of feature data
comes from an early approach on content based filtering. Content based filtering originated from an attempt to recommend text-based items, like news or webpages. In text-based content, terms are considered as features. Value or weight of the feature is relevancy of the term to the document also known as vector space representation tf*idf: term frequency times inverse document frequency.

\[
tf * idf = w(t, f) = \frac{tf_{t,d}\log \left( \frac{N}{df_t} \right)}{\sqrt{\sum_i (tf_{tid})^2 \log \left( \frac{N}{df_t} \right)^2}}
\]

Equation 1 Vector Space Representation

Equation 1 is how vector space representation is calculated [6]. \(tf_{t,d}\) is frequency of term \(t\) in document \(d\), \(N\) is number of documents in the collection, and \(df_t\) is number of documents contains term \(t\). The vector space representation can be used to calculate feature weight in text-based items without human intervention. The biggest drawback for this model is that it removes the context from document, for example it is easy to tell a TV review is highly related to black level and design but it is hard to tell which one is good or bad. Other than text-based items, a collection of features and their values are still need to be collected by the one who use this system.

An ideal feature set for a content-based recommender system should contain all the features leads to user’s rating. The effort to generate features is helpful when dealing with text-based items, but it is not very generic. Things influence user’s preference can include a lot subjective factors than terms or specification of the product. Not all the influence can be obtained, there is no content-based system can tell if there are enough features being considered.
4.1.1 Characteristics of content based filtering

- Training data:
  - Per user preference ratings
  - Content features
- Result quality depends on:
  - Number of ratings expressed by the user.
  - Completeness of content features.
- Scalability and speed:
  - Users are independent from each other, and can be calculated individually.
  - Since it can be parallelized per user, calculation can be easily distributed on cluster and speedup should increase linearly with cluster size.
- Cold start issue:
  - New user will have limited data points, which leads to poor recommendation.
  - New item (assume comes with all attribute values) can related to existing items therefore no cold start issue, can be recommended automatically.

4.2 Collaborative filtering

“Collaborative filtering methods are based on collecting and analyzing a large amount of information on users’ behaviors, activities or preferences and predicting what users will like based on their similarity to other users.” [7] In common word, this is to say: based on previous ratings, there is a group of people like the same music as you do, therefore you probably sharing the same taste and the songs been liked may share some similar qualities. When they also highly rated another song, or the song also has those qualities, you may like it too.
Collaborative filtering takes in user oriented data and user oriented data only, because the only thing to learn is to find out whom are the one having the similar taste.

Collaborative filtering algorithms can be very different on the implementation but usually falls into one out of two subsets: Memory based and Model based. Memory based algorithms are looking for people having the same taste and Model based ones are to find those shared qualities in items based on ratings. The third alternative is to combine the two to create a hybrid solution, but they are basically extension of what will be covered here.

4.2.1 Memory based collaborative filtering

Memory based algorithms are based on user-to-user or item-to-item relationship, they are called item based collaborative filtering or user based collaborative filtering. There three steps to give recommendations using memory-based algorithms:

- Similarity matrix

The first step to make recommender possible is to quantify the relationships between users or items.

There are several different ways to quantify similarity of two users or items, most well-known ones include: Pearson Correlation, Euclidean Distance and the Cosine Distance [8].

Usually similarity of two users or items represent by float value between -1 to 1. Similarity 1 means two users are indefinitely similar to the other one and -1 means they are negatively related (fully opposing pairs: one will dislike every item the other one likes), and 0 means not
related. Both positive and negative related users and items have high influence to the rating prediction of each other.

Take Pearson correlation for example:

$$w_{u,v} = \frac{\sum_{i \in I} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{v})}{\sqrt{\sum_{i \in I} (r_{u,i} - \bar{r}_u)^2 \sum_{i \in I} (r_{v,i} - \bar{v})^2}}$$

Equation 2 User-User Pearson Correlation

Equation 2 is an example of how similarity is quantified between user $u$ and $v$ using Pearson correlation. In Equation 2, $r_{u,i} - \bar{r}_u$ is used to leverage people’s tendency of average ratings. Weight for each item $i$ in collection $I \in I_u \cap I_v$ is the product of $r_{u,i} - \bar{r}_u$ and $r_{v,i} - \bar{v}$, the sum of weight for each item is the weight between user $i$ and $v$. Dividing weight between user $u$ and $v$ by the absolute value results in normalize the final result to a value between -1 to 1.

