Baldominos, A.; Saez, Y. ; Albacete, E. ; Marrero, I. "An Efficient and Scalable Recommender System for the Smart Web". *Proceedings of the 2015,11th International Conference on Innovations in Information Technology (IIT). Innovations 2015. Special Theme: Smart Cities, Big Data, Sustainable Development,* pp. 296 - 301. Available in DOI: 10.1109/INNOVATIONS.2015.7381557

An Efficient and Scalable Recommender System for the Smart Web

Alejandro Baldominos, Yago Saez and Esperanza Albacete
Computer Science Dept. Universidad Carlos III de Madrid
Email: {alejandro.baldominos, yago.saez, esperanza.albacete}@uc3m.es

Ignacio Marrero
Zed Worldwide
Email: ijmarrero@gmail.com

Abstract: This work describes the development of a web recommender system implementing both collaborative filtering and content-based filtering. Moreover, it supports two differ-ent working modes, either sponsored or related, depending on whether websites are to be recommended based on a list of ongoing ad campaigns or in the user preferences. Novel recommendation algorithms are proposed and imple-mented, which fully rely on set operations such as union and intersection in order to compute the set of recommendations to be provided to end users. The recommender system is deployed over a real-time big data architecture designed to work with Apache Hadoop ecosystem, thus supporting horizontal scalability, and is able to provide recommendations as a service by means of a RESTful API. The performance of the recommender is measured, resulting in the system being able to provide dozens of recommendations in few milliseconds in a single-node cluster setup.

1. Introduction

This work describes the development of a system aimed at providing web recommendations to Internet users in real time. In order to generate these recommendations, the system requires the HTTP request for the web the user is currently browsing, a set of categorized URLs and a user profile containing a set of URL categories and raw URLs, mapping those to some affinity values.

The recommender system will be able to provide recommendations in two different working modes, either a **related mode**, where the recommended URLs are based on the user preferences and the current HTTP request, and a **sponsored** mode, where the recommended URLs are retrieved from a set of active ad campaigns while still considering the user preferences and the current HTTP request.

Moreover, the system will also provide both **content-based** recommendations, where recommended URLs are based on the user profile; and **collaborative-based** recommendations, where recommended URLs are based on the profiles of similar users.

The recommender system will be implemented using a scalable machine learning architecture suitable for big data real time analysis, deployed over a Hadoop cluster in order to provide scalability for an increasing number of URLs and users. Additionally, efficient recommendation algorithms are proposed which rely on the computing of set operations (such as union and intersection of sets) for most of their logic, thus offering low computational complexity.

This document is structured as follows: first, section 2 introduces some related work which is relevant to this paper. Later, section 3 describes the big data architecture used for deploying the recommender system, as well as the input expected by the system and the output it generates; while section 4 provides a detailed description of the designed recommendation algorithms. Finally, a preliminar evaluation of the recommender system is performed by measuring its performance, results being discussed in section 5; and some conclusive remarks on this work are provided in section 6

