Towards a Scalable Social Recommender Engine for Online Marketplaces: The Case of Apache Solr

Emanuel Lacić
Graz University of Technology
Graz, Austria
elacic@know-center.at

Dominik Kowald
Know-Center
Graz, Austria
dkowald@know-center.at

Denis Parra
Pontificia Universidad Catolica
Santiago, Chile
dparra@ing.puc.cl

Martin Kahr
BLANC-NOIR GmbH
Graz, Austria
martin.kahr@blanc-noir.at

Christoph Trattner
Know-Center
Graz, Austria
ctrattner@know-center.at

ABSTRACT

Recent research has unveiled the importance of online social networks for improving the quality of recommenders in several domains, what has encouraged the research community to investigate ways to better exploit the social information for recommendations. However, there is a lack of work that offers details of frameworks that allow an easy integration of social data with traditional recommendation algorithms in order to yield a straight-forward and scalable implementation of new and existing systems. Furthermore, it is rare to find details of performance evaluations of recommender systems such as hardware and software specifications or benchmarking results of server loading tests.

In this paper we intend to bridge this gap by presenting the details of a social recommender engine for online marketplaces built upon the well-known search engine Apache Solr. We describe our architecture and also share implementation details to facilitate the re-use of our approach by people implementing recommender systems. In addition, we evaluate our framework from two perspectives: (a) recommendation algorithms and data sources, and (b) system performance under server stress tests. Using a dataset from the SecondLife virtual world that has both trading and social interactions, we contribute to research in social recommenders by showing how certain social features allow to improve recommendations in online marketplaces. On the platform implementation side, our evaluation results can serve as a baseline to people searching for performance references in terms of scalability, model training and testing tradeoffs, real-time server performance and the impact of model updates in a production system.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information filtering*

WWW'14 Companion, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2745-9/14/04.
http://dx.doi.org/10.1145/2567948.2579245 .

Keywords

social recommender engine; scalability; online marketplaces; Apache Solr

1. INTRODUCTION

Recommender systems aim at helping users to find relevant information in an overloaded information space [11]. Although there are well known methods (Content-based [1], Collaborative Filtering [16, 18], Matrix Factorization [10]) and libraries to implement, evaluate and extend recommenders (Apache Mahout[1], Graphlab[2], MyMediaLite[3], among others [8]), the deployment of a real-time recommender from scratch which considers a combination of algorithms and data sources, the effect of large volumes of data and hardware configuration, or the impact of model updates in the recommender performance remains unsolved or at least not publicly available for the research community. In this paper, we contribute to bridge this gap between research on recommender algorithms and system deployment by presenting in detail our approach to implement a social marketplace recommender. We describe our solution in terms of the data model that allows modularity and extensibility, as well as the system architecture that relies on the Apache Solr project to facilitate the scaling of our approach to big data.

We support our decision of using Solr on recent work that shows the strong relation between memory-based recommendation approaches and ready-to-use text analytic techniques [3]. Since Solr has already most of these techniques implemented, documented and optimized by a well established open-source community, we believe that it provides not only a good basis for a large-scale search engine but it also provides a good foundation to implement an efficient and scalable social recommender engine for online marketplaces. We appeal for Apache Solr since one might have to consider several dimensions of the data or already existing indices based on Apache Lucene, the kernel search engine Apache Solr is built upon.

To evaluate our implementation we consider diverse metrics – accuracy and ranking metrics along diversity, and user coverage – to unveil not only the performance of each recommender algorithm isolated but also to show the importance of each single feature and data source. In addition, a performance benchmarking experiment was conducted to show the scalability of our approach.

