Development and Evaluation of a Highly Scalable News Recommender System

Ilya Verbitskiy[1], Patrick Probst[2], and Andreas Lommatzsch[3]

[1] Complex and Distributed IT Systems, CIT
Technische Universität Berlin, Einsteinufer 17, D-10587 Berlin, Germany
ilya.verbitskiy@campus.tu-berlin.de
[2] Technische Universität Berlin, Straße des 17. Juni 135, D-10623 Berlin, Germany
patrick.c.probst@campus.tu-berlin.de
[3] 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. The development of highly scalable recommender systems, able to deliver recommendations in real time, is a challenging task. In contrast to traditional recommender systems, recommending news entails additional requirements. These requirements include tight response times, heavy load peaks, and continuously changing collections of users and items. In this paper we describe our participation at the CLEF-NewsREEL challenge 2015. We present our highly scalable implementation of a news recommendation algorithm. The developed approach alleviates all the specific challenges of news recommender systems. We use the Akka framework to build an asynchronous, distributable system able to run concurrently on multiple machines. Based on the framework a time window-based, most popular algorithm for recommending news articles is implemented. The evaluation shows that our system implemented using the Akka framework scales well with the restrictions and outperforms the recommendation precision of the baseline recommender.

Keywords: recommender system, scalability, akka framework, most popular recommender, stream-based recommender

1 Introduction

Nowadays, recommender systems are frequently used to support users in navigating through data-rich contents. Helping users to discover relevant information in the overwhelming mess of data is an important task in many applications scenarios. In web applications (such as online shops, forums, news, video and music portals recommendation) algorithms support users in finding new content relevant according to the user context and the current user profile.

Recommending news articles is a hard task due to the highly dynamic environment. News items are updated frequently and user's news preferences are often very diverse and difficult to track. Therefore, recommender algorithms for news portals must be able to process a continuous incoming stream of data in

real time. The complex requirements make the news recommendation scenario an interesting field of research.

1.1 The CLEF-NewsREEL 2015 Challenge

The CLEF-NewsREEL 2015 challenge is a competition held yearly giving researchers the opportunity to analyze and evaluate news recommendation algorithms based on real-life data. The task in the NewsREEL challenge is recommending news articles from different publishers to users. It is organized in cooperation with plista[4], a company that offers recommendations as a service. The company cooperates with different news publishers and discussion board websites. The CLEF challenge provides an online [4] and an offline task [6]. We will discuss the specific requirements of both tasks in the following paragraphs.

The Online Scenario For task 1, plista provides an interface to the online news recommendation service [1]. Participants can register their recommendation engines. The communication favors the HTTP protocol and JSON data descriptions[5]. The system emits four types of messages. *Recommendation requests* expect a list of item references in return. *Impressions* and *item updates* let participants keep up with the system activity. *Error messages* inform about technical issues. Recommendations have to be returned within not more than 100 ms. The system presents them to visitors and tracks the user's reactions. Participants can monitor the performances of their own team as well as the performance of other participants in terms of number of clicks, number of requests, and Click-Through-Rate (CTR). The system visualizes these statistics on a leaderboard.

The Offline Scenario For the offline task (task 2) a previously recorded data set of item updates and event notification is used [5]. Participants ought to predict interactions between users and news articles using a sliding window approach. Additionally, the evaluation considers technical qualities including scalability and responsiveness during load peaks. The offline task allows researchers to analyze the scalability and the performance of the implemented algorithms in a reproducible setting. Thus, different approaches for the concurrent handling of messages as well as strategies for the running algorithms on a cluster of different machines can be evaluated based on the provided dataset.

1.2 The Structure of the Paper

The remaining paper is structured as follows: Section 2 explains the analyzed problems in detail and discusses the addressed challenges. In Section 3 we present

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

[5] See the official protocol description for further details: `http://orp.plista.com/documentation/download`

our approach. We introduce the used framework and explain the developed system architecture. The evaluation of the implemented system is discussed in Section 4. Finally, Section 5 gives an conclusion and an outlook to future work.