2. State of the Art

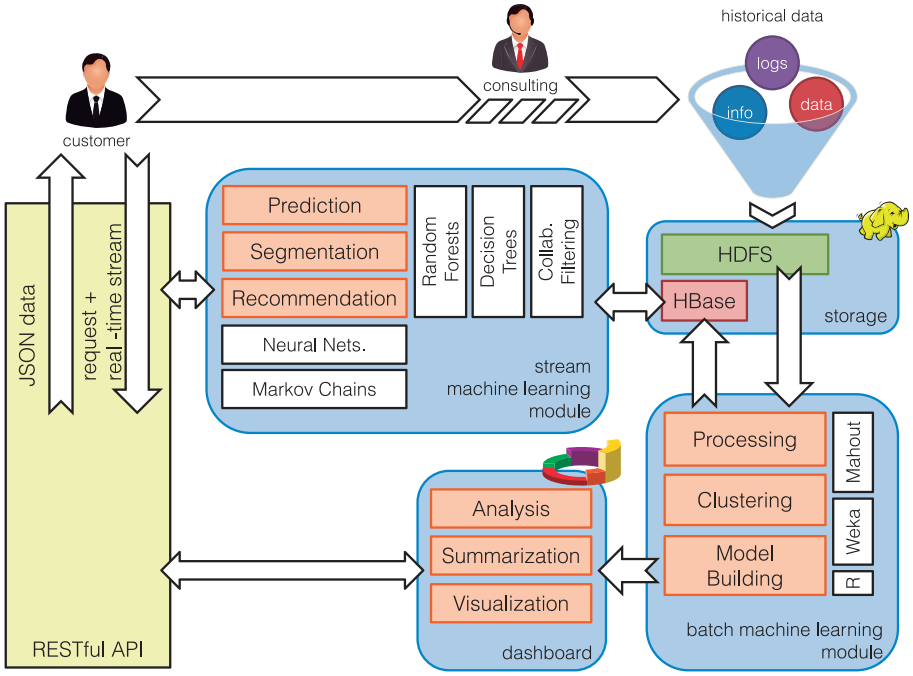
Recommender systems have been widely used for years for many different purposes ranging from e-commerce (recommending products to potential customers) to social networks (recommending possible friends or interests). While these techniques have been used for many years, the more recent appearance of big data has lead to more exhaustive studies on how high volume and velocity of data affect recommender systems, and how the latter can benefit from the prior. One of these studies is provided by Jamiy et al. [1]; and other was published by Amatriain [2], VP of Engineering at Netflix, where he revisits the problem of recommendation from a new perspective, finally discussing what big data has brought into the field. Amatriain has also published works explaining the models and large volume of data behind Netflix recommendations [3], [4]; and has also tackled the problem of recommendations given large streams of data [5].

In the last years, specific techniques enabling recommendations on big data have been studied. For instance, Xie et al. [6] propose a new approach for facing the problem of data sparsity in collaborative filtering, which is designed to work over the MapReduce framework; and Yu et al. [7], [8] have studied scalable parallel matrix factorization for recommender systems. More recently, Bokde et al. [9] have proposed an efficient algorithm for multi-criteria item-based collaborative filtering involving dimensionality reduction implemented over Apache Mahout, a scalable machine learning framework.

Moreover, the number of applications of scalable recommender systems has also grown in the last years and involves now many different areas. Sun et al, for instance, propose a MapReduce-based recommender system for e-commerce [10]; Han et al. [11] have proposed a big data model for recommendation in social networks; and Jiang and Xu [12] have presented a big-data framework for doctor recommendation.

There are many works using the Hadoop ecosystem for supporting item-based recommender systems. It is the case, for instance, of Bhatia and Prasad [13], Verma et al. [14], Kumar

Fig. 1. Scalable machine learning architecture used for supporting the recommender system



and Pandey [15] and Vinodhini et al. [16] who use Apache Hadoop and Apache Mahout for building general-purpose recommender systems. Meanwhile, Meng et al. have used Hadoop MapReduce for keyword-aware [17] and preference-aware [18] service recommendation.

3. Framework

This section will first describe the big data architecture used for real-time recommendation and how the recommender uses this architecture. Later, it will delve into the format of the input accepted by the system and the output it generates.

A. Architecture

The recommender system is built over a real-time machine learning architecture for big data recently introduced by Baldominos et al. [19]. As shown in figure 1, this architecture comprises different modules aimed at storing big data, performing batch and stream processing and displaying results; and can be operated using a RESTful API.

Figure 2 provides a more specific picture of how the proposed recommender system interacts with this architecture. In particular, raw logs are first introduced in HDFS, and a batch process periodically build clusters (using Mahout implementation of K-Means) of both websites and users. Websites are clustered using meta keywords and meta descriptions, while users (identified by IPs unless more accurate information is available) are clustered by the websites they visit. Resulting clusters are stored in HBase, as indexed row keys will enable efficient retrieval of the models required by the recommender system. The algorithms for computing recommendations are developed inside the stream machine learning module so that they can provide responses in real time.

B. Input

This section provides further detail on how the inputs of the algorithm are stored and formalized. These inputs involve the HTTP request of the user who will be recommended a website and the user profiles and web categories stored in the system, which result from the clustering process described before.