[1]http://mahout.apache.org
[2]http://graphlab.org
[3]http://www.mymedialite.net

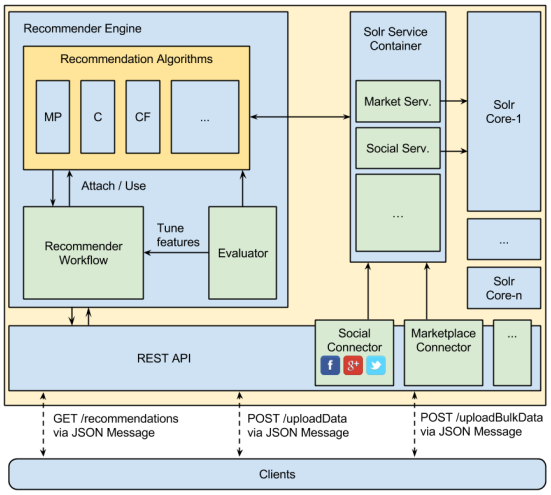

Figure 1: System architecture.

In detail, the paper is structured as follows: we begin by describing our system architecture and implementation details in Section 2. Section 3 describes the experimental setup we chose to evaluate our framework, while in Section 4 we present the results in terms of recommendation quality and also system's performance. After that, in Section 5, we discuss related work in the area and how it differs from our approach. Finally, Section 6 concludes the paper and provides an outlook to future work.

2. APPROACH

In the following sections we describe the architecture and highlight some implementation details of our approach towards a scalable social recommender engine for online marketplaces. The engine described below was implemented in Java as a joint effort with the Austrian start-up company Blanc Noir[4] and was designed in a modular way based on Apache Solr as an highly efficient data processing and storing unit.

2.1 Implementation

The overall system architecture of our framework is illustrated in Figure 1. It consists of the following four main components:

The **Recommender Engine** consists of the implemented recommender algorithms (e.g., Most Popular, Content-based, Collaborative Filtering, etc.) that can be attached to the Recommendation Workflow component. The algorithms can be called separately, in a specific sequence or combined (e.g., as a hybrid approach). This design gives our framework not only the flexibility that new algorithms can be easily implemented and instantiated (see Listing 1) but also that new recommendation workflows can be defined based on a given use case or domain.

Moreover, the Recommender Engine component contains an Evaluator that can can be used to test and tune the different algorithms (and the combinations of those) based on various evaluation metrics (see Sections 3.2 and 4.1).

The **Solr Service Container** acts as an abstraction layer for the Solr core functionalities to encapsulate the different queries and methods (e.g., facets or MoreLikeThis) into data-driven services (e.g., Marketplace or social services) that can be consumed by the recommender engine. Furthermore, this modular abstraction allows the Solr backend to be replaced by another data store or search strategy (e.g., ElasticSearch[5]) if needed.

The **Solr Cores** contain the indexed data used to generate recommendations. Each core is described by its own Solr schema that specifies a data structure and that can be extended easily by simply adding new data fields to it and calling Solr's RELOAD function. Currently, we store four types of data structures in the Solr cores: (1) user profiles, (2) item profiles, (3) user actions (e.g., purchases) and (4) social interactions (e.g., comments). New Solr cores can be added if new data structures are needed in the data model. For adding a new Solr core, a new data schema needs to be defined and registered in the solr.xml configuration file, which can be automatically done using Solr's CREATE function.

Another major reason for using Solr is its support for horizontal scaling. Since version 4.0, full automatic index distribution and searching across multiple machines (either shards or replicas) is supported. Under a scenario where the maximum capacity for handling queries per second is reached, horizontal scaling with additional replicas can be performed. On the other hand, with sharding on multiple machines, Solr supports the need to store large amounts of data distributively.

The **REST API** is the interface to our framework that a client can either use to request item recommendations, based on a specific algorithm or workflow, or to update the data model (e.g., if a user has purchased an item) via JSON messages. These JSON messages are fully configurable and let the client, for example, define user-specific data filters to, e.g., request only recommendations for a given item category.

The data updates are handled by data connectors, where each connector is responsible for a different type of data. Currently there are two connectors in the system, one for social data (e.g., Facebook, G+ and Twitter streams) and the other one for marketplace data (e.g., purchases).

