Optimizing and Evaluating Stream-based News Recommendation Algorithms

Sebastian Werner[1] and Andreas Lommatzsch[2]

[1] Database Systems and Information Management Group, DIMA
Technische Universität Berlin, Einsteinufer 17, D-10587 Berlin, Germany
sebastian.werner@campus.tu-berlin.de
[2] Agent Technologies in Business Applications and Telecommunication Group, AOT
Technische Universität Berlin, Ernst-Reuter-Platz 7, D-10587 Berlin, Germany
andreas.lommatzsch@tu-berlin.de

Abstract. Due to the overwhelming amount of items and information users need support in finding the information matching the individual preferences and expectations. Real-time stream-based recommender systems get in the focus of research allowing the adaption of recommendations to the user's context and the current set of relevant items. In this paper we focus on recommending news articles. In contrast to most traditional recommender systems, our system must handle several additional challenges: News articles have a short lifecycle forcing the recommender system to continuously adapt to the set of news articles. In addition, the recommender algorithms should work efficiently: On the one hand, news recommendations must be provided within milliseconds since the recommendations must be embedded in news article pages. On the other hand, the news algorithms must be able to handle a huge amount of recommendation request in order to process load peaks without violating the time constraints. We present algorithms optimized for providing real-time news recommendation given limited hardware resources. We present an offline evaluating framework allowing us the efficient optimizing of recommender algorithms taking into account the available hardware resources. The evaluation shows that our approach allows us to find optimal recommender algorithms for a given hardware setting.

1 Introduction

The overwhelming amount of news available in the internet makes it difficult for the user to find information making it difficult for the user to find articles matching the individual preferences and interests. Recommender systems address this problem by filtering the huge amount of data and suggesting the most relevant information according to the user's preferences [5]. Recommending web-news articles differs from "traditional" product recommendation scenario in several aspects: News articles have a short life cycle forcing the recommender to continuously consider new articles and to discard outdated articles. Users are interested in several different topics and domains making it hard to predict the relevance articles in a new domain. Unexpected events may be relevant for a user even though no similar event has been observed in the past [3].

From the perspective of the recommender system provider, there are high demands according to the scalability of the recommender algorithms. The systems must be able to

handle a large number of news articles, item updates and requests ensuring a very short response time for delivering recommendation results. Since users tend to read online articles especially in the morning and during the lunch break but not at night, there is high variance in the number of requested recommendations over a typical day. This requires special effort to guarantee quick responses also during the load peak hours of the day [9].

In this paper we suggest several recommender algorithms tailored for the news recommendation scenario. The algorithms are optimized for delivering high-quality news recommendations even if only limited computing resources are available. For proving the efficiency of our algorithm, we implemented a framework for evaluating news recommender algorithms. This framework allows us optimizing the algorithms for typical news recommendation scenarios and pre-defined hardware resources.

The remaining paper is structured as follows: In Section 2, we give an overview on related work. The problem description and the addressed challenges are explained in Section 3. Our approach is presented in Section 4. Subsequently, the evaluation results are discussed in Section 5. We conclude with a view on future work in Section 6.

2 Related Work

There is a big variety of domains in which recommender systems are used [1]. This section presents two systems tailored to recommend news articles and videos content. These systems have been selected because both systems are optimized for handling a huge number of requests and do not require an exact identification of users. We discuss the approaches used by the portal REDDIT and YOUTUBE and explain how the algorithms can be adapted for our news recommendation scenario.

2.1 reddit

The internet platform REDDIT[3] gives users the ability to share news articles and comment on them. The portal aggregates the user interactions in order to support users in finding the most interesting articles and comments.

The portal REDDIT allows users to rate articles by assigning `+1` ("up") for interesting or `−1` ("down") for boring articles. These votes are evaluated immediately and used for sorting the articles on the website: The most frequently positively voted articles are displayed on top of the page. The top voted news articles can be seen as recommendations computed by the system. The recommendations are similar for all users; individual user preferences are not taken into account. The aggregation of all user votes in one "global" user profile prevents problems with sparse profiles and new users ("cold-start problem"), but limits the recommendation quality.