Similarly for item to item algorithm:

$$w_{i,j} = \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_i)(r_{u,j} - \bar{r}_j)}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{r}_i)^2 \sum_{i \in I} (r_{u,j} - \bar{r}_j)^2}}$$

Equation 3 Item-Item Pearson Correlation

The Equation 3 takes each user $u$ in all the users $U$ into consideration for each two items $i$ and $j$. Other study suggests: in item based collaborative filtering, the cosine similarity is more effective and popular [9].

**Neighbors**

The process looking for similar users or items is also known as find neighbors. Usually a recommender system has a stable number of items but growing number of users. Taking everything into recommendation
consideration is not scalable and unrelated users or items will decrease efficiency [10]. There are mainly two common practices to limit number of neighbors:

• **Top N Neighbors**
  
  As the name suggests, “top N” also known as kNN-k Nearest Neighbors are k most related users or items will be taken into consideration regardless how related they are. The advantage of this method is that users will more likely to have enough neighbors to make recommendations. The drawback for “top N” is: when none of the neighbors is similar enough, recommendation quality get poor accuracy. This can happen when there is insufficient data, bad data quality or user is new.

• **Threshold filtering**
  
  Threshold filtering sets bar to qualify all users above certain similarity as neighbors. The result of this method is that not all the users will have enough neighbors to make recommendations, especially when users are scattered. Overtime, this method will have more qualified neighbors, since it does not limit number of neighbors, number of neighbors can grow indefinitely. Neighbors will contain all the ones within the quality range and lower quality neighborhood will bring down accuracy.

A threshold with Top N approach can combine the advantages from both approaches, where only the top N passed threshold neighbors will take into calculation.

• **Rating prediction**
  
  With previous result of similarity matrix, to recommend to user u is to calculate top rated items by other users among neighbors but not rated by
A common way to estimate rating that user $u$ would give item $i$ is to calculate the weighted average of the rating that other users given in the neighborhood [11].

$$\hat{r}_{u,i} = \bar{r}_u + \frac{\sum_{v \in N(u)} W_{u,v} (r_{v,i} - \bar{r}_v)}{\sum_{v \in N(u)} |W_{u,v}|}$$

Equation 4 User-User rating prediction

$$\hat{r}_{u,i} = \frac{\sum_{v \in N(u)} W_{u,v} r_{v,i}}{\sum_{v \in N(u)} |W_{u,v}|}$$

Equation 5 Item-Item rating prediction

Equation 4 and Equation 5 are used in user to user and item to item rating prediction of user $u$ and item $i$ respectively. In those two equations, prediction $\hat{r}_{u,i}$, is calculated using weight between user $u$ and user $v$: $w_{u,v}$ and $v$’s rating to item $i$: $r_{v,i}$, where user $v$ belongs to $u$’s neighborhood $N(u)$ calculated previously. The difference between the two algorithms is the consideration of $u$’s average rating: item to item prediction is a simple average of weighted neighborhood ratings, because the prediction is not user specific.

4.2.2 Model based collaborative filtering

Early approaches in model based collaborative filtering consider user’s rating vector as feature vector and turn the recommendation problem becomes a classification problem [12].

A more recent approach is latent factor and matrix factorization models [12]. In this model, it assumes the influence of underlying features between similar users and items which also called “latent factors”. Because there is no explicit feature or feature value, the advantage of latent factors model including: higher accuracy and better incorporation with inexplicit
Some of the most successful latent factor models are based on matrix factorization. Matrix factorization makes recommendation when there is a high correlation on both user and item’s latent factors. Assume user to item interactions is the inner product of latent factors pace of dimensionality $f$. Item $i$ and user $u$’s latent factor vectors are $q_i \in \mathbb{R}^f$ and $p_u \in \mathbb{R}^f$ respectively. Product of $q_i$ and $p_u$ reflects the rating of user $u$ to item $i$.

$$ r_{u,i} = q_i^T p_u $$

Equation 6

Equation 6 calculates predicted rating of user $u$ to item $i$, where $q_i^T p_u$ is the product of $q_i$ and $p_u$. Since the predicted ratings are based on latent factors, breakdown raw data into feature vector is a major challenge.