2 Problem Description

In contrast to "traditional" recommender systems trained on static datasets, a news recommender system must be able to handle a steadily changing set of items as well as a continuously changing environment. Special requirements emerge as we seek to efficiently provide recommendations in a stream-based scenario. Firstly, the quantity of received recommendation requests can be very high, just as the number of impressions. Since the item set is permanently changing, the model for computing the most relevant items must be adapted frequently. This leads to the question of choosing algorithms able to find relevant items under these circumstances. Furthermore, the load characteristics and the usage of news portals depends on the daytime and involves high peaks during certain times [7]. That is why it is necessary to resist these peak loads. In addition to that, there is only a short time window for answering recommendation requests. This timing constraint is a hard challenge and can impede very time consuming calculations if not handled dexterously as [3,8] pointed out. The approaches for solving these problems are described in the following section.

3 Approach

In this section we motivate the implemented algorithms. Subsequently, we explain the architecture of our system and discuss the applied methods.

3.1 Most Popular Recommender

In order to handle the requirements according to recommendation precision based on steadily changing collections of items, a suitable algorithm must be implemented. In addition, we have to keep in mind the technical requirements making sure that the algorithm fulfills the expectations according to scalability and response time.

Collaborative Filtering has been established as an out-of-the-box recommendation technique. But the technical restrictions disallow its application in real-time settings (cf. [9,2]). Therefore, we searched for alternative approaches. We asked ourselves how traditional "analog" newspapers present their contents. Typically, important news articles cover large parts of the title page. Less significant novelties are spread across the later pages. Following this method, we decided to implement a most popular recommender. We expect users to focus their attention on current as well as important news. We determine articles' importance in terms of their popularity in the last m minutes. Thus, we decided implementing a most-popular recommender taking into account only the most current news articles.

3.2 Technical Requirements

We expect our systems to face hard demands with respect to response time and load peaks inducing scalability problems. We propose to alleviate this issue by means of concurrent message passing. The Akka framework[6] supports concurrently passing messages between so-called actors. Thereby, the system enables us to distribute computations across several machines. Thus, we manage to decrease response times and deal with load peaks. Response times decrease as the system routes requests on nodes with idle resources. The system can handle load peaks by adding additional nodes when necessary.

3.3 Realization of a Distributed Most-Popular Recommendation Algorithm with the Akka Framework

The Akka framework is chosen from a set of different distributed computing frameworks. Being a distributed real time engine, the Apache Storm framework[7] is an alternative but Akka provides a more flexible programming model. Especially in the context of recommender systems a less restrictive programming model is important simplifying the implementation. Our architecture starts with a cluster of computers. The nodes are divided into one master node and n worker nodes (see Figure 1). All requests are distributed along the worker nodes using a load balancer.

Worker Actor The actor is responsible for handling the requests sent to the worker node. The system transforms incoming HTTP requests to Akka messages and forwards them to the actor. When an actor starts, it registers itself on the master node. This dynamically extends the cluster. The actor may have multiple child actors to scale assigned work across the cores of a machine. The actor's main task is to create intermediate rankings of clicked items (e.g. news articles). In our scenario the items are sorted by the amount of clicks received by users. Furthermore, the actor collects information about which items should be filtered and not recommended to users. This is true for the items that the user has already retrieved and the items that have been marked as non-recommendable. The worker actor ensures that the master node keeps track of resource usage (CPU, memory, request throughput) of the workers. Data about the resource usage is sent to the statistics aggregator. The statistics built based on the aggregated data are used for ensuring an optimal automatic scaling (e.g. during peak loads).