Listing 1: Example of how to implement and run a new recommender strategy.

```
// Implement the recommender strategy
public interface RecommendStrategy {
   public RecommendResponse recommend(RecommendQuery q,
        Integer maxResults, SolrServer SolrServer);
}
// Run the new recommender strategy
RecommendStrategy strategyToUse = new MyStrategyImpl();
Filter filter = new ContentFilter(); // optional
RecommendationService.getRecommendations("some_user",
       "some_product", 10, filter, strategyToUse);
```

2.2 Recommender Algorithms

Currently, our framework implements four algorithms types to recommend items (in our case products) to users. This set of algorithms can easily be extended or adapted as explained in Section 2.1.

MostPopular (MP): This approach recommends for any user the same set of items, which are weighted and ranked by purchase frequency.

Collaborative Filtering (CF): Consists of recommending items to a target user that have been previously favorited, consumed or liked by similar users, the neighbors. This method is also known as K-NN because it is usually accomplished in two steps: first, find the K nearest neighbors based on some similarity metric, and second, recommend items that the neighbors have liked that the target user still has not consumed [20].

[4]http://blanc-noir.at/

[5]http://www.elasticsearch.org/

In our case, we construct the neighborhood of a user based on two types of features: marketplace features (purchases and categories), and social features (interests, groups and interactions) as shown in Table 2. As an example: In the case of purchases (CF_p), we get all purchased items of the target user and query all users that have also bought these items through the Solr data model in order to recommend their purchased items to the target user. The resultant lists of users and items are ranked and weighted using Solr's facet queries. The necessary queries for this process are the following:

```
// Find similar users based on purchased items using
     Solr's facet queries
/select?q=id:("some_product_1")+OR+id:("some_product_2")&
      facet=true&facet.field=my_users_field
// Find items purchased by those similar users that are
     new to the target user
/select?q=my_users_field:("user_1"^5+OR+"user_2"^3)&
      fq:-id:("some_product_1")+OR+-id:("some_product_2")
```

Content-based Recommendations (C): Content-based recommendation systems analyse item meta-data to identify other items that could be of interest for a specific user. This can be done based on user profile data or on the meta-data of the items that the user has liked or purchased in the past [15]. Our implementation of a content-based recommender is based on the second method and uses the built-in MoreLikeThis functionality of Solr that finds similar items for one or multiple given items by matching their content. We use two different types of meta-data features, namely the title and the description of items (see Table 2). There are several parameters for the MoreLikeThis function that can be set, e.g., the minimum document frequency (mindf), the minimum term frequency (mintf), minimum word length (minwl), etc. In the current implementation, both frequency parameters are set to 1 and the word length to 4, which gives us a good trade-off between accuracy and scalability. However, our implementation allows the application developer also to set the parameters herself, if needed. New content-based recommendation algorithms with different features can be developed by implementing the aforementioned *RecommendStrategy* Interface. The listing below shows how a content-based recommender can be called and customized in terms of the field (mlt.fl) used to match items with similar content:

```
/select?q=id:("some_product_id")&mlt=true&
 mlt.fl=description
```

Hybrid Recommendations (CCF): All three mentioned recommender algorithms have unique strengths and weaknesses, e.g., CF suffers from sparse data and cold start problems, while content-based approaches suffer from item meta-data to be utilized[4]. Hybrid recommenders combine different algorithms to tackle this issue in order to produce more robust recommendations [5]. Considering that we want to favor items recommended by more than one method, we chose to implement the hybrid approach called Cross-Source Hybrid defined in [4]:

$$W_{rec_i} = \sum_{s_j \in S} (W_{rec_i,s_j} \cdot W_{s_j}) \cdot |S_{rec_i}| \qquad (1)$$

, where the combined weighting of the recommended item i, W_{rec_i}, is given by the sum of all single weightings for each recommender source W_{rec_i,s_j} multiplied by the weightings of the recommender sources W_{s_j}. Furthermore, it uses the number of recommender sources where i appears $|S_{rec_i}|$ to strongly favor items that have been identified by more than one recommender. We use this approach to combine the different features and algorithms shown in Table 2 where each recommender source can be weighted according to its impact on the given data (e.g., its Mean Average Precision value as described in Section 3.2). This hybridization approach is just one of the many approaches of how to combine different recommender strategies as described in the aforementioned work by Burke [5]. Hence, implementing the $RecommendStrategy$ interface can lead to other approaches also in terms of the feature selection process, if needed (e.g., [17]).