A traditional way of mapping users and items into $\mathbb{R}^f$ is to use SVD: single value decomposition. Because large number of $u,i$ pairs are unknown and needs prediction, study suggests that it is better to work with known $u,i$ pairs instead of imputation, and learn $q_i$ and $p_u$ vectors with least squared error:

$$ \min_{q^*,p^*} \sum_{(u,i) \in K} (r_{u,i} - q_i^T p_u)^2 + \lambda (\|q_i\|^2 + \|p_u\|^2) $$

Equation 7

The additional $\lambda (\|q_i\|^2 + \|p_u\|^2)$ is introduced here to avoid the effect of over fitting, where $\lambda$ should be determined from cross validation. There are two ways to calculate Equation 7 two unknown variables $q_i$ and $p_u$: stochastic gradient descent and alternating least squares.

**Stochastic gradient descent**
The stochastic gradient decent optimizes Equation 7 by looping through each training set and the system calculates predicted rating: $r_{u,i}$ and error: $e_{u,i}$.

$$e_{u,i} \overset{\text{def}}{=} r_{u,i} - q_i^T p_u$$

Equation 8

Then change both $q_i$ and $p_u$ proportionally with $\gamma$ to lower the error.

$$q_i \leftarrow q_i + \gamma \times (e_{u,i} \times p_u - \lambda \times q_i)$$

$$p_u \leftarrow p_u + \gamma \times (e_{u,i} \times q_i - \lambda \times p_u)$$

Equation 9

**Alternating least squares (ALS)**

Unlike stochastic gradient decent, for each round, alternative least squares fixes one set at a time $q_i$ or $p_u$, and calculate the least square for the other set. This way of calculation means, when calculate $q_i$, $p_u$ is the only variable, and vice versa. Although ALS is more complex and slower to calculate, one big advantage is the calculation can be optimized for parallel computing.

### 4.2.3 Characteristics of Collaborative Filtering

- **Training data:**
  - User preference ratings
- **Result quality depends on:**
  - The amount of similar users or items.
- **Scalability and speed:**
  - New ratings will influence on similarity between users or items, changes in neighborhood affects rating prediction, therefore entire model needs to be re-trained.
o Larger the training set meaning longer it takes to train.
  Speedup can be achieved by reduce training set. Reducing training set means lower the chance to find similar user and recommendation quality.

- Cold start issue:
  o New user or item will have limited data points, which leads to poor recommendations.
  o New item cannot be recommended before it is rated first.

### 4.3 Algorithm Summary

Both content based and collaborative filtering algorithms are depending on user ratings either explicitly or inexplicitly and both share the cold start issue for new users. There are several differences between the two approaches:

- The despite the effort of vector space model to generate features, content base filtering needs extra maintenances on feature dataset makes it more expensive to implement.
- Quality of content based filtering is relying on the quality of content features, where collaborative filtering is not.
- Content based filtering is easier to scale compare to collaborative filtering since user interest profile is independent from each other.
- Content based filtering does not have cold start issue on new items. Given that feature values are provided with new contents.

In real world recommender system, usually the number of items and item features are finite, where the number of users is not. Users in content based filtering algorithms are independent from each other. It is easier to implement content-based algorithms onto a distributed system. The
downside of content-based recommender is the extra requirement of item features dataset, which increased cost upfront.

On the other hand, collaborative filtering is highly automated. The only input in collaborative filtering is the user ratings. The drawback of collaborative filtering is its scalability and cold start issue for the new items.

Content base filtering can be an option for low latency recommendation, because it is user independent, but it means more expensive for customer with non text based contents. For some customers, the cheaper and more generic collaborative filtering maybe a better choice. It refine the challenge to this project: How to resolve the scalability issue of collaborative filtering?
5 System Architecture

When designing the overall system structure, it is almost guaranteed to have trade off to achieve low latency. The aim for the system architecture is to have the best of both streaming and batch world.

Netflix is one of a few companies using recommendation, more specifically collaborative filtering in a large scale. Netflix recommender system is designed as following diagram Figure 1 [15]:

Figure 1 Netflix recommender system workflow

The system they have has a 3 layers approach. Training data from user goes into each layer, where online layer going to respond first probably with less accurate answer and offline layer is going to respond last with most precise data.
Having recommendation served by multiple layers to provide speed and eventual quality is very similar to a design originated by the creator of Apache Storm: Nathan Marz. A low latency architecture that is easy to maintain and scale as known as Lambda Architecture [16].

Figure 2 Lamda Architecture

Figure 2 is a high level overview of lambda architecture by Nathan Marz. Compare to the Netflix recommendation architecture, the batch layer matches to the “offline” and “nearline” layer, and speed layer represent “online” layer in Figure 1.

Batch layer can be built by existing technologies like Apache Spark MLlib or Mahout on Apache Hadoop etc. Building batch layer is not included in this paper because the focus for this paper is on low latency recommendation or speed layer.