Most Popular Merger The merger is the key component in the distributed system. Every fixed time interval the merger asks all registered workers for their rankings and filters. The received rankings and filters are then merged into one

[6] `http://akka.io/`
[7] `https://storm.apache.org/`

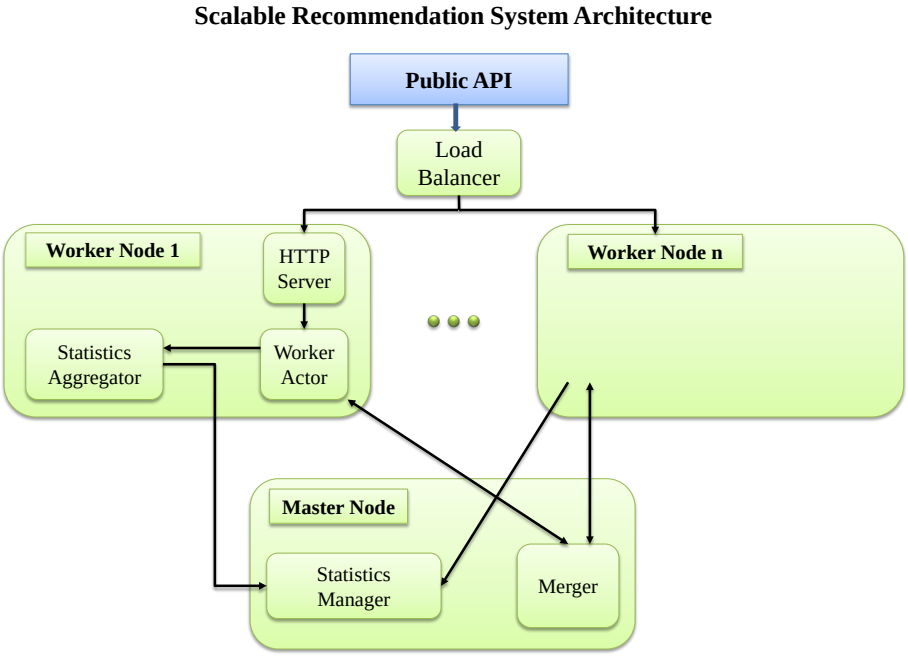

Fig. 1. This scheme visualizes the architecture of our scalable recommender system. A load balancer distributes requests to n machines (worker nodes). The worker nodes calculate intermediate rankings and filters. The intermediate results are sent to the master node. The master node is responsible for merging the intermediate results and sending it back to all worker nodes.

global ranking and filter. Subsequently the aggregated "global" information is sent back to all workers. Thus, every worker will hold the merged results and can respond to recommendation requests using the "global" knowledge. All workers cache the results until updated data are received from the merger node. Our approach has two important advantages. Firstly, the master node is relieved from receiving requests as every worker can answer requests. In addition, the merger runs only periodically (every fixed time interval), preventing the master from becoming a bottleneck. Secondly, the caching of the result on each worker node enables the system to answer requests very quickly (as our evaluation shows).

Statistics Aggregator and Manager The statistics aggregator collects statistics about the worker and passes them every second to the central statistics manager on the master node. The statistics manager prepares the received information for a detailed analysis. This information can be used to track cluster status or to implement an automatic deployment (during load peaks) and undeployment (during low load) of worker nodes.

3.4 Discussion of the Architectural Design

As explained above, the architecture of our system ensures that the system scales well without the master node becoming a bottle neck. But with the increasing amount of worker nodes, the merging becomes more expensive. If increasing the time interval between two merging steps is not an option, this design can be enhanced, whereby the merger will be distributed as well. Therefore multiple mergers are deployed. Each merger is responsible for a set of workers. Thus, these mergers produce intermediate merged rankings. The intermediate rankings are then merged by another merger ("cascading mergers").

In this section we discussed the high-level concepts and strategies. Implementation details are explained in the source code available on GitHub[8].

4 Evaluation

The system design is evaluated live with regard to recommendation precision (CTR). In addition, we evaluated the scalability of our system offline, using the data set provided for the NewsREEL task 2. Based on the previously recorded stream of messages, we analyzed the response time of the system in situations characterized by a high load.

