FRAC+: A DISTRIBUTED COLLABORATIVE FILTERING MODEL FOR CLIENT/SERVER ARCHITECTURES

Sylvain Castagnos, Anne Boyer
LORIA - Universit Nancy 2
Campus Scientifique, B.P. 239
54506 Vandoeuvre-lès-Nancy Cedex, France

Keywords: Collaborative filtering, user modeling, client/server algorithm, privacy, scalability, sparsity.

Abstract: This paper describes a new way of implementing an intelligent web caching service, based on an analysis of usage. Since the cache sizes in software are limited, and the search for new information is time-consuming, it becomes interesting to automate the process of selecting the most relevant items for each user. We propose a new model (FRAC+), based on a decentralized collaborative filtering algorithm (FRAC) and a behavior modeling process. Our solution is particularly designed to address the issues of data sparsity, privacy and scalability. We consider the situation where the set of users is relatively stable, whereas the set of items may vary considerably from an execution to another. We furthermore assume that the number of users is much more important than the number of items. We argue that a user-based approach seems to be more suitable to address the aforementioned issues. We present a performance assessment of our technique called FRAC in terms of computation time and prediction relevancy, the two most reliable performance criteria in the industrial context we are involved in. This work has been implemented within the ASTRA satellite website broadcasting service.

1 INTRODUCTION

With the development of information and communication technologies, the size of information systems all over the world has exponentially increased. The amount of data on the Web, for example, has crossed the threshold of the 7500 terabytes in 2004. Consequently, it becomes difficult for users to identify interesting items in a reasonable time, even if they use a powerful search engine. To cope with this problem, more and more companies choose to integrate a recommender system in their products. The goal is then to provide users with resources likely to interest them, instead of waiting that they ask for them. These processes of investigation may be provided by collaborative filtering techniques. They rely on the principle that users who liked the same documents have the same topics of interests. Thus, it is possible to predict pieces of data likely to live up users' expectations by taking advantage of experience of a similar population.

Nevertheless, collaborative filtering algorithms are still faced with numerous problems: mobility of user profiles (Miller et al., 2004), security of the system, trust, portability on different platforms, etc. In this paper, we propose an optimization of the distributed collaborative filtering model we have made explicit in (Castagnos et al., 2005). It has been especially designed to deal with problems of privacy, sparsity (cf. infra, 3.1 Behavior modeling) and scalability (cf. infra, 3.2 Clustering algorithm).

First, we would like to familiarize readers with collaborative filtering methods (cf. infra, section 2). Afterwards, we will present our model called FRAC+ which has been applied to satellite website broadcasting. The fourth part is then dedicated to the evaluation of our algorithm, both in terms of computation time and relevancy of recommendations.

2 STATE OF THE ART

BREESE *et al.* (Breese et al., 1998) have identified, among existing techniques, two major classes of algorithms to solve this problem: memory-based and model-based algorithms.

The memory-based algorithms maintain a database containing votes of all users. A similarity score is de-

Castagnos S. and Boyer A. (2006).
FRAC+: A DISTRIBUTED COLLABORATIVE FILTERING MODEL FOR CLIENT/SERVER ARCHITECTURES.
In *Proceedings of WEBIST 2006 - Second International Conference on Web Information Systems and Technologies - Internet Technology / Web Interface and Applications*, pages 435-440
DOI: 10.5220/0001248804350440

termined between the active user and each of the other members. Then, each prediction leads to a computation on all of this source of data. The influence of a person is all the stronger when his/her degree of similarity with the active user is high.

These memory-based techniques offer the advantage to be very reactive, by integrating immediately modifications of users profiles into the system. However, BREESE *et al.* (Breese et al., 1998) are unanimous in thinking that their scalability is problematic: even if these methods work well with small-sized examples, it is difficult to change to situations characterized by a great number of documents or users. Time and space complexities of algorithms are much too important for large databases.

The model-based algorithms are an alternative to the problem of combinatorial complexity. In this approach, collaborative filtering can be seen as the computation of the expected value of a vote, according active user preferences. These algorithms create descriptive models correlating persons, resources and associated votes using a learning process. Then, predictions are infered from these models.

According to PENNOCK *et al.* (Pennock et al., 2000), model-based algorithms minimize the problem of algorithmic time complexity. Furthermore, they view in these models an added value beyond the sole function of prediction. They highlight some correlations in data, thus proposing an intuitive reasoning for recommendations or simply making the hypotheses more explicit. However, these methods are not reactive enough and they react badly to insertion of new contents in database. Moreover, they require a learning phase being both detrimental for the user[1] and expensive in computation time for large databases.