Collaborative filtering is good for recommend different item types but it tend to use a lot of resources. The nature of collaborative filtering is to recommend based on similarity between users or items, therefore it is important to keep a large number of user behaviors on fast access storage preferably in memory. Among computer hardware, memory is one of the
elements in computer system that is expensive to scale vertically. To run collaborative filtering and scalable, it requires expanding the memory usage to other machines in the same cluster.

One way to scale horizontally is to split users into several isolated partitions, each partition will eventually have large enough user base to find similar ones. Even each partition only gets part of the data, but the amount of data in the partition can still grow infinitely. By limit number of user per partition expect to maintain constant and controllable calculation speed, but may decrease recommendation precision. The questions to be answered are:

1. Is the calculation time increases linearly on amount of training data?
2. What is the relationship between training data size and recommendation quality? If requirement can weight one more than the other?

By answering the previous questions we can understand the performance on speed and quality of the system.
6 Methods

To take measurement of speed and precision realistically, the prototype is developed using tools and dataflow that would be used in real world. Training data is shuffled and streamed into the system via message queue.

Speed in this system is measured between data entered and sends out from the training logic block, since the rest of the system can always be improved. The time unit here is milliseconds, if not specified otherwise.

The test data is a set of data outside training data to verify quality of recommendation. The difference between real and estimated value will be measured in form of Mean Absolute Error (MAE) and Training error.

This chapter includes explanation of performance evaluation terms and how they are calculated, what are the information being measured, and how it supports the goal.

6.1 Mean Absolute Error

Mean Absolute Error also known as MAE is one of the methods to measure prediction quality. MAE is the average of absolute distance of all the measure points between predicted and actual values.

\[
MAE = \frac{1}{|N|} \sum_{(u,i) \in N} |\hat{r}_{u,i} - r_{u,i}|
\]

Equation 10

In Equation 10, \( \hat{r}_{u,i} \) is the predicted value of user \( u \) to item \( i \), and \( r_{u,i} \) is the corresponding feedback/real value in the testing set.

In this implementation MAE is measured using 20% of total training dataset.
6.2 Training error

Like MAE, training error is also measuring the distance between estimated values and real values. Difference between MAE and training error is the verifying data source. In MAE a separate data set is selected to verify the results, but in training error training data was used instead.

The training error is easier to measure compare to MAE, because it does not need a separate round just to calculate errors, but becomes part of “Top N” calculation. In this project, both training error and MAE are measured where training error are used in large-scale measurements.

6.3 Measurements

Here are the measurements should be performed in this project in order to visualize relationship between speed, quality and training data size. The result of these measurements can answer the question how effective of each strategy to make a low latency recommender system.

Control recommendation quality and calculation time

The control value of recommendation quality measured by MAE can act as a guideline to indicate the quality lost and time gained by using the speed up strategy.

Calculation time and quality on different partition size

Relationship between calculation time and quality gives hint on what is the optimal partition size to have reasonable speedup without sacrifice too much quality compare to the control values.

Horizontal compare partition performances
The result should prove that: each partition should statistically be the same, and show the differences or similarities between partitions.

**Speed up by multi partitions**

Partition should have brought speed up on number of training dataset. The result of this can revile what is the nature of the speed up, and how much for each additional partition.

**Calculation time and quality using different number of cross references**

Cross reference should produce better quality results but the impact on the calculation time is yet to be determined.
7 Implementation

The prototype implementation of this system is based on a set of existing tools and utilities. This chapter will talk about tools, algorithm, system schematics and some implementation details.

7.1 Tools

To avoid reinventing wheels, libraries are used if possible. Here are some comparison between popular machine learning libraries, big data process platforms and how decision was made on the choice of tools. In this project, two open source projects provide most of the functionalities.