4.1 Online Plista Challenge

We have analyzed the recommendation precision of our system in June 2015. Our algorithm achieved a CTR of 1.37 %. The algorithm outperforms the two baseline recommenders (Table 1). The variance of the measured CTR is low, similar to the baseline algorithms, showing that our algorithm provides robust recommendations.

Table 1. The table shows the CTR and the variance of our most popular and the two baseline algorithms. The results show that our algorithm outperforms the baselines. The CTR variance of our algorithm is similar to the variance of the baseline recommenders.

Algorithm	∅ CTR in June 2015	CTR variance
Our algorithm – Most Popular	1.37 %	$0.00196 \cdot 10^{-4}$
Riemannzeta – baseline	1.21 %	$0.00226 \cdot 10^{-4}$
Gaussiannoise – baseline	0.97 %	$0.00168 \cdot 10^{-4}$

4.2 Offline Load Testing

For the evaluation of the scalability a small cluster of three computers is used. All machines have identical specifications: Intel(R) Xeon(R) CPU E5-2650L v3

[8] `https://github.com/verbit/orp`

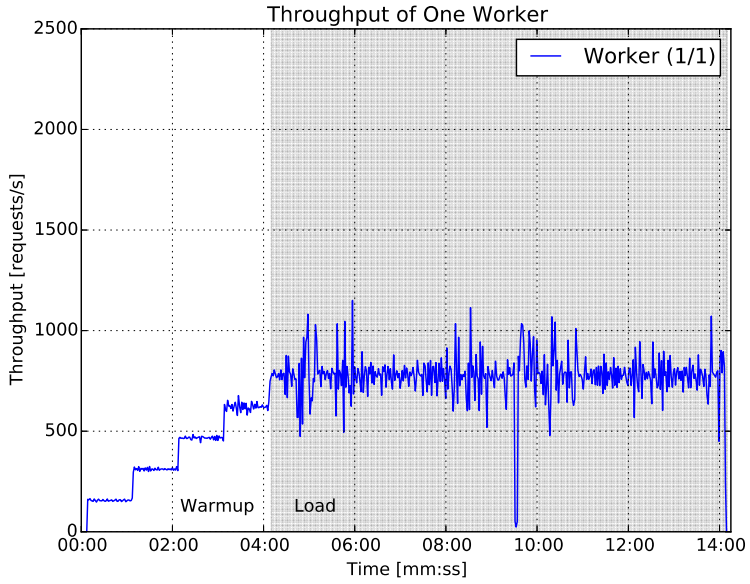

Fig. 2. This figure visualizes the throughput using one worker node during the benchmark. The benchmark starts with a 4-minute long warmup phase followed by a 10-minute long load phase.

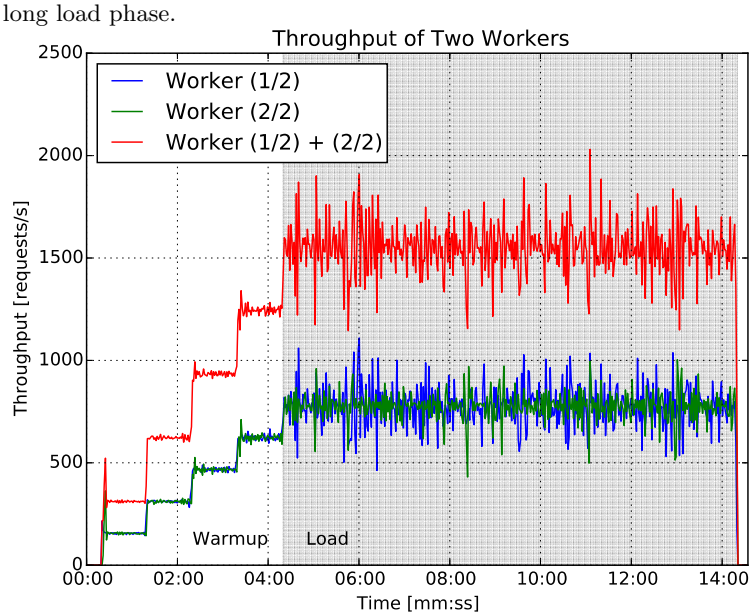

Fig. 3. This figure visualizes the throughput using two worker nodes during the benchmark. The benchmark starts with a 4-minute long warmup phase followed by a 10-minute long load phase.