Consequently, one of the main difficulties of collaborative filtering remains the scalability of systems. (Sarwar et al., 2001) have paved the way by proposing an alternative. They suggest to compute recommendations by identifying items that are similar to other items that the user has liked. They assume that the relationships between items are relatively static. Nevertheless, we have chosen to investigate the case where available items change periodically and radically after a while. The item-based algorithm doesn't seem relevant to handle this problem. Moreover, we also consider the situation where the number of users is far more important than the number of items. Therefore, we have chosen to explore ways to decentralize computations. The model proposed in this article is an hybrid approach, combining the advantages of memory-based and model-based methods to distribute the computations between the server and the clients. We study this approach, because it is always used for satellite broadcasting or for e-commerce applications at times.

[1]The recommendations system won't be able to provide relevant documents as soon as it receives the first queries.

3 ARCHITECTURE (FRAC+)

The architecture of our information filtering system is shown on figure 1. This model associates a user modeling method based on the `Chan` formula (Chan, 1999) (cf. infra, 3.1 User modeling, p. 3) and a new version of the hierarchical clustering algorithm, also called `RecTree` (Chee et al., 2001) (cf. infra, 3.2 Clustering algorithm, p. 3). This new version – called `FRAC` in the rest of this article – has the advantage to be distributed and optimized in computation time.

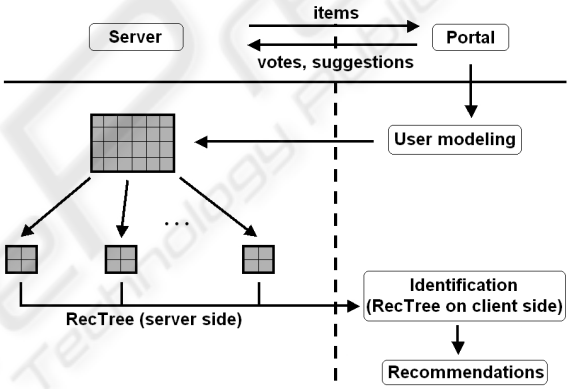

Figure 1: Architecture of the information filtering module.

We implemented our work in the context of satellite website broadcasting. Our model has been integrated in a product of `ASTRA`[2] called `Casablanca`. The satellite bouquet holds hundreds of websites which are sent to about 120.000 persons using satellites. Moreover, the users can send non-numerical votes (cf. infra, 3.1 Behavior modeling, p. 3). These votes appear as a list of favorite websites.

In order to distribute the system, the server side part is separated from the client side. The function of user modeling determines numerical votes for items according to user actions. Then, numerical votes are sent to the server, like the non-numerical ones[3]. Thus, the server uses the matrix of votes to build typical user

[2]http://www.ses-astra.com/

[3]The list of favorites are given explicitly by users, while numerical votes are estimated in a transparent way. We consequently use the non-numerical votes to determine the content of the bouquet.

Interest(item) = 1 + 2 . IsFavorite(item) + Recent(item) + 2 . Frequency(item) . Duration(item) + PercentVisitedLinks(item))

$$\text{With: } Recent(item) = \frac{date(last\ visit) - date(log\ beginning)}{date(present) - date(log\ beginning)}$$

$$\text{And: } Duration(item) = max_{consultations}\left(\frac{time\ spent\ on\ pages of item}{size\ of\ the\ item}\right) \quad (1)$$

profiles. In this way, the server has no information about the population, except anonymous votes. User preferences are stored in the profile on clients. Thus, the confidentiality criterion is duly fulfilled. At last, the active user is identified on client to one of the typical users groups in a very short time, in order to do predictions.

3.1 Behavior Modeling

In our context, we assume that users have the possibility to define a list of favorites. However, we can't describe these non-numerical votes as boolean. We can't differentiate items in which the active user is not interested (negative votes) from those he/she doesn't know or has omitted. This kind of votes is not sufficient to do relevant predictions with collaborative filtering methods.

For this reason, we have chosen to determine numerical marks without any rating[4] from users. Another advantage of this method is to deal with the problem of sparsity by increasing the number of votes in the matrix. To do so, we chose to develop the user modeling function shown in equation 1 (Chan, 1999). In our case, items correspond to websites, that is to say sets of pages. Thus, the time spent on an item is calculated as the cumulative times spent on each of its pages for example. We modified coefficients in the original `Chan` formula, in order to optimize the results in accordance with log files of `ASTRA`.

