# Towards using Cached Data Mining for Large Scale Recommender Systems

Swapneel Sheth, Gail Kaiser
Department of Computer Science, Columbia University, New York, NY 10027
{swapneel, kaiser}@cs.columbia.edu

***Abstract*—Recommender systems are becoming increasingly popular. As these systems become commonplace and the number of users increases, it will become important for these systems to be able to cope with a large and diverse set of users whose recommendation needs may be very different from each other. In particular, large scale recommender systems will need to ensure that users' requests for recommendations can be answered with low response times and high throughput. In this paper, we explore how to use caches and cached data mining to improve the performance of recommender systems by improving throughput and reducing response time for providing recommendations. We describe the structure of our cache, which can be viewed as a prefetch cache that prefetches all types of supported recommendations, and how it is used in our recommender system. We also describe the results of our empirical study to measure the efficacy of our cache.**

***Keywords*-data mining, caching, recommender systems, empirical study**

## I. Introduction

Recommender systems have become increasingly commonplace. Recommender systems are being used in a variety of domains such as recommending music we may like [1], [2], things we might like to buy [3], and friends we may know [4]. There have also been many recommender systems targeted towards specialized domains such as software engineering [5], [6], [7], [8] and medicine [9]. While there has been a lot of work in the academic community on various aspects of recommender systems such as recommendation algorithms [10], [11] and implications of social networks in recommender systems [12], [13], there has been very limited work that has explored the use of caches and cached data mining to improve the performance of recommender systems by increasing throughput and reducing response time for providing recommendations. This will be of particular concern as these recommender systems become even more popular and their user and fan base grow. With a large number of users, there are two specific issues that recommender systems would have to deal with - how to generate recommendations efficiently from a large set of data and how to provide these recommendations efficiently to a diverse set of users, where each user's requirements for recommendations are different from the others.

In this paper, we describe how we use cached data mining to answer users' queries and provide recommendations in a very efficient way. We describe our background and motivation in the next section. Section III describes in detail the recommendations provided by our system and how we use cached data mining. Section IV describes our empirical study and results. Finally, we conclude the paper with a discussion of the related work in Section V.

## II. Background and Motivation

We are working with researchers at Columbia University's Center for Computational Biology and Bioinformatics (C2B2), particularly its MAGNet (Multiscale Analysis of Genomic and Cellular Networks) Center to explore new ways in which researchers in computational biology and bioinformatics can collaborate to share data, analyze results, and share knowledge. Our approach is based on social networking metaphors for collaborative work where users can ask questions such as: Who likes movies that I like?; What food and wine pairings go well together?; What book would I like given that I like this book?

Our implementation of this approach is a system called "genSpace" [14], which uses collaborative filtering to provide recommendations to users. genSpace is a plugin to an open-source Java-based platform for integrated genomics called "geWorkbench" [15]. Using geWorkbench, researchers in computational biology and bioinformatics can load in data sets such as DNA, protein, and gene sequences. They can then run complex analysis tools such as filtering, normalization, clustering, and pattern detection. There are over 50 such analysis tools supported by geWorkbench, and each tool has many different runtime parameters. Choosing the right tool to use and the sequences in which to use these tools (workflows) can be very daunting, especially to new users. One substantial way that we diverge from and expand upon the collaborative filtering provided by popular websites is that we address the *ordering* among related activities conducted in sequence, i.e., as a **workflow**. E.g., a common workflow in geWorkbench is to run the ARACNe (Algorithm for the Reconstruction of Accurate Cellular Networks) analysis [16] followed by the MINDy (Modulator Inference by Network Dynamics) analysis [17]. Issues stemming from this ordering concern are, however, outside of the scope of this paper.

genSpace aims to flatten the learning curve and enable users to quickly become productive. In particular, for users who do not know where to start, it recommends the most popular three tools and workflows. For users already familiar with using one or more tools in their standalone form outside geWorkbench, it recommends the most popular workflow that starts with or includes a particular tool and the best tool to run

next given that you've just run a particular tool. In order to achieve this, we log users' activities as they use geWorkbench and send the logs to a central server, where data mining and collaborative filtering techniques generate these and other kinds of recommendations.

Currently, our genSpace recommender system is modest in size. Our database has about 10000 rows of data from around 150 distinct users. Since we anticipate a significant increase in usage when geWorkbench soon introduces a Web-based client, we wanted to study how our system would respond to and/or if it could cope with a large increase in the number of users and user data.

In this paper, we discuss how we use cached data mining for providing recommendations to users in genSpace. We also describe an empirical study highlighting their benefits and improvements to the response time and throughput to user queries.