In addition to the functionality to vote on articles, REDDIT allows the users to categorize articles. The categorization is used to compute several different lists of recommendations. Based on the categorization, users can focus on news articles related to a particular domain.

[3] `http://www.reddit.com`

REDDIT is used by millions of users every day. This results in a high number of recommendation requests and article updates. REDDIT is a relevant example of a news recommendation systems which provides an efficient solution for recommending news articles based on some limited computing resources. The used algorithm is simple, but it allows the portal to provide good recommendations without high computational complexity.

2.2 YouTube

YouTube is a popular video portal providing a rich variety of audio and video content. Since YOUTUBE is financed by advertisements, the web portal aims to prolong the time users spend on YouTube. An integrated recommender system suggests the user potentially interesting items trying to give the users the illusion that the web page has been designed according to the individual user preferences. Due to the huge number of users and items, the recommender algorithms must be designed serving a large number of recommendations and to analyze millions of user interactions each hour.

The YouTube recommender system must be able to cope with several challenges: Video content uploaded by users is often very sparsely annotated with meta-data. In general, users are interested in several different (often only loosely related) topics. In contrast to movie rental or purchase sites (such as Netflix or Amazon), YOUTUBE users do not pay for videos. Therefore, an user-item interaction on YouTube gives a less significant indication of user interests [4]. The YOUTUBE recommendation engine solves these challenges by analyzing the co-visitation between videos. This approach works well with noisy user profiles and sparsely annotated items since it is based on the user-item interaction statistics. Due to the huge number of users visiting YOUTUBE every hour, this scalable approach provides good recommendations even if limited computational resources are available.

2.3 Discussion

Most traditional recommended systems use item-based or user-based collaborative filtering algorithms [6]. The approaches enable high quality recommendation results if an appropriate number of user-item relations are available. In general, collaborative filtering algorithms suffer from sparse user-item matrixes and missing data for new users and items. Dimensionality reduction approaches (e.g., based on SVD) or clustering algorithms are frequently used to address these challenges. Since these algorithms are computational expensive, the pre-processing is often computed offline. In news recommendation scenarios characterized by a short item lifecycle and a continuously changing set of items an offline processing results in a delay inappropriate for recommending the most current news articles.

In scenarios characterized by a frequently changing set of users and items simplified recommender models are used aggregating the profiles from different users in one big, dense profile. Such recommender approaches used by REDDIT and YOUTUBE allow the portals to avoid the cold-start problem as well as to mitigate the lack of sophisticated user tracking capabilities. The low computational complexity of the algorithms, the good recommendation quality the highly aggregated model as well as the ability to provide

recommendations for unregistered users are the strengths of the approach. A weakness of the impersonalized recommender approach is that individual preferences are not taken into account. The use of categories (e.g. in REDDIT) attenuates the problem allowing the users to restrict the recommendations to one domain.

In most recommender scenarios, suggestions must be provided within seconds to ensure a good user experience. The PLISTA news recommendation scenario has much tighter time constrains: The recommendation requests must be answered within 100ms. This limits the complexity of the recommender algorithms. Thus, we focus in our research on robust, efficient algorithms able to guarantee a fast response time even in the daily load peaks.

3 Problem Description

In this section we discuss the characteristics of the analyzed news recommendation scenario and explain the challenges.

3.1 The analyzed scenario

We analyze the task of recommending interesting article for a set of news portals. When a user visits a news portal, the news portal sends a recommendation request to the PLISTA server. The PLISTA server delegates the request to a randomly selected recommender algorithm. The PLISTA challenge gives researchers the unique opportunity to test recommender algorithms online in a real-world scenario. The recommendation algorithm selected for handling the request must provide a list of recommendations within 100ms in order to ensure that the recommendations can be embedded into the requested news article. The process is visualized in Fig 1. In order to support extended offline testing of recommender algorithms, we add an offline test server to the contest architecture. The offline test server is set up by a VAGRANT[4] script and allows us to analyze the performance of recommender algorithms based on user interactions data logged in the past ("replay of logged message streams").