This function undertakes to estimate marks that the user is likely to give to different sites from implicit criteria (such as time or frequency that user takes to consult a page[5]). The system analyses log files of the active user to retrieve useful data. But all pieces of information retrieved in these log files remain on client side, in order to preserve privacy. Only numerical votes which have been deduced from this process are sent anonymously to the server. We call them "user profiles". They are required for the use of `FRAC` clustering algorithm.

[4]Ideally, the numerical votes should be submitted to their approval for checking.

[5]These are pieces of information easily and legally salvageable in Web browser of client.

3.2 Clustering Algorithm

Once the profiles of users have been sent to the server, the system has to build virtual communities of interests. In our model, this step is carried out by an improved hierarchical clustering algorithm, called `FRAC`. It attempts to split the set of users into cliques by recursively calling the nearest neighbors method (`K-Means`).

The original algorithm was purely centralized, such as most of existing collaborative filtering methods. One of our contributions consists in distributing this process. In this section, we explain how to build typical user profiles on server side and how to identify the active user to a group. This second step takes place on client side. We optimized the identification phase so that the response time is very short. Thus, the client part provides real-time predictions. In a second time, we improve the offline computation time by refining the initial conditions of `K-Means`.

The `FRAC` algorithm is a model-based approach, described as a clustering method. However, it is managed as a memory-based approach because all the pieces of information are required for similarity computation. It allows, within the scope of our architecture, to limit the number of persons considered in the prediction computations. Thus, the results will be potentially more relevant, since observations will be based on a group closer to the active user. A way to popularize this process amounts to considering that the active user asks for the opinion of a group of persons having similar tastes to his/hers[6].

In order to compute these groups of interests, the server extracts data from the profiles of users and aggregates the numerical votes in a global matrix. This matrix constitutes the root of the tree. The set of users is then divided into two sub-groups using the `K-means` method. In our case, the number k equals 2, since our overall strategy is to recursively divide the population into binary sub-sets. Once this first subdivision has been completed, it is repeatedly applied to the new subgroups, and this until the selected depth of the tree has been reached. This means, the

[6]The computer process is obviously transparent for users.

more one goes down in the structure of the tree, the more the clusters become specific to a certain group of similar users. Consequently, people belonging to a leaf of the tree share the same opinion concerning the assignment of a rating for a given item.

The `K-Means` algorithm is very sensitive to initial starting conditions and may converge more or less quickly to different local minima[7]. The usual way to proceed consists in choosing randomly k centers in the users/items representation space. Numerous studies have been made to improve `K-Means` by refining the selection of these initial points (Bradley and Fayyad, 1998). But it remains a difficult problem and some of these approaches don't obtain better results than the method with a random initialization. In our case, the problem is much simpler since we only have two centers. Thus, we propose a new way to select the starting points, shown on figure 2.

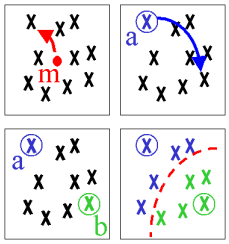

Figure 2: Initialization of `2-Means` algorithm.

We work in a N-dimensional space, since the coordinates correspond to the votes of users for the N items. However, the example on the diagram 2 is in dimension 2 for more legibility. We start from the principle that the two most distant users are inevitably in different clusters. Consequently, they constitute the ideal candidates for the initial points. To identify them, we first search the most distant point from the middle `M` of the users/items representation space. This point is called `A` on figure 2. Then, we compute the point `B`, which is the most distant from `A`. `A` and `B` are subsequently the starting points of the `2-Means` algorithm. This initialization phase is in $o(2n)$, where n is the number of users. Afterwards, each user is positioned in the cluster of the nearest center.

Once groups of persons have been formed as previously mentioned, the position of the center is recalculated for each cluster (either by computing the isobarycenter, either by using the equation 2 according to the precision we want) and this operation is repeated

[7]We want to minimize the distances between users of a same group and maximize them between users of different clusters.

$$r_{c_{t+1},l} = \frac{1}{\sum_{Y_{c_t,u}} |w(c_t,u)|} . \sum_{Y_{c_t,u}} (r_{u,l}.|w(c_t,u)|) \quad (2)$$

With: $r_{c_{t+1},l}$ the value of c_{t+1} for item l;
$r_{u,l}$ the vote of the user u for the item l;
$w(c_t,u)$ the distance between c_t and u;
$Y_{c_t,u} = \{u|w(c_t,u) \neq 0\}$;

from the beginning until we have obtained a stable state (where the centers no longer move after recalculation of their position). Our initialization allows to reach this state much more quickly.