Marketplace (Market)	
Number of users	72,822
Number of purchases	265,274
Mean number of purchases per user	3.64
Number of products	122,360
Mean number of purchases per products	2.17
Online Social Network (Social)	
Number of users	64,500
Number of likes	1,492,028
Number of comments	347,755
Mean number of likes per user	14.91
Mean number of comments per user	3.47
Number of groups	260,137
Mean number of groups per user	8.91
Number of interests	88,371
Mean number of interests per user	1.57

Table 1: Basic statistics of the SL dataset.

3. EXPERIMENTAL SETUP

In the following sections we describe in detail the dataset and the evaluation method and metrics used for our evaluation.

3.1 Dataset

In order to evaluate our social recommender architecture, we relied on two different sources of data to predict future product purchases (see also [25]) – online Marketplace data and an online social network data obtained from the virtual world SecondLife[6] (SL). The reason for choosing SL over other real world sources is the lack of freely available datasets that combine both social network with marketplace data from the same set of users. The overall statistics of whole dataset can be found in Table 1.

Similar to eBay, every seller in the SL marketplace[7] owns her own sub-page – called the seller's store – where all items offered are presented to the general public. As with other trading platforms such as Amazon, sellers in the SL Marketplace have the possibility to apply meta-data information such as price, title, or description to their products. Customers in turn are able to provide reviews or ratings to products. In order to crawl all stores and corresponding meta-data information as well as interactions from the SL marketplace, we exploited the fact that every store has a unique URI built from the URL pattern http://marketplace.secondlife.com/stores/STORE_ID, where STORE_ID is an incremental integer starting at 1. With this exploit at hand, we were able to download 72,822 complete user profiles with corresponding 265,274 purchases from the stores.

The online social network MySecondLife[8] was introduced by Linden Labs, in July 2011. It can be compared to Facebook regarding postings and check-ins but aims only at residents of the virtual world. Hence, users can interact with each other by sharing text messages, and commenting or liking (= loving) these mes-

[6]https://secondlife.com/
[7]https://marketplace.secondlife.com/
[8]https://my.secondlife.com/

	Set	Name	Alg.	Feature	$nDCG$	MRR	MAP	F_1	D	UC
Market	CCF_m	CF_p	CF	purchases	.0812 (.0305)	.1310 (.0492)	.0628 (.0236)	.0442 (.0166)	.4404 (.1654)	37.56%
		CF_c	CF	categories	.0312 (.0145)	.0202 (.0094)	.0226 (.0105)	.0146 (.0068)	.4945 (.2298)	46.47%
		C_t	C	title	.0361 (.0168)	.0250 (.0116)	.0267 (.0124)	.0155 (.0072)	.6000 (.2789)	46.30%
		C_d	C	description	.0370 (.0172)	.0260 (.0121)	.0280 (.0130)	.0153 (.0071)	.5991 (.2785)	46.61%
Social	CF_s	CF_i	CF	interests	.0018 (.0003)	.0012 (.0002)	.0012 (.0002)	.0006 (.0001)	.3814 (.0630)	16.52%
		CF_g	CF	groups	.0257 (.0129)	.0205 (.0103)	.0215 (.0108)	.0128 (.0064)	.3816 (.1913)	50.13%
		CF_l	CF	likes	.0120 (.0010)	.0084 (.0007)	.0096 (.0008)	.0084 (.0007)	.3269 (.0273)	8.35%
		CF_{co}	CF	comments	.0112 (.0008)	.0084 (.0006)	.0096 (.0007)	.0084 (.0006)	.3147 (.0225)	7.15%
		CF_{in}	CF	interactions	.1670 (.0192)	.1174 (.0135)	.1417 (.0163)	.1626 (.0187)	.3235 (.0372)	11.50%

Table 2: Results of the performance experiment for each recommendation approach with corresponding features (normalized to the actual UC in the row). Values in brackets represent the results normalized to 100% UC.

sages. A user pro- file can be accessed through a unique URL, https://my.secondlife.com/en/USER_ID, where USER_ID depicts the user's name. The necessary names were extracted from the SL Marketplace dataset. In order to gather the whole network we also extracted all interaction partners recursively until no new user could be found. All over, 1,839,783 interactions (likes, comments) were downloaded for 64,500 user profiles.