3.2 Dataset analysis

For the communication between the PLISTA server and the recommender algorithms four types of messages are used:

Impression: If a user requests a news article or clicks on an article recommendation, the PLISTA server informs all recommender algorithms about the event. In detail, the message contains a unique article ID, a user ID generated by the PLISTA server as well as environment parameters provided by the user's browser. The recommender algorithms do not have to answer impression messages.

Request: If a recommender algorithm is selected to provide recommendations, the PLISTA server sends a recommendation request. In addition to the data provided in the impression messages, the request defines how many recommendations must be

[4] `http://www.vagrantup.com/`

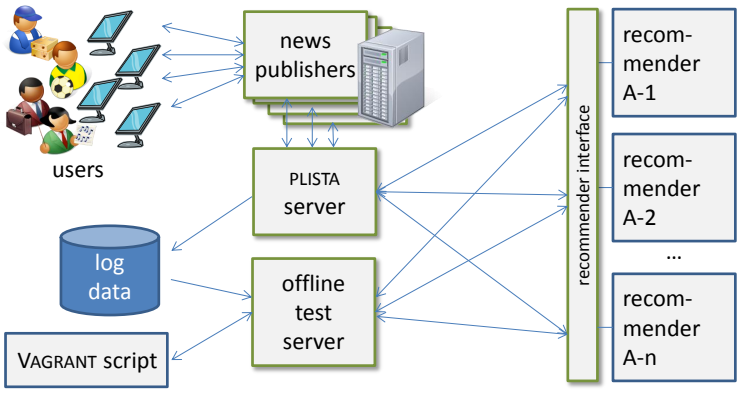

Fig. 1: The Figure visualizes the architecture of our system. The implemented recommender algorithms communicate with the PLISTA server using the recommender interface in the online evaluation. In order to evaluate recommender strategies offline the PLISTA server can be replaced by an offline testing server controlled by a VAGRANT script. In the offline evaluating log data recorded in the online evaluation are re-played allowing us to evaluate the performance of the recommender algorithms in a reproducible way. The messages as well as the log data are represented as JSON objects due to the fact that JSON is a very flexible data format well supported by web applications.

provided. Recommendation requests must be answered with the requested number of recommendations within 100ms.

Item update: If the news portal publishes a new article or if a news article changes, the PLISTA server uses item-update messages to inform the recommender agents. Item-update messages do not have to be answered by the recommender agents.

Error messages: If the recommender agent has not answered a recommendation request correctly, the PLISTA server informs the recommender agent. Potential reasons triggering an error message are timeouts or invalid article IDs in the set of recommended articles.

Since the news recommendations are provided in real-time for a live system, the amount of messages highly depend on the time of the day and the number of news published on the news portals [2].

3.3 The challenges

The implementation of a news recommendation system leads to a number of interesting challenges. These challenges range from computational limitations to the quality of available data to the heterogeneity of the news portals [8]. The quality of data, that PLISTA receives and forwards to the different participants of the open recommendation research platform vary from news portal to news portal. On some sites, users can be identified using cookies and other tracking methods. Other sites do not track user behavior and therefore make it difficult to target specific user interests.

Since users do not need to log into the news portals, the accuracy of identifying unique users is limited. This makes it difficult for user based methods to acquire comprehensive data to compute highly personalized recommendations. The way user tracking is

implemented on news portals makes it difficult to recognize a user when the user returns to a news portal. Fig. 2 shows the number of impressions for uniquely identified users. The graph shows that most users could only be identified once or twice.

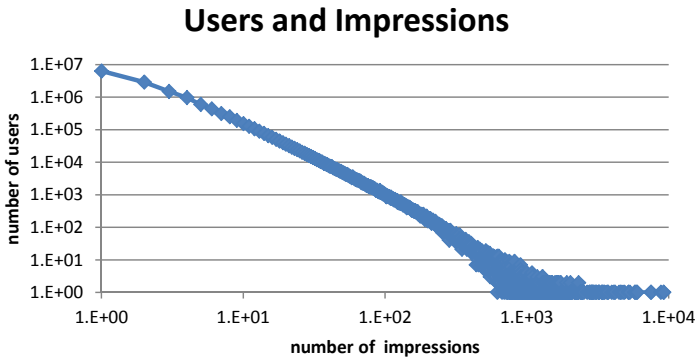

Fig. 2: The figure visualizes the number of visits per user of the PLISTA data set.