1) HTTP Request: The HTTP request is required for the user who is going to receive a recommendation for visiting another website. This request must contain the next information: the user identifier (login information or, if unavailable, the visitor IP), the URL of the website he/she is browsing, additional navigation information (environment, timestamp, etc.) and meta keywords and description of the website. The HTTP request is introduced as a JSON document.

2) User Profile: The user profile for the user to be recommended is also required in order to provide interesting recommendations. The user profile comprises an identifier and a set of web categories and URLs, each of these sets containing two affinity values. In the case of URLs, the first affinity value represents the number of visits of the user to that URL, while the second refers to the number of recommendations accepted by the user for that URL. Similarly, those values are computed for each web category by aggregation of the affinities for each URL belonging to the category. The user profiles are stored in Apache HBase, and figure 3 shows the data model for the table. The user identifier determines the row key of the entry, and qualifiers specify the URL or the web category name. Column families starting by 'r' indicate that the values they store for the affinities refer to the second type of affinity described before, i.e., are computed from the number of recommendations accepted by the user.

3) Web Categories: As already described, web categories are obtained by a clustering process using Mahout's implementation of K-Means, which periodically groups URLs with

Fig. 2. Architecture of the web recommender system

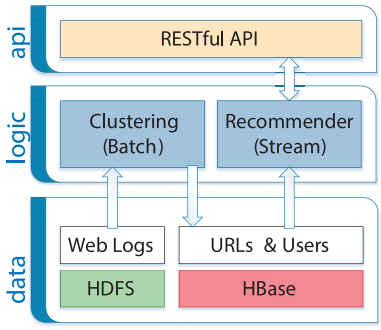


Fig. 3. Data model for the HBase table storing user profiles

rk	*cf*:cat	*cf*:url	*cf*:rcat	*cf*:rurl
<uid>	*q*:cat *v*:aff	*q*:url *v*:aff	*q*:cat *v*:aff	*q*:url *v*:aff

similar keywords and descriptions into the same cluster or category. Once this process is completed, each category is manually labelled with a human-readable name. In order to retrieve web categories efficiently, these are stored in an HBase table, whose model is described in table 4. While keywords (*kws*) are not required by the recommender system, they are used for the clustering process. The value for the category represents the total affinity of users with that category.

4) User Communities: User communities, or groups of users with similar browsing behaviors, are required for providing collaborative recommendations. Again, this information is retrieved by a clustering process using K-Means, which groups similar users into communities, this similarity being computed from the set of URLs and web categories preferred by each user, as well as their affinities. User communities are stored in HBase, in a table whose data model is equivalent to that described in figure 3 for user profiles, the only difference being that the identifier refers to a group rather than a user.

5) Ad Campaigns: Optionally, when the recommender system is working in the *sponsored* mode, it will require a set of active ad campaigns that will provide the set of possible URLs to be recommended. Each campaign stores a campaign identifier, the URLs to be recommended by the campaign and a priority value, so that some campaigns can be priorized over others based on factors such as the investment of the advertiser in the campaign. Ad campaigns are introduced into the system as JSON documents, and are only required when sponsored recommendations are requested.

C. Output

The algorithm generates a recommendation which is de-fined by a list of items, each one containing the URL to be recommended and a priority for the URL. In the case of sponsored recommendations, if no suitable recommendations were found in the ad campaigns then it is explicitly indicated.

4. Algorithms

In order to provide recomendations in real-time, not only the architecture must be scalable but the algorithms must have low computational complexity. For this reason, algorithms are designed in a way that most of the computations rely on set operations, which can be performed very efficiently. Before further details are provided, some terminology must be introduced: *a*) req refers to the current HTTP request, *b*) usr refers to the current user profile, *c*) ads refers to the ongoing ad campaigns, *d*) com refers to the user community, *e*) $wAds$ specifies the weight of the ad campaign priority in the recommendation priority, *f*) $wAff$ specifies the weight of the user affinity in the recommendation priority, *g*) $recL$ refers to a list of URLs which can be recommended for each web category; and finally *h*) $colL$ refers to a list of URLs which can be recommended for each user category.