3.2 Evaluation Method and Metrics

To evaluate the performance of our recommendation methods in terms of accuracy, ranking, diversity and coverage, we performed a number of off-line experiments. Therefore, we split the SL dataset in two different sets (training and test set) using a method similar to the described in [14], i.e. for each user we withheld 10 purchased items (= products) from the training set and added them to the test set to be predicted. Since we did not use a p-core pruning technique to prevent a biased evaluation as suggested in related work [7], there are also users with less than 10 relevant items. For these users, we considered half of their purchased items for training and the other half for testing. With that method at hand we are able to simulate cold-start users for whom there is no item in the training set and the only relevant item is used in the test set.

For the evaluation metrics we used a diverse set of well-established measures in recommender systems. In particular, we report F_1-score (F_1), Mean Reciprocal Rank (MRR), Mean Average Precision (MAP), Normalized Cumulative Discounted Gain ($nDCG$), User Coverage (UC) [13], and Diversity (D) [21]. All performance metrics are reported for 10 recommended items (k=10).

To assess the performance of our recommender framework in terms of execution time and scalability we conducted two evaluations. In the first one, we compared the runtime of train and test datasets using different approaches and data sources. In the second, we compared different stress tests within three different scenarios. In scenario (i), we report the mean response time (in seconds) according to an increasing number of requests, in scenario (ii), we report the mean response time with 10% new randomly generated data updates (i.e., purchases) during the recommendation process and in (iii) we report the mean time needed to persist data on the disc for a different number of updates. The experiments have been executed on an IBM System x3550 server with two 2.0 GHz six-core Intel Xeon E5-2620 processors, a 1TB ServeRAID M1115 SCSI Disk and 128 GB of RAM using one Apache Solr 4.3.1. instance and Ubuntu 12.04.2.

4. RESULTS

In this section we present the results of our experiments in respect to the recommender performance and the scalability of our framework.

4.1 Algorithmic Performance

The evaluation of the performance of the recommender algorithms and data sources has been conducted in two steps, first we compared the different approaches and features on their own (see Table 2) and then we compared the combinations of those (see Table 4). The results for the different algorithms are calculated related to their user coverage and so are based only on the users where they were able to calculate recommendations as suggested in related work [9, 2]. Furthermore, also the values based on all users in the datasets are shown in parenthesis.

Table 2 shows that the best results for the accuracy metrics (F_1, MRR, MAP and $nDCG$) are reached by CF_{in} followed by CF_p, the CF approaches based on social interactions and purchases. However, the results also reveal that CF_{in} only provides a small user coverage (UC) value, where CF_p performs much better and CF_g (CF based on groups) performs best. The best diversity (D) values are reached by the two content-based approaches based on title and description (C_t and C_d). Another thing that comes apparent is, that all shown approaches clearly outperform CF_i (CF based on interests). Although the user's interest seems conceptually a good metric to assess user similarity, in the SL social network it is defined by free-written keywords and phrases of the user, which would require additional validation or processing steps in order to exploit it as an efficient source of similarity.

This pattern of results also shows that the different algorithms and features have their unique strengths and weaknesses and that a hybrid combination of those should increase the overall recommender quality in terms of accuracy, diversity and user coverage [5]. Table 4 proofs this assumption and shows the combination of the marketplace-based approaches (CCF_m = CF_p + CF_c + C_t + C_d), the combination of the social based approaches (CF_s = CF_i + CF_g + CF_l + CF_{co} + CF_{in}) and the combination of both together with MP (All = CCF_m + CF_s + CF_s) to also address the issue of cold-start users. It can be seen that our hybrid approach not only outperforms the other approaches on all metrics but also provides a UC of 100% and so it can provide recommendations for all users in the datasets.