An additional challenge for learning a powerful recommendation strategy is the high variance in the user behavior. In contrast to shopping portals offering relatively expensive products, the consumption of news articles is driven by reported events, the mood of the user and the particular context (e.g., the time of the day). Since it is difficult to predict the events about which will be reported on the news portals, learning a powerful recommendation algorithm is a big challenge.

From the technical perspective load peaks in the usage of news portals are challenging. Fig. 3 shows the number of impressions in the PLISTA contest in the 3rd week of June. The graph shows that in the morning hours and during the lunch break at working days the highest load can be observed. The minima in the curve describing the number of article requests are reached late at night, when most users sleep.

In the PLISTA contest a maximal algorithm's response of 100ms the recommender algorithms must be ensured. Due to limited available computational resources, the recommender algorithms must be optimized to meet the time-constraints also during load peaks. An appropriate tradeoff between the quality of the recommendation results and the computational complexity must be found. An extensive testing of the algorithms supports the analysis of the algorithms under heavy load given a pre-defined hardware setting and helps to find the optimal algorithms for a particular scenario.

The performance of the recommender algorithms is evaluated online based on the click rate. Since the click rate varies with the international news situation and the season making it difficult to reproduce the user behavior. This complicates the fair comparison of several different recommender algorithms. There is a need for an offline testing framework able to repeat pre-recorded streams for measuring different recommender algorithms. A test framework ensures reproducibility of results and speedup the optimization of recommender algorithms.

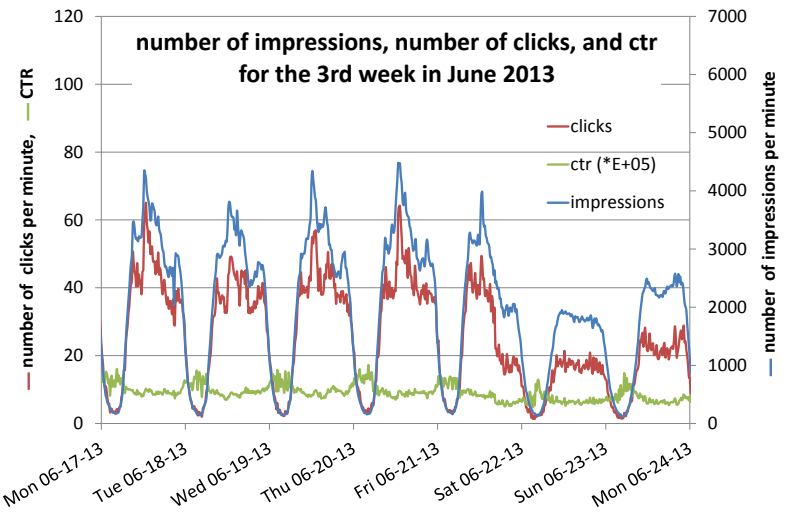

Fig. 3: This figure shows the web traffic on PLISTA's news portals over the course of one week in June. The data is aggregated based on bins having a length of 15 minutes.

4 Approach

In this section we present our recommender algorithms optimized for the news scenario. In the second part of this section we present the implemented testing framework allowing us to evaluate the algorithms on exactly defined hardware.

4.1 Recommender Algorithms

We focus on unpersonalized recommender algorithms. This is based on the observation that the userIDs in the PLISTA contest are noisy (due to insufficient user tracking) and a large number of requests are initiated by new users. We discuss the strength and weaknesses of the implemented algorithm taking into account the recommendation quality and the amount of required resources.