Fig. 4. Data model for the HBase table storing web categories

rk	*cf*:kws	*cf*:cat	*cf*:url
`sha512(<url>)`	*q*:kwd	*q*:cat *v*:aff	*q*:cat

A. Content-Based Filtering

1) Sponsored Mode: The pseudocode for retrieving sponsored content-based recommendations is shown in figure 5. This algorithm first checks whether the URL in the HTTP request is categorized. In case it is not, two different actions may happen: either to *a*) provide a sponsored collaborative recommendation, in case the user is categorized or to *b*) provide a default sponsored recommendation if he/she is not.

In case the HTTP request is categorized, then the intersection between the URLs in the ad campaigns and the URLs in the recommended list for that web category is computed. Again, if this intersection is empty, one of the previous actions is taken based on whether the user is categorized or not. Otherwise the recommendation is built from these URLs, and priorities are asigned to each of them based on the weighted average of the priority of the ad campaign containing that URL and the user affinity with the web category of the URL.

2) Related Mode: The pseudocode for retrieving related content-based recommendations is shown in figure 6. In summary, the algorithm first checks whether the URL in the HTTP request is categorized. If it is not, two different actions may happen: either to *a*) provide a related collaborative recommendation, in case the user is categorized or to *b*) provide a default related recommendation if he/she is not.

In case the HTTP request is categorized, then the recommended list for that web category is retrieved. Again, if this list is empty, one of the previous actions is taken based on whether the user is categorized or not. If the recommended list is not empty, then the recommendation is built from URLs in this list, and priorities are asigned to each of them based on the user affinity with the web category of the URL.

B. Collaborative Filtering

1) Sponsored Mode: The pseudocode for retrieving sponsored collaborative recommendations is shown in figure 7. In summary, the algorithm first checks whether the user is categorized and, if he/she is not, a sponsored content-based recommendation is provided. Otherwise the intersection

Fig. 5. `WebR.Sp.Cont` Pseudocode for retrieving sponsored content-based recommendations

```
Data: req, usr, ads, wAds, wAff, recL
Result: Sponsored content-based recommendation.
if req.cats = ∅ then
    if usr.cats = ∅ then
        recs ← WebR.Sp.Default();
    else
        recs ← WebR.Sp.Col();
    end
else
    urls ← ads.urls ∩ recL.urlsByCats(req.cats);
    if urls = ∅ then
        if usr.cats = ∅ then
            recs ← WebR.Sp.Default();
        else
            recs ← WebR.Sp.Col();
        end
    else
        recs ← new Recommendation;
        foreach ad ∈ ads do
            url ← ad.url;
            if url ∈ urls then
                priority ← wAds× ad.priority +
                           wAff× usr.aff(cat);
                recs.add(url,priority);
            end
        end
    end
end
return recs;
```

Fig. 6. `WebR.Rel.Cont` Pseudocode for retrieving related content-based recommendations

```
Data: req, usr, wAff, recL
Result: related content-based recommendation.
if req.cats = ∅ then
    if usr.cats = ∅ then
        recs ← WebR.Rel.Default();
    else
        recs ← WebR.Rel.Col();
    end
else
    urls ← recL.urlsByCats(req.cats);
    if urls = ∅ then
        if usr.cats = ∅ then
            recs ← WebR.Rel.Default();
        else
            recs ← WebR.Rel.Col();
        end
    else
        recs ← new Recommendation;
        foreach url ∈ urls do
            priority ← wAff× usr.aff(cat);
            recs.add(url,priority);
        end
    end
end
return recs;
```

Fig. 7. `WebR.Sp.Col` Pseudocode for retrieving sponsored collaborative recommendations

```
Data: req, usr, ads, wAds, wAff, colL
Result: sponsored collaborative recommendation.
if usr.cats = ∅ then
    recs ← WebR.Sp.Cont();
else
    urls ← ads.urls ∩ colL.urlsByCats(usr.cats);
    if urls = ∅ then
        urls ←
        ads.urlByCats(colL.catsByCats(usr.cats));

        if urls = ∅ then
            recs ← WebR.Sp.Default();
        end
    else
        recs ← new Recommendation;
        foreach ad ∈ ads do
            url ← ad.url;
            if url ∈ urls then
                priority ← wAds× ad.priority +
                           wAff× usr.aff(cat);
                recs.add(url,priority);
            end
        end
    end
end
return recs;
```

between URLs in the ad campaigns and the URLs in the collaborative list for that user category is computed. If this intersection is empty, then URLs from ad campaigns with common web categories in the collaborative list for the user category are retrieved. If the resulting list of URLs is also empty, then a default sponsored recommendation is provided.