## III. Cached Data Mining and genSpace Recommendations

### A. Recommendations in genSpace

In genSpace, we support two different kinds of recommendations - static and dynamic.

*1) Static Recommendations:* Static Recommendations are those recommendations that do not depend on the current activity of the user. Typically, such recommendations follow a "pull" model where a user explicitly asks for these recommendations. Examples of such recommendations include the top tools, the top workflows, and the most popular workflow that includes or starts with a particular tool.

*2) Dynamic Recommendations:* Dynamic Recommendations are those recommendations that do depend on the current activity of the user. Typically, such recommendations follow a "push" model where the system automatically pushes these recommendations to the user. Examples of such recommendations include suggesting the best analysis tool to run next based on what the user has done so far and suggesting popular superflows (workflows that include the user's current workflow).

All these recommendations are generated using data mining to derive patterns and trends from the user data.

### B. genSpace Caching

genSpace has a server-side cache that supports pushing or pulling recommendations to/from the users. It can be viewed as a prefetch cache that prefetches all types of recommendations supported by the system. It is not a traditional cache where items are added to the cache when they are requested and there exist notions of cache hits, cache misses, cache replacement policies and so on. Every recommendation that we need will be present in the cache and we won't need to go to the database for any information. Due to this, we do not have the problem of a cache miss and we do not need to worry about cache replacement and by definition, our hit rate and recall is 100%. When the genSpace server starts up, the genSpace cache is generated using a combination of SQL queries and stored procedures from our SQL database backend that stores all the user data. The cache is periodically re-generated as needed - currently, every day. If we did not have a cache, we would have to run the SQL queries on demand every time a user request came in for recommendations. We would also have to re-run the same query multiple times if different users asked for the same set of recommendations.

We also address the problem of concept drift [18] where workflows performed by users six months may not be so relevant today. E.g., after publication of major findings that involved a form of analysis that was previously rare or after upgrading to a new geWorkbench release that integrates additional tools or even for no known reason, many users shift their usage patterns. We use an exponential time-decay formula [19] to weigh recent user data more heavily. This weighting is done each time the cache is generated.

After weighting the data, the static recommendations are computed and stored in the cache. We build an index for each analysis tool found in the log data, to represent the following information: the number of times this tool has been used, the number of times this tool has been used as a workflow head, the most popular tool before and after this tool in workflows, the most popular workflows containing this tool, and all workflows that include this tool. This cached index uses hashing based on the tool name to give us constant time lookup for tool-specific information. Finally, a tree-based index of popular workflows aids in the dynamic recommendations. All these parts together comprise the genSpace Caching System.

genSpace usually gets around 10-20 new logs every day and due to this, we re-generate our cache every day. As the number of users for our system increases, concept drift may take place on shorter timescales and we may need to re-generate the cache more often to deal with it. The re-generation frequency is easily configured on our server and will be ramped up as needed. However, more studies need to be done to measure and fully understand the effect of concept drift on cache re-generation and this is part of our future work. The next section contains some empirical results on the time required to re-generate our cache.

Finally, due to the structure of the genSpace cache, it can only support the currently existing types of recommendations. If we wanted to support additional types of recommendations, the cache would need to be augmented with the appropriate information. E.g., we currently don't support providing recommendations based on the file-type on which the analysis tools are run. To support this, our cache would need to store information regarding the file-types for the analyses.

## IV. Empirical Study

The genSpace cache has already been deployed in our production system although it may not be needed currently due to the modest number of users. In order to understand the prospective real-world improvements due to our cache, we carried out an empirical study. For our study, we varied the number of rows of our database (in the range of around 3500, 10000, 100000, and one million) and measured its impact

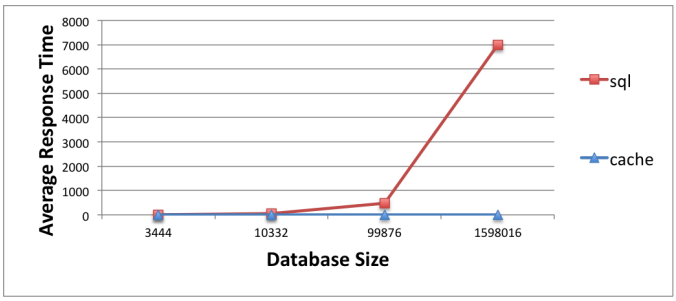


Fig. 1. Database Size vs. Average Response Time, for "Get Most Popular Workflow Heads"