Most popular News articles read by a large number of users are often interesting for a greater audience. Thus, the most popular articles are often a good recommendation. In order to take into account the limited lifecycle of news articles, our most-popular recommender determines the most frequently requested news articles from a pre-defined time window. When a set of recommendations is requested, the recommender determines the most popular news articles based on the user-item interaction statistic from the last hours and returns these articles as recommendations. The idea behind this approach is that users who do not yet know the most popular articles will probably be interested in these articles since they are relevant to almost everyone. However, the algorithms should check, whether the articles have not been seen by the particular user before. An important parameter that must be optimized by the most-popular recommender is the

size of the time window in that the articles are seen as potentially relevant. The older an article gets the more likely it becomes that a user has already read that article or the article does not describe the latest state of an event.

One approach to determine the expected lifetime of an article on a given website is to observe user behavior over a long period of time. The average lifetime of articles in the PLISTA data set lies around 3 hours, but varies between news portal to news portals as well as time of day.

The REDDIT-like recommender The portal REDDIT allows registered users to explicitly rate news articles. Based on the explicit user feedback and the freshness of the articles a ranking is computed. The strategy used by the portal REDDIT can be adapted for the PLISTA news recommendation contest. Instead of counting the "up" and "down" votes (as it is done by REDDIT) we calculate the popularity of an article based on the number of user requests not followed by another article request in the next 30 seconds. We assume that 30 seconds is the minimal time needed to read an interesting article. In order to compute a list of recommendations, we have to aggregate the freshness and the user-news article interaction statistic. We calculate the popularity rank of an article based on the following function: Let $\mathrm{p}(a)$ be the number of readers for article a and $\mathrm{t}(a)$ the publication time of the article a. The rank $r(a)$ can then be computed as follows:

$$\begin{aligned} r(a)_{T,P} \;=\; & \log_{10}\left(\max\left(\mathrm{abs}\left(\mathrm{p}\left(a\right)-T\right),1.0\right)\right) \\ & + \mathrm{sign}\left(\mathrm{p}\left(a\right)-T\right)\cdot\frac{-\mathrm{currentTime}+\mathrm{t}\left(a\right)}{P} \end{aligned}$$

where P determines the influence that the age of the article has on its rank and T determines how many users have to have read an article before it is considered news worthy[5].

Item-to-Item recommendations One approach (that has been utilized when facing a strong variation in user interest) is based on item-based collaborative filtering [10]. This method is used by AMAZON to predict what kind of items users will buy based on items they have in there virtual shopping baskets [7]. Item-to-Item filtering computes the relation between different articles based on common properties. Two articles are related, for example if multiple users have read both of them. The algorithm provides a set of related articles for each known article. If a user requests a new article, this approach looks for related articles to the article that the user was reading before and uses the resulting set of articles to recommend new ones. Item-to-Item methods can be used for the news scenario, despite the fact, that they can only be tracked for short periods of time. Instead of calculating the interests of each user and likely articles per user, the Item-to-Item method uses observed user behavior to calculate article relations. This is more efficient than using user based collaborative filtering, since articles have a longer lifetime than user sessions.

[5] based on `https://github.com/reddit/reddit/blob/master/r2/r2/lib/db/_sorts.pyx`

Most recently created Based on the idea, that the most important property of news articles is that the information is "new", we implement a recommender that suggests the most-recently created news articles. The algorithm can be implemented based on a size-bounded list, containing the most recently added news articles. If a recommendation request is received, the recommender returns the first n articles from the head of the list. The main advantage of this algorithm is that it is based on efficient data structures supporting a large number of requests. In addition, there is a high probability that new articles provide information not yet known to the user. A weakness of the recommender strategy is that neither the context nor the individual user preferences are taken into account.

4.2 Offline load testing

In order to analyze how well the recommender algorithms can cope with the tight time constrains in the PLISTA contest, we implement an offline testing framework. The framework simulates the online contest and allows us to assigns predefined resources to each recommender agent. Since the same interface is used in the offline and the online tests the implemented recommender algorithms can be evaluated in both scenarios without changing implemented recommender components.

The usage of an offline testing framework allows us defining reproducible test cases resulting in a faster, more agile development of recommender systems. Embedding a recommender into a news portal is certainly a more reliable way to test a system, since it is evaluated by real users under realistic conditions. But online testing takes time and not every recommendation that is calculated is seen by a user. Offline tests allow us an efficient benchmarking of recommender algorithms. Instead of waiting hours for 10,000 recommendation request, an offline test can simulate and evaluate 10,000 requests within minutes.