The tree building complexity yields $o(n.log_2 n)$. The final center of each leaf of the `FRAC` tree corresponds to a profile of typical users. It means that we consider these centers as virtual users synthesizing the preferences of each subset of users.

The profiles of typical users are then sent on client side. Subsequently, the system compute distances between the active user and the typical users. We consider that the active user belongs to the community whose center is the closest to him/her. At last, we can predict the interest of the active user for a resource r_l with the equation 3.

$$p_{u_a,r_l} = max(r_{min}, min(r_{u_t,l} + (\overline{r_{u_a}} - \overline{r_{u_t}}), r_{max})) \quad (3)$$

With: u_a the active user;
u_t the nearest typical user;
p_{u_a,r_l} the prediction of u_a for r_l;

The clusterization can be performed so that cliques hold about the same number of persons for a given depth of the tree. In this way, we introduce novelty in recommendations.

4 PERFORMANCE ANALYSIS

In this section, we compare our clustering algorithm with `Item-item` (Sarwar et al., 2001), `RecTree` (Chee et al., 2001) and the Correlation-based Collaborative Filter `CorrCF` (Resnick et al., 1994). We have implemented all these methods in Java. We evaluate these techniques in terms of computation time and relevancy of predictions.

4.1 Computation Time

In order to compare the computation times of the aforementioned algorithms, we have generated matrices with different sizes. In this simulation, the votes of each user follow a Gaussian distribution centered on the middle of the representation space. We argue that this situation increases the number of iterations needed in the clustering algorithm, since the users are close to each other. Moreover, there is only 1% of missing data in the generated matrices. Consequently, we almost work in worse case for the computation time tests.

The results of these tests are shown in the table 1. The announced times include the writing of results in text files. The `FRAC` algorithm provides results in a quite short time. It is thus possible to apply it to large databases. For example, the system only needs about 6 or 7 minutes to compute typical behavior profiles with 10.000 users and 100 items. In the same case, the `CorrCF` algorithm requires several hours of computation (one of which is spent to compute the similarity matrix). The response time of `CorrCF` increases besides exponentially with the size of the database and is not in accordance with industrial constraints.

We note that `Item-Item` gets results much more quickly than `FRAC` when there is a lot of users and only few items. Nevertheless, the tendency may be reversed when the number of items grows, even if the number of users is still much more important (cf. table 1 for 10.000 users and 1000 items). Moreover, the corpus we used is not the most appropriate for our algorithm, since the number of items is almost twice as big as the number of users.

At last, we have tried to cluster huge populations with the `FRAC` algorithm. The latter was able to supply results:

- in 6 hours and 42 minutes for 100.000 users and 100 items;
- in about 11 hours for 120.000 users and 150 items.

4.2 Recommendations Relevancy

In order to compute the prediction relevancy in our system, we used the `GroupLens` database[8]. The latter is composed of 100.000 ratings of real users. Thus, we considered a matrix of 943 users and 1682 items. Moreover, each user has rated at least 20 items. The database has been divided into a training set (including 80% of all ratings) and a test set (20% of votes). We compare our algorithm with the three others by using the *Mean Absolute Error* (`MAE`).

[8]http://www.grouplens.org/

The results are shown in the figure 3. The `FRAC` algorithm does predictions as good as the `CorrCF` – which is memory-based – and not so far from the `Item-Item`.

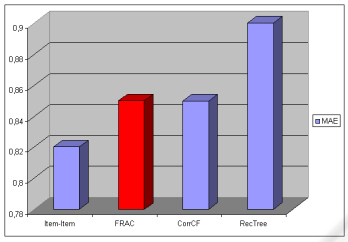

Figure 3: Comparison of prediction quality.

4.3 Discussion

This study highlights the fact that our algorithm gets rather quickly results on server side, even when the corpus is not very adequate. By way of comparison, the offline part of `RecTree` (Chee et al., 2001) – that is to say the clustering process – with 1.400 users and 100 items is done in about 1000 seconds. Our algorithm does the same job in 11 seconds (cf. supra, table 1) and is consequently almost hundred times faster in this case.