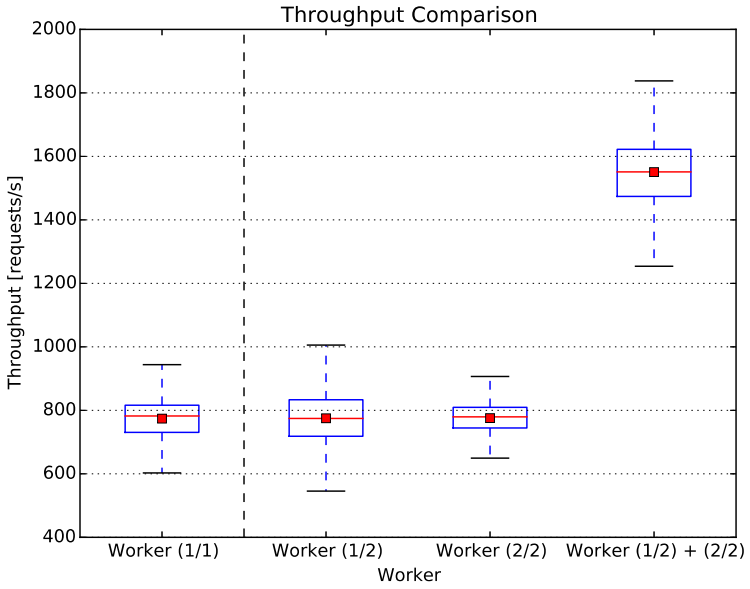

Fig. 4. This box plot compares the throughput of 1 (Worker 1/1) vs. 2 (Worker 1/2 + 2/2) worker nodes during the load phase. A red line indicates the median. A red square visualizes the average. With one node we could reach an average of 774 requests/s, with two nodes an average of 1550 requests/s.

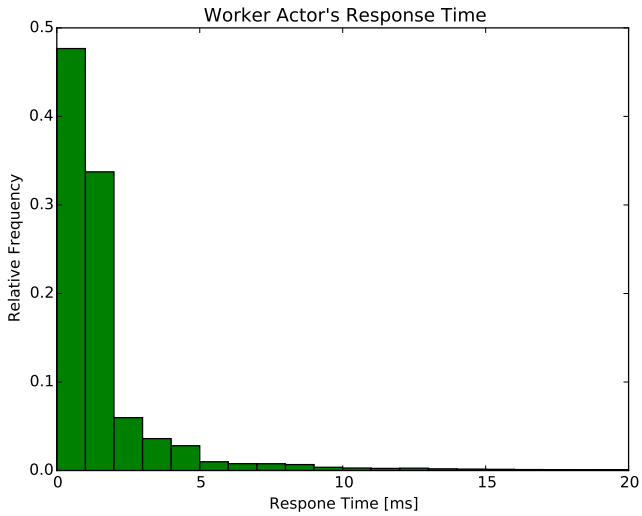

Fig. 5. This histogram shows the distribution of the worker actor's response times. This is the time it takes a worker actor to answer the HTTP server with a recommendation. The response times were measured during the benchmark with two worker nodes. The histogram is cut at the value of 20 ms (exclusive) displaying 99.07 % of all realized response times.

@ 1.80GHz x 2, 2048 MB RAM. An NGINX server on the master node serves as the load balancer. The merging rate is set to 2 seconds to achieve near real time rankings. The cluster is benchmarked with one and two worker nodes, respectively. For the scalability evaluation, the offline data set is used[9].