The core of our testing framework is implemented using server component which emulates the open recommendation platform from PLISTA. This allows all teams who participate in this challenge to test their solutions with our framework. The server components are VAGRANT and PUPPET scripts used to automatically set up a virtual machine for deploying recommendation components. This allows us to analyze performance characteristics like CPU usage and memory usage of the system for exactly defined test scenarios.

The measured results describe the recommendation quality, CPU and memory usage of the algorithms for the relevant test cases. The offline tests use log files recorded in the online challenge to simulate the messages from different news portals. The only pre-requisite to the different solutions is that they do not use the identical log files to train or to optimize algorithms to ensure an unbiased assessment.

The log files are converted to requests and updates that the server can send to the different recommender systems. The log files are split into a set of messages that have already been send to the client and a set of messages that will be send in the future. The set of future messages can be used to evaluate the quality of a recommendation.

4.3 Testing scenarios

We analyze three different test scenarios, representing exemplarily three critical situations in handling the message stream in the PLISTA contest.

Rush Hour: We define a test case analyzing how well the algorithms scale with the number of parallel requests. This test case simulates the traffic that news recommenders have to deal with during rush hours. Fig. 3 shows that news websites experience interest spikes and fluctuations in user interactions. Our first test case simulates the worst case scenario of these spikes and can show how resources are consumed in this scenario.

Short in RAM: We define a test case analyzing how well the algorithms scale with amount of available RAM. Similar to the previews test case, this test case can also be used to analyze extreme situations. By incrementally increasing the number of parallel requests and messages, the maximal number of simultaneous connections is experimentally determined. In other words, the maximal number of updates and impressions can be computed given a maximal response time per request and a fixed number of resources that can be used in the tests. This helps to determine how efficient data is processed and stored by the tested recommender algorithms.

Large number of portals: We define a test case analyzing how well the algorithms scale with the number of portals. The more websites a algorithm can handle with the same hardware resources the less money has to be spend to handle more request.

All tests also evaluate the received recommendations based on prediction acuracy ("click through rate").

5 Evaluation

We evaluated all mentioned recommendation methods as well as a user based approach using our testing framework. Each test scenario was repeated multiple times to ensure meaningful results. We discuss the following properties for each algorithm. Given a recommendation the framework searches through all future user interaction that are presented within the log files and counts how often any user has clicked on the recommended article. We call this count global offline click through rate (ctr). The next criteria checks if the user for whom the recommendation was given will ever read the recommended article in the future. We call this criteria "user-specific offline ctr". Every recommended article that cannot be found in the remaining log file is counted as an invalid result. We compute the accuracy (in percent) for each algorithm based on the three criteria discussed above. Table 4 summarizes the evaluation results.

A detailed analysis of the amount of required memory in contexts with a large number of requests per second is shown in Fig. 5. Starting with 1,000 requests per second, we increase the number of request every 10 steps by additional 1,000 requests per second. The measured results show that the MOST RECENTLY CREATED ARTICLE recommender approach has the smallest memory footprint. This algorithm uses an almost constant amount of memory independent from the number of handled requests per second. The amount of memory used by the other three recommender algorithms grows linear with the number of recommendation requests. This can be explained by the memory structures used for storing received requests and impressions.

Methodes	user-specific offline CTR	global offline CTR	cpu usage in %	maximum number of simultanius requests	avarage memory usage
Most-Popular	4 ★★	80 ★★★★	11 ★★★★	5000 ★★★	640 ★★★★
Reddit	2 ★	87 ★★★★★	11 ★★★★	5500 ★★★★	480 ★★★★★
Item -to-Item	8 ★★★	12 ★	12 ★★★★★	1000 ★★	800 ★★★
Most Recent	8 ★★★	70 ★★★	8 ★★	6300 ★★★★★	400 ★★★★★
User -Based	17 ★★★★★	76 ★★★★	100 ★	200 ★	2000 ★

Fig. 4: The table gives an overview of the properties of the evaluated recommender algorithms. The evaluation results show the tradeoff between the recommendation quality and the required amount of resources for the analyzed recommender algorithms. The results help us to decide what algorithm is suitable for a new recommendations scenario.