7.1.1 Machine learning library

There are several machine learning libraries available to use directly, and some of them require or optimized for some of the distributed frameworks. Here is a simple comparison:

<table>
<thead>
<tr>
<th>Library Name</th>
<th>Require/Optimized platform</th>
<th>Primary language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scikit-learn</td>
<td>Standalone</td>
<td>Python</td>
</tr>
<tr>
<td>Shogun</td>
<td>Standalone</td>
<td>C++</td>
</tr>
<tr>
<td>Accord</td>
<td>Windows</td>
<td>.Net</td>
</tr>
<tr>
<td>Mahout</td>
<td>Spark/Hadoop/Standalone</td>
<td>Java</td>
</tr>
<tr>
<td>MLlib</td>
<td>Spark/Hadoop</td>
<td>Scala</td>
</tr>
<tr>
<td>H2O</td>
<td>Hadoop</td>
<td>Java</td>
</tr>
<tr>
<td>Cloudera Oryx</td>
<td>Spark</td>
<td>Java</td>
</tr>
<tr>
<td>GoLearn</td>
<td>Standalone</td>
<td>Go</td>
</tr>
<tr>
<td>Weka</td>
<td>Standalone</td>
<td>Java</td>
</tr>
<tr>
<td>ConvNetJS</td>
<td>Standalone</td>
<td>Node.js</td>
</tr>
</tbody>
</table>

Table 1
As we can see here out of these 10 machine learning libs in Table 1: Mahout, Mlib, H2O and Oryx (highlighted ones) are optimized for distributed computing platforms, others libs will have longer path to be deeply implemented on distributed system. Picking from the ones does implement with distributed computing system, all except Mahout can run standalone and with potential to boost its performance using Hadoop. Since Hadoop requires an additional cluster of servers that we do not have for the moment, therefore Mahout made to the final decision.

Apache Mahout is a collection of ready to use machine algorithms. It is designed to help quickly build machine learning applications. In this prototype, Mahout was not setup on Hadoop, but it is possible for future work to benefit further from Hadoop for a performance boost.

7.1.2 Distributed computing framework

In recent years (2015) several jvm based distributed computing frameworks emerges and becomes the go to tools for big data processing. Each of those tools has a set of its own natures and made them more suitable for certain tasks. All the frameworks listed here can deliver high availability distributed computing across a cluster of computers.

- Hadoop
  - Batch processing by nature.
  - Minute level response time.
  - Provides YARN, HDFS, MapReduce functionalities.
- Spark
  - Batch processing by nature.
  - Spark streaming based on micro batching.
  - Second level response time.
  - Support very popular machine learning library: MLlib
• Job scheduling using entire cluster or isolated by Mesos or YARN.

• Storm
  o True streaming
  o Micro batching with Trident API with provides exactly-once property.
  o Millisecond level response time.
  o Runs on YARN interact with Hadoop and HDFS.
  o Supports DRPC with linear topology.
  o Long liv apps sharing one infrastructure, scalable by adding new supervisor nodes and rebalance.

• Flink
  o True streaming.
  o Micro batching by accumulates stream messages.
  o Millisecond level response time.
  o Runs on YARN as an application.
  o Long live operators.

• Samza
  o True streaming.
  o Sub-second response times.
  o Requires YARN and HDFS.
  o Relies on Kafka for internal messaging.

Out of those distributed computing frameworks, Hadoop is often mentioned because its contribution on distributed file system: HDFS and scheduling system: YARN. Spark and Hadoop are batch based systems even though the throughput can be higher but it almost guaranteed process windows from several seconds to several hours, which less than ideal for low latency system. Samza, Flink and Storm are true streaming frameworks but Samza is heavily rely on Hadoop, it can be a good alternative but for Innometrics, it means additional cost on cluster hardware.
Storm was picked in the end. The property made Storm stand out is scalability. Scaling is built in Storm, and it is possible to share the storm cluster form multiple apps. Scaling a Storm topology basically means run more java virtual machines on newly joined server by a process called “rebalance”. On the other hand, Flink will give all the resource to current running operator, even with Mesos or YARN the memory is fixed and made it harder to share the platform with other apps. Storm is also the tool built by Nathan Marz who brought up lambda architecture concept.

### 7.2 Algorithm

Alternating least squares (ALS) is one of the model based collaborative filtering algorithms. On top of the benefit of general model based algorithms ALS have an advantage: it can be parallelized and works better with implicit feedback [13].

Existing implementation of ALS like Apache Mahout or Spark Mllib can run ALS on Hadoop to further speed calculation up.

### 7.3 Speed up techniques

Nature of collaborative filtering needs large number of user ratings to support recommendations. As discussed earlier, in this project the approach is to take limited size of training data, trade quality to speed.

Several techniques are used to build the system with manageable partitions.
7.3.1 Partition

In order to speed up to a low latency level, the approach is partition the training data so that more training tasks can run simultaneously. Cross-referencing is a technique to improve precision, which is also built into the system.

The principle of partitioning in collaborative filtering is to make sure one user's ratings will not split into different partitions.