4.2 Framework Scalability

The recommender scalability has been evaluated in two ways, first we compared the runtime of the different approaches and features, as well as the hybrid combinations of those, and second we compared the mean response time of the algorithms in form of a stress test with an increasing number of requests in three scenarios.

The results of the runtime comparison are shown in Table 3. The table reveals the mean test time ($\overline{Test}$) that is needed to calculate recommendations for a user and the overall time ($|Test + Train|$) that is needed to process all the users from the test set together with the training time (711 seconds) for building the data model

Type	CF_r	CF_c	C_n	C_d	CF_i	CF_g	CF_l	CF_c	CF_{in}	MP	CCF_m	CF_s	All
$Test$	0.020	0.097	0.029	0.094	0.024	0.023	0.011	0.013	0.021	0.016	0.194	0.024	0.197
$\|Test + Train\|$	2,167	7,775	2,823	7,556	2,459	2,386	1,513	1,658	2,240	1,876	14,838	2,459	15,057

Table 3: Results of the runtime experiment (in seconds) for each single recommendation approach and feature together with the hybrid approaches and MP as a baseline.

in Solr (i.e., indexing the data). In general these results reveal that Solr is capable of providing real-time recommendations for users as the maximum mean test time is only 0.197 seconds for our hybrid approach.

Figure 2 shows the results of the stress test with an increasing number of requests in three scenarios, first without data updates during the recommendation process, the second one is similar but includes a 10% rate of data updates (i.e., randomly generated purchases), and the third scenario shows the time needed to update data. It can be seen in the first plot (without data updates) that the mean response time follows a near linear progress for our combined hybrid approach which clearly shows the scalability of Apache Solr and our framework. Most surprisingly this is also the case in the second plot that also takes data updates during the recommendation time into account and so shows the capability of Solr in maintaining its data index in near real-time. The third plot shows that Solr is also designed to handle a high number of update requests as there is a much sharper increase in the mean update time for a small number of update requests than for a high number.

This shows that our framework based on Solr already contains algorithms that not only provide a good trade-off between recommendation accuracy, diversity and user coverage, but also provide and calculate recommendations in real-time and at scale. Furthermore, there are additional ways to optimize Apache Solr (e.g., soft commits, using an SSD disk, ...) to even better tackle the performance of committing new or existing data.

5. RELATED WORK

There are already multiple frameworks and approaches out there that focus on scalable recommendation mechanisms. Most of these approaches are based on Collaborative Filtering techniques to predict the user's ratings for items, such as movies or products, based on the user's preferences in the past. However, the computational complexity of these calculations is typically very high, especially in the case of real-time streams.

To tackle this issue, previous work focused on distributed and scalable data processing frameworks such as Apache Hadoop or Mahout based on the map/reduce paradigm (e.g., [26] or [23]). In contrast to our framework based on Apache Solr, these approaches lack the mechanisms that enable near real-time updates of the data model (data indexing) in case of new user interactions (e.g., a user purchased an item) and updates of the data schema in case of new data sources that have to be plugged in (e.g., data from HBase tables). Furthermore, it is not trivial to handle social- and content-based data with these framework, whereas this functionality comes directly out-of-the-box with Solr (e.g., with the MoreLikeThis function) together with powerful full text search functionalities. An alternative method to improve Collaborative Filtering is based on Matrix Factorization as for example proposed by Diaz-Aviles et al. [6]. However, in this work the authors focus on the near real-time processing of Twitter streams for topic recommendations and not on item recommendations in social online marketplaces as it is done with our framework.

Other approaches use database systems in order to "query" the recommendations from a data model or to simply cache the already calculated recommendations. One example for a database-driven online recommender framework is the *RecDB* project by Sarwat et al. [19] which is built on the basis of a PostgreSQL database with an extended SQL statement set. The authors show that RecDB can provide near real-time recommendations for movies, restaurants and research papers. Although these approaches perform fairly good, it has been shown that relational database management systems are insufficient for full text searches, that are the basis for content-based recommendations, where information retrieval software like Solr greatly speed up the response time of the requested queries [24].