The detailed analysis of the recommender performance and the required amount of resources allows us to determine the optimal recommender algorithm for a specific recommendation scenario. It helps us to find a powerful recommendation strategy running well even if only limited resources are available.

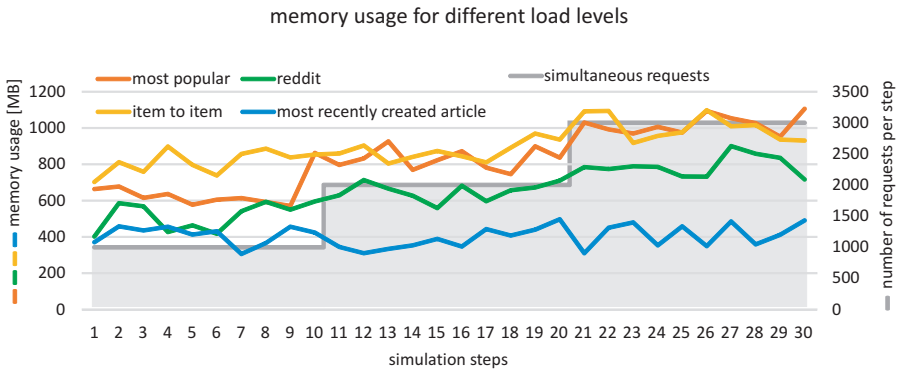

Fig. 5: The figure shows the memory usage in correspondence to the number of simultaneous request for all tested recommendation methods. The number of simultaneous request was increased by 1,000 for every 10 steps in this rush hour simulation.

6 Conclusion and Future Work

In this paper we present and evaluated four different recommender algorithms tailored to the news recommendation scenario. The algorithms have been optimized to provide high-quality recommendation even if only limited resources are available ensuring a fast response time.

We have implemented a testing framework allowing us to evaluate and optimize the recommendation algorithms offline. The offline evaluation results give us valuable

hints for finding optimal parameters settings for critical situations in the online contest (e.g., for handling load peaks). We showed that our framework is capable to predict the resource demands for the recommender algorithms in the analyzed setting. In addition, the presented testing framework supports a more agile development of recommendation algorithms giving us extended options for analyzing our recommender algorithm.

The implemented framework can be used to test any recommender algorithm in the PLISTA challenge. In addition, the framework can also be used in scenarios in that stream-based algorithms must be optimized on different hardware settings. Each part of the framework has been designed to be flexible configurable and customizable. The use of virtual machines allows us to get interesting insights of the resource usage of a system without modifying the physical system. As future work, we plan to extend our framework by adding new quality measures and an support for additional dataformats.

References

1. J. Bobadilla, F. Ortega, A. Hernando, and A. GutiéRrez. Recommender systems survey. *Know.-Based Syst.*, 46:109–132, July 2013.
2. T. Brodt, T. Heintz, A. Bucko, and A. Palamarchuk. ORP Protocol, 2014. online available at: http://orp.plista.com/documentation/download.
3. M. Claypool, A. Gokhale, T. Miranda, P. Murnikov, D. Netes, and M. Sartin. Combining content-based and collaborative filters in an online newspaper. In *Proceedings of ACM SIGIR workshop on recommender systems*, volume 60, 1999.
4. J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, and Others. The YouTube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296. ACM, 2010.
5. D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.
6. Y. Koren and R. M. Bell. Advances in collaborative filtering. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 145–186. Springer, 2011.
7. G. Linden, B. Smith, and J. York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.
8. A. Lommatzsch. Real-time news recommendation using context-aware ensembles. In *Proc. of the 36th European Conference on Information Retrieval*, volume 8416 of *Lecture Notes in Computer Science*, pages 51–62. Springer, 2014.
9. P. Resnick and H. R. Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.
10. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. *Proceedings of the 10th international conference on World Wide Web*, pages 285–295, 2001.