With help of Storm, the system uses the built in filed grouping to send ratings of one particular user to one partition and to that partition only. If cross-referencing is configured, it will send to multiple partitions to calculate recommended items on several populations and the final result takes the average of them.

Figure 3 shows how Strom’s Field grouping working on user without cross-referencing. In the figure, user A’s ratings are marked as uArX.
On top of field grouping on user, the system is equipped with cross-referencing mechanism.

Assume number of cross-referencing is two. Each rating from user will be duplicated and sends to two partitions. In Figure 4 user 1 marked with \( u1rX \) have destination partition \( A \) and \( D \), user 2 mapped to \( A \) and \( E \). Both user 1 and 2 are calculated and evaluated in 2 different populations and result register block will average out the result, which will suppose providing better result.

### 7.3.2 Micro batching

One additional user rating will not change the full picture, instead of making the effort to run re-training process on each new rating, the system will run on small batches on best effort. Each partition will re-train when batch is received. A batch will be sent to training when the batch reach to the batch size or timer expired. The timer starts when the first training data join the batch and expires after certain time period. By default the
batch size is 10,000 and timer set to 1 minute in this implementation. There are several distributed computing system supports micro batching like Apache Spark or Storm trident. Normal Storm topology is used here because the need to group by partition reference for example user id first.

7.3.3 Fix training data size
The system runs on a long living platform. Even though data will be partitioned, each partition can still grow to an un-manageable size. By introduce limitation per partition will further speed up the system but will also limit quality. The limitation is done by removing oldest data from training dataset, when new data is joining the training set.
7.4 **System Design**

The system is designed follows lambda architecture described previously. Both batch and speed layer are applied on Apache Storm. The following chapter talks about system schematics that maps to lambda architecture, how and what kind of data flows through the system.

### 7.4.1 System schematics

![Diagram of system schematics](image)

Figure 5

This system showing in Figure 5 has three high level logical components boxed in dotted lines are as lambda architecture suggested:

- **Speed layer** – block on top
  - Generates results fixed partition size, and expect less accurate result than batch layer.
  - Aiming to deliver results to serving layer with low latency.
- **Batch layer** – block on the bottom
- Only works with full training dataset.
- Work with all data.
- Deliver highest quality data, but takes more time.

- Serving layer – block on the right
  - Contains results from both speed and batch layer.
  - Delivers best results at the given moment.

A Strom topology is responsible for message processing and speed layer, Spark cluster responsible for batch calculation and a group of web apps act as serving layer. Each serving layer app is responsible for one customer, contains a data storage, which stores top N recommended items for each known user.

There are three major components in the topology:

- **Partition Bolt**
  
  Responsible for partition logic, in this case random field grouping on users and send multiple copies ratings for cross-referencing. Batch for each worker bolt is prepared according configured batch size or batch time to live.

- **CF Worker Bolt**
  
  Holds one partition of users and their ratings, calculates Top N recommendation items when user have new items in a batch. Each bolt is mapped to one partition of data for one company, for prove of concept, this is not implemented in prototype project.

- **Delivery Bolt**
  
  Top N ratings for each user from worker bolts are collected and deliver to serving layer in the delivery bolt. If cross reference is enabled, the average of results is calculated here as final result.

Training data flow into the system via a message queue and read by the spout. The spout broadcast messages to two streams one for batch layer and one for speed layer. One batch side, it will first go into persistent
storage to be ready for next batch training. Speed layer gets the same messages from spout. Messages will first go through the partition bolt, where one or more copies of the training data go into worker bolts. Finally worker bolts will send out its top N recommendations results on each user changed in the batch to the delivery bolt and deliver to serving layer. In this prototype implementation a quality evaluation bolt is also introduced to collect and measure the recommendation quality.

7.4.2 Training Data and Data model

In this project training and verification data is movie lens 1 million dataset, one million dataset. The data contains 3 columns: user id, item id and rating. The reasons that this dataset was picked:

- Open and widely used dataset.
- Reasonable data quality.