Moreover, the online part of computations – that is to say identification of user to a group – is in $o(2p)$, where p corresponds to depth of the tree. This part of computations has been optimized in our model, in comparison with the centralized `RecTree` algorithm of Chee *et al*. The online part of Chee's algorithm was in $o(b)$, where b was the number of users in each partition. Users had consequently to wait for a few seconds. In our version, the complexity of client part only depends on the depth of the tree and the response time is much faster.

Another advantage of our algorithm is the stability of the model. Thanks to the new initialization phase, the results are reproducible when we launch the clustering process several times. Furthermore, the convergence is assured, contrary to the original `RecTree`.

We have also noticed that the `FRAC` computation time can still be important for large databases. However, although the tests showed the `Item-Item` algorithm is suitable when there are few items, the required time has been considerably reduced in comparison with `RecTree` or `CorrCF`. Moreover, we recall that we consider the case where the set of items can change radically. In this case, a reasonable number of votes must be done on new items so that `Item-Item`

Table 1: Computation times of different collaborative filtering algorithms.

Items	100			150			1000		
Users	FRAC	CorrCF	Item-Item	FRAC	CorrCF	Item-Item	FRAC	CorrCF	Item-Item
400	1”84	6”09	3”87	2”09	7”62	5”29	8”58	32”24	1’22”
800	7”03	19”98	7”23	7”34	25”67	10”53	30”17	1’52”	2’33”
1.400	**11”21**	1’00”	11”50	12”81	1’17”	18”10	49”47	6’04”	4’29”
10.000	6’50”	7h30’	1’22”	9’12”	-	2’05”	14’22”	-	49’28”

can compute similarities. On the contrary, our algorithm needs less votes and recomputations because correlations are made between users.

Of course, the more the offline computations of our algorithm take time, the more it can augur for slight differences between the updated votes and the preferences really taken into account during the clustering process. But these differences should be minimal because of the great number of users.

5 CONCLUSION AND PERSPECTIVES

The novelty of our model relies on the fact that we have mixed a distributed collaborative filtering method with a behavior modeling technique. The main advantage of this combination is to take into account in an overall way the strong constraints due to an industrial context such as the privacy of users and the sparsity of the matrix of votes. Moreover, thanks to the new version of the clustering algorithm, we use the matrix of votes to divide all the users into communities. This new version has been especially designed to treat a high quantity of information (Castagnos et al., 2005) and allows the scalability to real commercial applications by dealing with time constraints. We have implemented our architecture in a satellite broadcasting software with about 120.000 users in order to highlight the benefits of such a system.

We are now considering the possibilities to combine our model with additional item-based filters in order to sort items in increasing order of importance for the active user on client side. In particular, we are studying the added value of bayesian networks and content-based filtering techniques in our architecture.

ACKNOWLEDGEMENTS

We want to thank SES ASTRA and the CRPHT which have encouraged this work.

REFERENCES

Bradley, P. S. and Fayyad, U. M. (1998). Refining initial points for k-means clustering. In *Proceedings of the 15th International Conference on Machine Learning (ICML98)*, pages 91–99, San Francisco, USA. Morgan Kaufmann.

Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 43–52, San Francisco, CA.

Castagnos, S., Boyer, A., and Charpillet, F. (2005). A distributed information filtering: Stakes and solution for satellite broadcasting. In *Proceedings 2005 Int. Conf. on Information Systems and Technologies (WebIST05)*, Miami, USA.

Chan, P. (1999). A non-invasive learning approach to building web user profiles. In *Workshop on Web usage analysis and user profiling, Fifth International Conference on Knowledge Discovery and Data Mining*, San Diego.

Chee, S. H. S., Han, J., and Wang, K. (2001). Rectree : An efficient collaborative filtering method’. In *Proceedings 2001 Int. Conf. on Data Warehouse and Knowledge Discovery (DaWaK’01)*, Munich, Germany.

Miller, B. N., Konstan, J. A., and Riedl, J. (2004). Pocketlens: Toward a personal recommender system. In *ACM Transactions on Information Systems*, volume 22, pages 437–476.

Pennock, D. M., Horvitz, E., Lawrence, S., and Giles, C. L. (2000). Collaborative filtering by personality diagnosis: a hybrid memory- and model-based approach. In *Proceedings of the sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-2000)*, San Francisco, USA. Morgan Kaufmann Publishers.

Resnick, P., Iacovou, N., Suchak, M., Bergstorm, P., and Riedl, J. (1994). Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, North Carolina. ACM.

Sarwar, B. M., Karypis, G., Konstan, J. A., and Reidl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *World Wide Web*, pages 285–295.