If the list is not empty, then the recommendation is built from the URLs in this list, and priorities are asigned to each of them based on the weighted average of the priority of the ad campaign containing that URL and the user affinity with the web category of the URL.

2) Related Mode: The pseudocode for retrieving related collaborative recommendations is shown in figure 8. In summary, the algorithm first checks whether the user is categorized and, if he/she is not, a related content-based recommendation is provided. If the user is categorized, then URLs in the collaborative list for that user category are computed. If this list is empty, then URLs in the recommended list with categories in the collaborative list for the user profile will be retrieved. If the resulting list of URLs is also empty, then a default related recommendation will be provided.

If the list is not empty, then the recommendation is built from URLs in this list, and priorities are asigned to each of them based on the user affinity with the category of the URL.

C. Default Recommendations

When the HTTP request or the users are not categorized, or the recommended or collaborative lists are incomplete, then there is not enough information for providing a recommendation. This case will happen frequently when the system begins operating, due to the effect of cold start. In this case, default (or uncategorized) recommendations are provided. These recommendations are not expected to be as suitable as the previous ones, yet they avoid the fact of not providing any recommendation at all.

Fig. 8. `WebR.Rel.Col` Pseudocode for retrieving related collaborative recommendations

```
Data: req, usr, wAff, colL, recL
Result: related collaborative recommendation.
if usr.cats = ∅ then
    recs ← WebR.Rel.Cont();
else
    urls ← colL.urlsByCats(req.cats);
    if urls = ∅ then
        urls ←
        recL.urlsByCats(colL.catsByCats(usr.cats));
    end
    if urls = ∅ then
        recs ← WebR.Rel.Default();
    else
        recs ← new Recommendation;
        foreach url ∈ urls do
            priority ← wAff× usr.aff(cat);
            recs.add(url,priority);
        end
    end
end
return recs;
```

Fig. 9. `WebR.Sp.Default` Pseudocode for retrieving default sponsored recommendations

```
Data: req, usr, ads, wAds, wAff, recL
Result: default sponsored recommendation.
urls ← ads.urls ∩ recL.urls;
recs ← new Recommendation;
foreach ad ∈ ads do
    url ← ad.url;
    if url ∈ urls then
        priority ← wAds× ad.priority + wAff×
                   usr.aff(cat);
        recs.add(url,priority);
    end
end
return recs;
```

1) Sponsored Mode: The pseudocode for providing default sponsored recommendations is shown in figure 9. In summary, the algorithm computes the intersection between the URLs from the ad campaings and the whole recommended list. Then, the recommendation is built from the URLs in this list, and priorities are asigned to each of them based on the weighted average of the priority of the ad campaign containing that URL and the user affinity with the web category of the URL. It should be noticed that the recommendation can be empty, as long as there are no common URLs between the recommended list and the ad campaigns.

2) Related Mode: The pseudocode for providing default related recommendations is shown in figure 10. In summary, the algorithm retrieves the whole recommended list for the user. Then, the recommendation is built from URLs in this list, and priorities are asigned to each of them based on the user affinity with the web category of the URL. It should be noticed that the recommendation will only be empty if the recommended list for the user is empty as well.

Fig. 10. `WebR.Rel.Default` Pseudocode for retrieving default related recommendations

```
Data: req, usr, wAff, recL
Result: default related recommendation.
urls ← recL.urls;
recs ← new Recommendation;
foreach url ∈ urls do
    priority ← wAff× usr.aff(cat);
    recs.add(url,priority);
end
return recs;
```

5. Evaluation

An evaluation of the web recommender system has been carried out in order to measure its performance both in terms of accuracy and efficiency. This section describes the experimental setup, the methodology and finally discusses the results obtained in the experiments.

A. Experimental Setup

In order to measure the quality of the recommender system, an internal pilot in Zed Worldwide was conducted with 120 subjects over a period of 4 months and a total of about 600,000 clicks, used to perform user clustering. A corpus of 456 URLs in English was used to train the model, with an average of 8 keywords used for computing URLs categories according to IAB QAG Taxonomy [20], considering up to tier 2 categories.