Here are the data models used in different phase of the calculation:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProfileId</td>
<td>String</td>
<td>Innometrics profile identifier. Unique per customer.</td>
</tr>
<tr>
<td>Item</td>
<td>Object</td>
<td>Flexible structure to store item to be recommended.</td>
</tr>
<tr>
<td>Preference</td>
<td>Float</td>
<td>Any number in any range to present score of item of particular user.</td>
</tr>
</tbody>
</table>

Table 2 Training data model

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserId</td>
<td>Long</td>
<td>Translated user id from profileId in the incoming data.</td>
</tr>
<tr>
<td>ItemId</td>
<td>Long</td>
<td>Translated item id from item in the incoming data, each item will have a unique id.</td>
</tr>
<tr>
<td>Preference</td>
<td>Float</td>
<td>Any number in any range to present score of item of particular user.</td>
</tr>
</tbody>
</table>
Table 3 Calculation data model

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProfileId</td>
<td>String</td>
<td>ProfileId is mapped back from user id, which is understandable identifier for the rest of the system.</td>
</tr>
<tr>
<td>ItemId</td>
<td>Long</td>
<td>Inherited from last step, item id to item mapping is stored in each serving layer application.</td>
</tr>
<tr>
<td>Preference</td>
<td>Float</td>
<td>Estimated preference to the item.</td>
</tr>
<tr>
<td>Rank</td>
<td>Integer</td>
<td>The Nth item in the top N results.</td>
</tr>
</tbody>
</table>

Table 4 Serving data model
8 Results and evaluation

In this part of this paper a series measurements was done to verify the effectiveness of the speedup techniques. A set of control value was taken and used to determine the tradeoff while aiming for higher speed.

The control MAE is measured using the same configuration as all other experiment in this project. The control MAE is calculated on 800,000 training and 200,000 test data. Control MAE is measured: \(0.7390101\) in \(24674\) milliseconds, and is the lowest MAE on traditional or batch system, in following results the control MAE is marked as black dash line.

Sampled training error is also used to indicate recommendation performance. Training error can be calculated while sending results to serving layer and it is cheaper to produce. In large scale calculation training error was used instead of MAE.

8.1 Training error versus testing error

MAE and training error are two similar way of measure prediction quality. The major difference between the two methods is the testing data and result in training error can be done during “Top N” recommendation process but MAE needs a separate process. In Figure 6, x axis is the size of training data in number of ratings, left y axis is error which is related to the MAE and training error lines and right y axis is used by training time. Dash line in the middle is the MAE of 1 million training set or the best MAE can be achieved.
The result in Figure 6 shows MAE and training error trend and training time over different training data sizes.

**MAE**

MAE is expected to decrease when there are more training data. What is interesting in the result is the MAE performance increase logarithmically instead of linearly to the training data size.

**Training error**

The training error curve likely to relate to a concept called “Overfitting”.

When the model is very complex and the data is not sufficient, the model can fit all data points perfectly, so it results in very low training error.
Figure 7 shows a model features with 6 variables matching 10 samples the equation almost hit all the dots and leave a very low training error. On the contrast when more samples join the population higher the chance that the model with limited features will not fulfill the dots, see Figure 8.

Training errors will appear to be very low when there is not enough data with relatively complex model. Very similar to over fitting issue, the cost function fits the existing data well. When querying for recommendations on an “over fitted” model, it usually returns worse results than it’s training error.

**Summary**
Training error is lower than control MAE, and have the opposite curve compare to paring MAE reading. The assumption is the training error will get indefinitely close to the control MAE. The gap between MAE and training error to control MAE closes logarithmically. In this setup, when training data size is 128,000 MAE reaches 90% of effectiveness of MAE 0.8951.

### 8.2 Training error trend on increasing partition size

![Graph showing training error trend](image)

Figure 9 shows a similar plot to Figure 8 training error versus training time over 1 million training data. This plot shows a trend that training error will get indefinitely close to the control MAE.
Applying the insight on the MAE to sample size relationship, the result on reducing time to delivery and maintain quality is shown in figure below.

Figure 10

By limiting the training data size to previously found number 128,000, the result should be 90% quality to the full data size.

In Figure 10, two training error measurements finish up 1 million training data. Measures marked with “128k” are from a limited rating size of 128,000.
Put Figure 6 into perspective, Figure 11 shows the difference it made to limit training data size. The two MAE plotting shows an expected 10% difference where time spends on those calculations is increasingly different.

**Evaluation**

Collaborative filtering quality related to missing ratings covered by neighbors. When there are more training data, there will be more neighbors and the “Top N” neighbors will have higher similarity. Neighbors by definition tend to have similar behaviors, and “Top N” neighbors statistically will eventually have the exactly same behaviors as training data size point to infinite. From 8000 ratings to 128,000 ratings training set MAE improved from ~2.02 to ~0.89, but training set from 128,000 to 1,000,000 only improved the MAE from ~0.89 to ~0.74. This result can conclude a logarithmical relationship between MAE and
training data size. By limit training data size using FIFO method can keep quality and calculation time on a balance.