Measure	MP	CCF_m	CF_s	All
$nDCG$	.0078	.0678 (.0316)	.0182 (.0103)	.0387
MRR	.0054	.0420 (.0196)	.0126 (.0071)	.0249
MAP	.0054	.0485 (.0226)	.0133 (.0075)	.0278
F_1	.0032	.0354 (.0165)	.0115 (.0065)	.0188
D	.3801	.4877 (.2274)	.3770 (.2129)	.4276
UC	100%	46.63%	56.47%	100%

Table 4: Results of the performance experiment for the hybrid approaches together with MP as a baseline (normalized to the actual UC in the row). Values in brackets represent the results normalized to 100% UC.

To date, there is only few research available that focus on the usage of search engines and information retrieval systems to implement recommendation services. In [22] a method is presented to implement a k-nearest neighbor-based recommendation system on top of full text search engines (MySQL and SphinxSearch) that provides linear scalability relative to the data size. Another work in this context is a recent contribution by Parra et al. [12] who implemented a recommender system for scientific talks based on Apache Solr. Although the latter mentioned contribution provides insights on how to implement a near real-time recommender system based on Apache Solr, they lack of extensive explanations and evaluations of how such an approach performs in a big data scenario.

6. CONCLUSIONS

In this paper we have presented the implementation details and evaluation of an online social marketplace recommender with a focus on two kind of readers: researchers and professionals in the area of recommender systems. On the research side, we provided results that highlight the importance of social features (interactions in the form of likes and comments) in order to improve the accuracy, diversity and coverage of product recommendations. From the side of professionals, we provided a description of our framework based on Apache Solr with detailed results in terms of performance and scalability in order to serve as a baseline for people interested in implementing a recommender system, information rarely found in current literature. Our framework evaluation considers dimensions such as hardware configuration, model training and testing trade-offs, real-time recommendation performance and the impact of model updates over the whole system performance.

We plan different tasks to extend our current study. In terms of algorithms, we would like to explore whether other hybridization techniques (weighted, mixed, etc.) can provide us alternative ways to combine methods and data sources, in order to produce

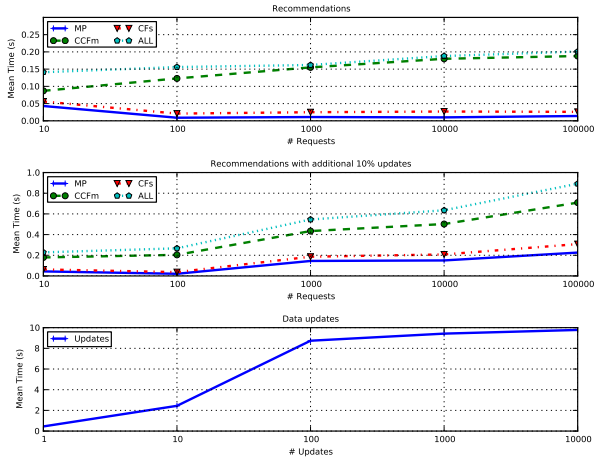

Figure 2: Scalability tests (in seconds) for 10 to 100,000 recommendation requests in three different scenarios.

an improvement in the quality of our recommendations. We also look forward to test and integrate matrix factorization techniques and study its impact in terms of recommendation quality and system scalability. We also intend to run user studies to make sure that improvements in accuracy, diversity and user coverage have a significant positive impact on user engagement and satisfaction.

Regarding the platform, our current work proofed the feasibility of only one well-known search engine backend to be easily utilized and extended as a collaborative and content filtering recommender engine. Therefore it is our aim to also investigate in depth other popular backend search solutions such as ElasticSearch to compare it with our current implementation based on Apache Solr. In this respect, we are also interested in a comparative study that investigates the performance of several collaborative and content filtering approaches grounded on SQL, Mahout or search engines (Solr, ElasticSearch). Also, we are interested in utilizing other data sources, in depth scalability experiments through sharding, and on testing different feature selection methods for recommender systems.