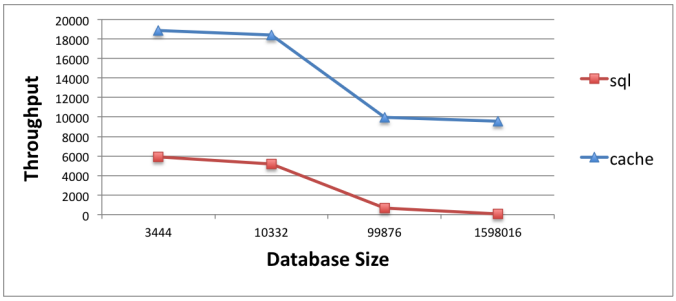


Fig. 2. Database Size vs. Throughput, for "Get Most Popular Tools"

on average response time and throughput to user queries for recommendations. We simulated 1000 concurrent users requesting recommendations. We also compared these results to the results obtained if we did not have a cache and used SQL queries instead every time for generating recommendations.

We used Apache JMeter [20] for load testing our server and measuring performance. The genSpace server, including the cache, is implemented in Java. Our server and client machines were common Windows XP machines with no non-essential system processes running and had more than 2GB of surplus RAM available.

Figure 1 shows the plot of the database size (in number of rows) versus the Average Response Time for a recommendation that gets the most popular workflow heads, i.e., tools at the start of a workflow. The red line with squares as data points shows the response time when using SQL queries-on-demand and the blue line with triangles as data points shows the response time when using our cache. As shown in the figure, as the size of the database increases, the response time using SQL queries-on-demand increases by a large amount. Meanwhile, the response time using our cache remains roughly constant. This shows that as the database size increases, using SQL queries-on-demand is not practical whereas using the cache enables us to answer users' queries in roughly the same time regardless of the database size.

Figure 2 shows the plot of the database size (in number of rows) versus the Throughput for a recommendation that gets the most popular tools in the system. The red line with squares as data points shows the response time when using SQL queries-on-demand and the blue line with triangles as data points shows the response time when using our cache. From the graphs, we see that the cache outperforms the SQL queries-on-demand approach by a factor of at least 3 to as much as 200 as database size increases.

Most of the static and dynamic recommendations mentioned in Section III were part of the empirical study and our results were similar to the ones shown above and generally show that using the cache improves the throughput and reduces the response time.

We also measured how long our cache generation process takes. As mentioned earlier, we currently re-generate our cache every day and it might be necessary to re-generate our cache more often. In our study, it takes around ten seconds to generate the cache for a database that has around one million rows and about 100 seconds for a database that has around ten million rows. Thus, even if our database size increases by a large amount, we can still manage to re-generate the cache periodically as often as needed.

## V. Related Work

To the best of our knowledge, there is very little in the published literature discussing caches for recommendation systems; in fact, we found exactly one paper that discusses this. Qasim et al. [21] propose a general solution using active caches for providing recommendations in all types of recommender systems. Active Caches are caches that can answer neighborhood queries for recommendations, i.e., similar queries to a given query and act as limited query processors. Due to this, the approach proposed by Qasim et al. is limited to neighborhood queries for recommendations and will not work well, in general, for all kinds of queries and focusing on just neighborhood queries may not improve overall system performance by a significant amount. As recommender systems become increasingly popular, there might exist a very diverse user base that is interested in different kinds of recommendations from the system.

In fact, as mentioned in their paper, due to overheads of caching, the system might actually perform worse than having no cache. Our genSpace solution, on the other hand, is not limited to neighborhood queries for recommendations and works well for all kinds of recommendations supported by our genSpace system. This is because our system, unlike the one mentioned by Qasim et al., is a prefetch cache that prefetches all recommendations; all user recommendations can be answered using the cache, rather than just the neighborhood ones. Of course, as our system evolves and new types of recommendations are added, we would need to enhance our cache to support those as well.

Further, Qasim et al., in the experimental section of their paper, focus on the Hit Ratio, Recall, and Efficiency of computing the cache. While these metrics are important, we feel it would more meaningful to see what this translates to, from a user's point of view. A typical user is not directly concerned about hit ratio and recall; rather, he is usually directly concerned with the latency and response time for these recommendations. Our empirical study shows that using caches in genSpace has significantly improved the throughput

and reduced the response time for recommendations, thus improving the overall user experience. Also, as we use a prefetch cache that prefetches all types of recommendations supported by the system, by definition, the hit ratio and recall for our system is 100%.

## VI. Conclusion

We have described how we use prefetch caching in our genSpace recommender system. We have also described the structure of our cache, which can be viewed as a prefetch cache that prefetches all types of supported recommendations, and our empirical study that shows the advantages of using our cache, which improves throughput and reduces response time for recommendations. We believe that the use of such caches will prove very beneficial to recommender systems, particularly as the number of users of such systems grow and the system needs to support the diverse needs of its users, where different users are interested in very different kinds of recommendations from the system and the recommendations they request do not overlap.