8.3 Partition performance

Figure 12 shows training error trends of randomly distributed users landed on six workers. Statistically each partition in the system will reach to same status if users are evenly distributed. Figure 12 just proven that all the partitions shows same trend on quality over time.

Partition parallelize the calculation, it is expected to bring down the time for the system calculates same amount of ratings compare to non-partitioned system. It will take longer for each partition to gather a big enough population to reach an optimal number if the new ratings come in fix rate.
Figure 13
Figure 13 shows the time it takes to run amount of ratings over one, two and three partitions. As the result shows, with some system overhead, more partition generally means better parallelization end with higher calculation speed.

Evaluation

This part of measurement shows a similar path for all partitions on the prediction quality curve. Each partition has a set of users that is not shared with other partition, but they achieved same result. When each partition gets big enough population, they become statistically the same. The previous measurements hinted an optimal training data size to balance quality and speed. If data is evenly spread in $x$ partitions and given that training data comes with a steady rate, each partition will take $x$ times the amount of time to reach same level of quality compare to single partitioned. Figure 13, shows the speedup by partition technique, throughput increases by increasing number of partitions. The non-linear
relationship between number of partitions and speed up is not yet clear, it maybe relate to some calculation overhead for example networking, maintaining the training set.

Combine results from Figure 13 and Figure 12, number of partitions decides quality ramp up time, resource consumption and long term speedup. More partitions result in longer time it takes to reach optimal quality, and more resource consumption, but higher speed up in long term.

8.4 Cross reference performance

Figure 14

Figure 14 shows a measurement of training error performance and the relevant time spent for both no cross-reference and cross-reference over
two populations on three partitions. Trend line on the figure shows very insignificant performance differences on the cost of high calculation latency.

**Evaluation**

By multiply user ratings into several partitions to create cross reference is going to use more resources, and the graph in Figure 14 shows a heavy performance penalty. When partitions contains fairly low number of users, neighbors in one partition can be very different from the ones from another partition, by average results from them should have better quality. As number of users in each partition increases, neighbors of a user in all partitions will become more similar to each other.

In Figure 14, quality with cross referencing is higher, which shows the differences in different populations and benefit to average them out, but this advantage becomes less and less when training data size increases. On the other hand, training time increases linearly, because two cross reference points were taken, therefore it is expected to take twice amount of the time. Cross referencing technique in this setup may create more value when partitions are kept in a small size.
9 Conclusion

In this work, measurement reviled a characteristic of collaborative filtering: MAE performance improves logarithmically over training data size. When there are more than 128,000 ratings in the training set in this setup, training time increases proportionally while recommendation performance remains mostly the same. By limit size of the training set, it is possible to control the balance between prediction quality and training data size. Spread ratings into several partitions according user id give the possibility for the system to scale horizontally. Multiple partitions parallels the training process so multiple users can be calculated at the same time. In addition to the effort on speed up the calculation, cross reference technique is also evaluation for its potential to boost prediction quality. As a result, cross reference technique proved to generate better quality predictions on small training data size per partition, but in large amount of data, this technique takes too much resource but brings too little on quality gain.

By applying the findings found in this project, a low latency recommender system is achieved by limit the training data size and parallel partitions. Limiting training data size controls prediction quality and latency, and number of parallelize partitions decides the throughput. In this project the calculation time is successfully limited to 3 seconds (instead of 25 in control value) while maintaining 90% of the prediction quality.
10 Future works

- Smarter partition logic to replace random user distribution. The nature of collaborative filtering is to find similar users. An article “Model-based Collaborative Filtering using Refined K-Means and Genetic Algorithm” [17] suggests that by applying clustering techniques on grouping users it is more likely to achieve higher prediction quality.

- Use batch layer result to adjust or enhance speed layer calculation. This can help speed layer in many ways, for example finding the optimal limitation for partition population by provide control MAE. Merge results between speed and batch according to result quality.

- Apply real world data and AB testing preferably with explicit feedback. Although the work is done using movie lens data, but live data may have different quality level and as all the recommender system, the best way to make sure it works is through AB testing.

- Cross partition reference can work better in certain circumstances. The total time cost in this project is measured after all the references reached to evaluation bolt. There are several ways to improve the system to make cross referencing more efficient.
  - Deliver the result from each partition as soon as they are calculated and improve the result when result from other partition arrives.
  - Shrink the training data size to increase calculation speed and magnify the benefit of cross referencing.
11 Reference