Acknowledgments: The authors would like to thank Michael Steurer and Lukas Eberhard for crawling the SL dataset and Alan Said and Alejandro Bellogin for value comments on the paper. This work is supported by the Know-Center. The first and the second author of this paper are supported by grants from the EU funded project Learning Layers (Nr. 318209).

7. REFERENCES

[1] M. Balabanović and Y. Shoham. Fab: content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72, Mar. 1997.

[2] A. Bellogín and J. Parapar. Using graph partitioning techniques for neighbour selection in user-based collaborative filtering. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 213–216. ACM, 2012.

[3] A. Bellogin, J. Wang, and P. Castells. Bridging memory-based collaborative filtering and text retrieval. *Information Retrieval*, 16(6):697–724, 2013.

[4] S. Bostandjiev, J. O'Donovan, and T. Höllerer. Tasteweights: a visual interactive hybrid recommender system. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 35–42. ACM, 2012.

[5] R. Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[6] E. Diaz-Aviles, L. Drumond, L. Schmidt-Thieme, and W. Nejdl. Real-time top-n recommendation in social streams. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 59–66. ACM, 2012.

[7] S. Doerfel and R. Jäschke. An analysis of tag-recommender evaluation procedures. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 343–346. ACM, 2013.

[8] Z. Gantner, S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Mymedialite: A free recommender system library. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 305–308, New York, NY, USA, 2011. ACM.

[9] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

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

[11] S. M. McNee, J. Riedl, and J. A. Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, pages 1097–1101, New York, NY, USA, 2006. ACM.

[12] D. Parra, P. Brusilovsky, and C. Trattner. User controllability in an hybrid talk recommender system. In *Proceedings of the ACM 2014 International Conference on Intelligent User Interfaces*, IUI '14, pages 305–308, New York, NY, USA, 2014. ACM.

[13] D. Parra and S. Sahebi. Recommender systems : Sources of knowledge and evaluation metrics. In *Advanced Techniques in Web Intelligence-2: Web User Browsing Behaviour and Preference Analysis*, pages 149–175. Springer-Verlag, 2013.

[14] D. Parra-Santander and P. Brusilovsky. Improving collaborative filtering in social tagging systems for the recommendation of scientific articles. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 1, pages 136–142. IEEE, 2010.

[15] M. J. Pazzani and D. Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[16] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.

[17] R. Ronen, N. Koenigstein, E. Ziklik, and N. Nice. Selecting content-based features for collaborative filtering recommenders. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 407–410. ACM, 2013.

[18] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[19] M. Sarwat, J. Avery, and M. F. Mokbel. Recdb in action: recommendation made easy in relational databases. *Proceedings of the VLDB Endowment*, 6(12):1242–1245, 2013.

[20] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer, 2007.

[21] B. Smyth and P. McClave. Similarity vs. diversity. In D. Aha and I. Watson, editors, *Case-Based Reasoning Research and Development*, volume 2080 of *Lecture Notes in Computer Science*, pages 347–361. Springer Berlin Heidelberg, 2001.

[22] J. Suchal and P. Návrat. Full Text Search Engine as Scalable k-Nearest Neighbor Recommendation System. In M. Bramer, editor, *Artificial Intelligence in Theory and Practice III*, pages 165–173. Springer Berlin Heidelberg, 2010.

[23] S. G. Walunj and K. Sadafale. An online recommendation system for e-commerce based on apache mahout framework. In *Proceedings of the 2013 annual conference on Computers and people research*, pages 153–158. ACM, 2013.

[24] O. Yilmazel, B. Yurekli, B. Yilmazel, and A. Arslan. Relational Databases versus Information Retrieval Systems : A Case Study. *IADIS International Conference Applied Computing 2009*, pages 1–4, 2009.

[25] Y. Zhang and M. Pennacchiotti. Predicting purchase behaviors from social media. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 1521–1532, 2013.

[26] Z.-D. Zhao and M.-s. Shang. User-based collaborative-filtering recommendation algorithms on hadoop. In *Knowledge Discovery and Data Mining, 2010. WKDD'10. Third International Conference on*, pages 478–481. IEEE, 2010.