## Acknowledgments

The authors would like to thank Aris Floratos, Kiran Keshav, and Zhou Ji for their guidance and assistance with genSpace. We would also like to thank Cheng Niu, Joshua Nankin, Eric Schmidt, and Yuan Wang for their assistance in the implementation of the genSpace cache and in the empirical study to measure its efficacy. The authors are members of the Programming Systems Lab, funded in part by NSF CNS-0905246, CNS-0717544, CNS-0627473 and CNS-0426623, and NIH 1 U54 CA121852-01A1.

## References

[1] "Pandora Radio," http://www.pandora.com.
[2] "Last.fm," http://www.last.fm.
[3] "Amazon.com," http://www.amazon.com.
[4] "Facebook," http://www.facebook.com.
[5] A. Begel, K. Y. Phang, and T. Zimmermann, "Codebook: discovering and exploiting relationships in software repositories," in *ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*. New York, NY, USA: ACM, 2010, pp. 125–134.
[6] F. McCarey, M. Cinnéide, and N. Kushmerick, "Rascal: A recommender agent for agile reuse," *Artificial Intelligence Review*, vol. 24, no. 3, pp. 253–276, 2005.
[7] R. Holmes, T. Ratchford, M. P. Robillard, and R. J. Walker, "Automatically recommending triage decisions for pragmatic reuse tasks," in *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering*, 2009, pp. 397–408.
[8] C. Murphy, G. E. Kaiser, K. Loveland, and S. Hasan, "Retina: Helping Students and Instructors Based on Observed Programming Activities," in *Proc. of the 40th ACM SIGCSE Techn. Symp. on CS Education*, March 2009, pp. 178–182.
[9] "WebMD Symptom Checker," http://symptoms.webmd.com.
[10] Y.-J. Park and A. Tuzhilin, "The long tail of recommender systems and how to leverage it," in *RecSys '08: Proc. of the 2008 ACM Conf. on Recommender systems*, 2008, pp. 11–18.
[11] J. Zhang and P. Pu, "A recursive prediction algorithm for collaborative filtering recommender systems," in *RecSys '07: Proc. of the 2007 ACM conference on Recommender systems*, 2007, pp. 57–64.
[12] V. Zanardi and L. Capra, "Social ranking: uncovering relevant content using tag-based recommender systems," in *RecSys '08: Proc. of the 2008 ACM Conf. on Recommender systems*, 2008, pp. 51–58.
[13] W. Geyer, C. Dugan, D. R. Millen, M. Muller, and J. Freyne, "Recommending topics for self-descriptions in online user profiles," in *RecSys '08: Proc. of the 2008 ACM conference on Recommender systems*, 2008, pp. 59–66.
[14] C. Murphy, S. Sheth, G. Kaiser, and L. Wilcox, "genSpace: Exploring Social Networking Metaphors for Knowledge Sharing and Scientific Collaborative Work," in *1st International Workshop on Social Software Engineering and Applications (SoSEA)*, September 2008, pp. 29–36.
[15] A. Califano, A. Floratos, M. Kustagi, and J. Watkinson, "geWorkbench: An Open-Source Platform for Integrated Genomics," http://www.geworkbench.org.
[16] K. Basso, A. Margolin, G. Stolovitzky, U. Klein, R. Dalla-Favera, and A. Califano, "Reverse engineering of regulatory networks in human B cells," *Nature genetics*, vol. 37, no. 4, pp. 382–390, 2005.
[17] K. Wang, M. Saito, B. Bisikirska, M. Alvarez, W. Lim, P. Rajbhandari, Q. Shen, I. Nemenman, K. Basso, A. Margolin *et al.*, "Genome-wide identification of post-translational modulators of transcription factor activity in human B cells," *Nature biotechnology*, vol. 27, no. 9, pp. 829–837, 2009.
[18] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine Learning*, vol. 23, no. 1, pp. 69–101, 1996.
[19] E. Cohen and M. Strauss, "Maintaining time-decaying stream aggregates," in *Proc. of the 22nd ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems (PODS)*, 2003, pp. 223–233.
[20] Apache, "Jmeter," http://jakarta.apache.org/jmeter/.
[21] U. Qasim, V. Oria, Y.-F. B. Wu, M. E. Houle, and M. T. Özsu, "A partial-order based active cache for recommender systems," in *RecSys '09: Proceedings of the third ACM conference on Recommender systems*. New York, NY, USA: ACM, 2009, pp. 209–212.