For the purpose of measuring computational cost we have used a single-node cluster with 8 processing cores and 16GB of RAM virtualized over VMWare ESXi 5.0, and Hortonworks HDP 2.1 as the Hadoop distribution, which includes Hadoop 2.4 and HBase 0.98. For deploying the web services we have configured JBoss AS 7. In this case, a larger synthetic dataset have been built based on URLs from DMOZ [21]. The experimental setup involves a dataset with 200,000 URLs clustered in 100 different categories, 10,000 users clustered in 20 different profiles and 2,000 ad campaigns, each of them having one or more different ads.

B. Methodology

Regarding the quality evaluation, the URLs clustering proccess was run and manually adjusted in some cases when IAB keywords showed ambiguity or lead to bad clustering results, and the recommender model was built by incorporating the URLs and users categories. For testing the recommender system, a set of 40 URLs was used to check the quality of 10 different recommendations provided to 4 different users. These recommendations were labelled as either “accurate”, “acceptable” or “wrong”.

For measuring the execution times of the recommender system we have performed three different experiments, the first one performing sequential requests and the other two performing 10 and 30 concurrent requests respectively. For each experiment the recommender system is executed 5,000 times with different input parameters, combining different recommendation modes.

TABLE I. AVERAGE AND MEDIAN TIMES (IN MS.) AS WELL AS STANDARD DEVIATION FOR THE RECOMMENDER SERVICE, BOTH WITH SEQUENTIAL AND WITH CONCURRENT (10 AND 30) REQUESTS

	Average	Median	Std. Dev.
Sequential	447.77	435	47.84
Concurrent (10)	538.80	540	40.76
Concurrent (30)	802.93	792	89.58

C. Results and Discussion

The maximum frequency of accurate recommendations for a user was 71%, whereas the average frequencies for segmented users (those already assigned to a group) was 62%accurate recommendations, 14% acceptable recommendations and 24% wrong recommendations. With unclassified users, the starting performance was of about 10% accurate recommen-dations, this value increasing as the system learnt about the user, finally reaching accuracies similar to those for previously classified users.

The recommender system is very sensitive to the keywords defined in each IAB QAG category and to the quality of the URLs keywords. For this reason, it is expected than populating the model with a large dataset of categorized URLs and users affinities could improve the results significantly.

The results for the execution times are shown in table I, which displays the mean and median times measured in milliseconds and the standard deviation. It is noticeable that average response times are always under 1 second even when the service has to attend 30 concurrent requests. The small standard deviation and the fact that the difference between the mean and the median is small reflects that there are no big differences in the response time for the different calls.

6. Conclusions

This paper has presented a scalable web recommender sys-tem built over a machine learning big data architecture recently introduced in the IEEE CIBD. Besides a scalable architecture working over Apache Hadoop ecosystem, the algorithms for computing recommendations are computationally efficient as they rely on set operations for most of their computations.

The developed recommender system is able to provide both content-based and collaborative recommendations, which can be related to the page visited by the user, or be influenced by ongoing advertisement campaigns. The system periodically computes and updates a model containing web categories and user categories, by using Mahout implementation of K-Means clusterer in an offline fashion. This model is stored in HBase, so that data can be efficiently retrieved by row key in order to provide recommendations in real time.

An evaluation has been carried out resulting in the system being able to respond to 30 concurrent requests in under a second, in a single-node cluster; and providing about 62% accurate and 14% acceptable recommendations. Further experiments are left for future work, for checking how the recommender scales when the number of nodes is increased and provide a more extensive performance evaluation.

Acknowledgment

This research work is part of *Memento Data Analysis* project, co-funded by the Spanish Ministry of Industry, Energy and Tourism with no. TSI-020601-2012-99 and TSI-020110- 2009-137.

References

[1] F. E. Jamiy, A. Daif, M. Azouazi, and A. Marzak, "The potential and challenges of Big data - Recommendation systems next level application," *CoRR*, vol. abs/1501.03424, 2015.