The requests are sent from a fourth server with a rate high enough to fully utilize the resources of the worker nodes. Figures 2 and 3 show the throughput when using one and two worker nodes, respectively. Figure 4 shows a comparison between one and two worker nodes during the load phase. Figure 5 shows the distribution of the response time in case of two workers. It is important to note that the response time was measured on the server side. Therefore the measurement does not take into account the latencies between the PLISTA servers and the cluster. Keeping this fact in mind it is strongly recommended to deploy the cluster in a location near to the PLISTA servers.

Overall, the offline evaluation shows that our system is able to handle load peaks, ensuring a high throughput.

5 Conclusion and Outlook

In this paper we presented a highly scalable recommender system for news portals. The evaluation shows that the implemented system outperforms the baseline recommenders according to CTR in the online evaluation. In addition, the offline evaluation (task 2) shows that our news recommender system is highly scalable. We doubled the system's throughput as we doubled the worker nodes. The distribution of the response time is similar between one and two workers. It originates from the fact that recommendation request are cached and thus returned instantly. Worker nodes can be deployed dynamically. Therewith, we can resist peak loads. This dynamic allocation uses resources efficiently and reduces costs.

Although the aim of building a highly scalable system is reached, some further improvements can be made. Firstly, our recommendation algorithm could not reach an as high CTR as compared to some other teams. It determines popularity for certain news articles because they are often clicked. Thus it only recommends articles which are already popular. One improvement could be to predict trends. Therefore, we could learn from what had been popular in the past to determine future popular articles or mix in recently created articles. Another promising improvement is applying different recommendation algorithms for certain publishers. Especially for discussion forums (less focused on the latest news) content-based recommender algorithms seems to be more powerful approaches [7].

[9] `http://data.dai-labor.de/corpus/clef-newsreel-2015/raw/CLEF-2015-Task2-Json07.tar`

Acknowledgement

This research is supported by funding from the European Commission's 7th Framework Program (FP7/2007-2013) under grant agreement number 610594.

References

1. Torben Brodt and Frank Hopfgartner. Shedding light on a living lab: The clef newsreel open recommendation platform. In *IIiX'14: Proceedings of the Information Interaction in Context Conference*, pages 223–226. ACM, 08 2014.
2. Fidel Cacheda, Víctor Carneiro, Diego Fernández, and Vreixo Formoso. Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems. *ACM Transactions on the Web (TWEB)*, 5(1):2, 2011.
3. Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280. ACM, 2007.
4. Frank Hopfgartner, Benjamin Kille, Andreas Lommatzsch, Till Plumbaum, Torben Brodt, and Tobias Heintz. Benchmarking news recommendations in a living lab. In *CLEF'14: Proceedings of the 5th International Conference of the CLEF Initiative*, LNCS, pages 250–267. Springer Verlag, 09 2014.
5. Benjamin Kille, Frank Hopfgartner, Torben Brodt, and Tobias Heintz. The plista dataset. In *NRS'13: Proceedings of the International Workshop and Challenge on News Recommender Systems*, ICPS, pages 14–22. ACM, 10 2013.
6. Benjamin Kille, Andreas Lommatzsch, Roberto Turrin, Andras Sereny, Martha Larson, Torben Brodt, Jonas Seiler, and Frank Hopfgartner. Stream-based recommendations: Online and offline evaluation as a service. In *Proceedings of the 6th International Conference of the CLEF Initiative*, CLEF'15, 2015.
7. Andreas Lommatzsch and Sahin Albayrak. Real-time recommendations for user-item streams. In *Proc. of the 30th Symposium On Applied Computing, SAC 2015*, SAC '15, pages 1039–1046, New York, NY, USA, 2015. ACM.
8. Alan Said, Alejandro Bellogín, Jimmy Lin, and Arjen de Vries. Do recommendations matter?: news recommendation in real life. In *Proceedings of the companion publication of the 17th ACM conference on Computer supported cooperative work & social computing*, pages 237–240. ACM, 2014.
9. Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *The Journal of Machine Learning Research*, 10:623–656, 2009.