[2] X. Amatriain, "The recommender problem revisited," in *Proc. 8th ACM Conf. Recommender Syst.*, 2014, pp. 397–398.

[3] ——, "Big & Personal: data and models behind Netflix recommendations," in *Proc. 2nd Int. Workshop Big Data, Streams Heterogeneous Source Mining: Algorithms, Syst., Programming Models Appl.*, 2013, pp. 1–6.

[4] ——, "Building industrial-scale real-world recommender systems," in *Proc. 6th ACM Conf. Recommender Syst.*, 2012, pp. 7–8.

[5] ——, "Mining large streams of user data for personalized recommendations," *ACM SIGKDD Explor. Newsl.*, vol. 14, no. 2, pp. 37–48, 2012.

[6] F. Xie, Z. Chen, H. Xu, X. Feng, and Q. Hou, "TST: Threshold based Similarity Transitivity method in collaborative filtering with cloud computing," *Tsinghua Sci. Technol.*, vol. 18, no. 3, pp. 318–327, 2013.

[7] H.-F. Yu, C.-J. Hsieh, S. Si, and I. S. Dhillon, "Scalable coordinate descent approaches to parallel matrix factorization for recommender systems," in *Proc. IEEE Int. Conf. Data Mining*, 2012, pp. 765–774.

[8] ——, "Parallel matrix factorization for recommender systems," *Knowl. Inf. Syst.*, vol. 41, no. 3, pp. 793–819, 2014.

[9] D. Bokde, S. Girase, and D. Mukhopadhyay, "An item-based collaborative filtering using dimensionality reduction techniques on Mahout framework," *CoRR*, vol. abs/1503.06562, 2015.

[10] C. Sun, R. Gao, and H. Xi, "Big data based retail recommender system of non E-commerce," in *Proc. Int. Conf. Comput., Commun. Netw. Technol.*, 2014, pp. 1–7.

[11] X. Han, L. Tian, M. Yoon, and M. Lee, "A big data model supporting information recommendation in social networks," in *Proc. 2nd Int. Conf. Cloud Green Comput.*, 2012, pp. 810–813.

[12] H. Jiang and W. Xu, "How to find your appropriate doctor: An integrated recommendation framework in big data context," in *Proc. IEEE Symp. Comput. Intell. Healthcare e-health*, 2014, pp. 154–158.

[13] L. Bhatia and S. Prasad, "Building a distributed generic recommender using scalable data mining library," in *Proc. IEEE Int. Conf. Comput. Intell. Commun. Technol.*, 2015, pp. 98–102.

[14] J. Verma, B. Patel, and A. Patel, "Big data analysis: recommendation system with Hadoop framework," in *Proc. IEEE Int. Conf. Comput. Intell. Commun. Technol.*, 2015, pp. 92–97.

[15] T. S. Kumar and S. Pandey, "Customization of recommendation system using collaborative filtering algorithm on cloud using Mahout," in *Intelligent Distributed Computing*, ser. Advances Intell. Syst. Comput. Springer, 2015, vol. 321, pp. 1–10.

[16] S. Vinodhini, V. Rajalakshmi, and B. Govindarajulu, "Parallel matrix factorization for recommender systems," *Int. J. Eng. Research Technol.*, vol. 3, no. 4, pp. 793–819, 2014.

[17] S. Meng, W. Dou, X. Zhang, and J. Chenou, "KASR: A Keyword-Aware Service Recommendation method on MapReduce for big data applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 12, pp. 3221–3231, 2014.

[18] S. Meng, X. Tao, and W. Dou, "A preference-aware service recommendation method on Map-Reduce," in *Proc. IEEE 16th Int. Conf. Comput. Sci. Eng.*, 2013, pp. 845–853.

[19] A. Baldominos, E. Albacete, Y. Saez, and P. Isasi, "A scalable machine learning online service for big data real-time analysis," in *Proc. IEEE Symp. Comput. Intell. Big Data*, 2014, pp. 1–8.

[20] "QAG Taxonomy," http://www.iab.net/QAGInitiative/overview/taxonomy, [Online, accessed 2015-07-01].

[21] "DMOZ - the open directory project," http://www.dmoz.org, [Online, accessed 2015-07-01].