A Partial-Order Based Active Cache for Recommender Systems

Umar Qasim
New Jersey Institute of Technology
Newark, New Jersey, USA
muq2@njit.edu

Vincent Oria
New Jersey Institute of Technology
Newark, New Jersey, USA
oria@njit.edu

Yi-Fang Brook Wu
New Jersey Institute of Technology
Newark, New Jersey, USA
wu@njit.edu

Michael E. Houle
National Institute of Informatics
Tokyo, Japan
meh@nii.ac.jp

M. Tamer Özsu
University of Waterloo
Waterloo, Ontario, Canada
tozsu@cs.uwaterloo.ca

ABSTRACT

Recommender systems aim to substantially reduce information overload by suggesting lists of similar items that users may find interesting. Caching has been a useful technique for reducing stress on limited resources and improving response time. In this paper, we propose an 'active caching' technique for recommender systems based on a partial order approach that not only benefits from popularity and temporal locality, but also exploits spatial locality. This approach allows the processing of answers to neighboring non-cached queries in addition to the reporting of cached query results. Test results for several data sets and recommendation techniques show substantial improvement in the cache hit ratio and computational costs, while achieving reasonable recall rates.

1. INTRODUCTION

By suggesting useful next steps for an information search, recommender systems have a huge potential for reducing information overload. It is expected that recommender systems will be an integral part of many future applications [14]. The two most common technologies for recommender systems are collaborative filtering (CF) and content-based filtering (CB). The former generates recommendations based on user evaluations and preferences, while the latter provides recommendations based on the similarities of the content [9]. Recommender systems, being computationally-intensive interactive applications, are highly vulnerable to access latency, and as such require latencies on the order of at most several seconds [11].

One well-established optimization technique with the potential of improving the overall performance of information retrieval in general, and recommender systems in particular, is that of caching. A cache is a temporary storage area where popular or recently accessed data is stored for rapid access. For traditional (passive) caching, the benefits of a cache can only be achieved if the objects in the cache have a high probability of being accessed in the future. The temporal locality principle states that this probability is higher for recently-referenced data items. If as for many domains the data follows Zipf's probability distribution, the probability is also higher for often-referenced (popular) data items. The performance of caching has shown its effectiveness for web search, where it has been applied to cope with redundant data transmission, limited bandwidth availability, and slow response times [7, 11, 13].

The performance of a cache can be further improved if it is able to answer queries other than those whose results are present in the cache, particularly if the stored result sets overlap significantly with the result of the current query. The spatial locality principle suggests that such 'neighbor' queries have a higher probability of being posed in the near future [1]. Caches that can answer neighboring queries would operate as limited query processors, and as such are referred to in the research literature as 'active caches'. Practical active caching solutions have already been proposed for information retrieval using keyword-based queries [3, 10].

In this paper we present an active caching method for top-k similarity queries that is capable of efficiently synthesizing answers for 'neighboring' queries using the contents of the cache. This approach uses the principle of monotonicity of rank order in partial-order lists to construct results for non-cached queries. Experimental results confirm that our approach is comparable in performance with a baseline approach that requires the more restrictive assumption of a distance function defined over the data domain. We test our method against the traditional caching approach using several recommendation techniques. The results show substantial improvement in terms of hit ratio and computational cost while achieving reasonable recall values. Our active caching approach is significant for several reasons. First, we focus on the domain of recommender systems for which, to the best of our knowledge, no research has been performed in the past. Second, our active caching design uses monotonicity in the partial order lists obtained from the cached query results to synthesize new query results. Finally, we provide a general active caching solution for all types of recommender systems while utilizing only list ranking information.

2. PARTIAL-ORDER BASED ACTIVE CACHING SOLUTION

We will take a simplified view of a recommender system as one that accepts an object as the query, and returns a ranked list of objects that are similar to it. More formally, let S be a ranked list drawn from some database D. For every object $o \in S$, we assume a unique ordering $(o_1, o_2, \ldots, o_{|S|})$ of the objects of S, where $i < j$ implies that o_i is deemed more relevant or similar to o than o_j. With respect to o, the rank of object o_i ranges from 1 to $|S|$, and will be denoted by $rank(o, o_i)$. In practical settings, the object most relevant to o is generally o itself. Nevertheless, unless otherwise stated, we will not require that $rank(o, o) = 1$.

2.1 Cache Implementation

Consider now the situation in which a main-memory cache $\mathcal{C}(C, k) \triangleq \{\mathrm{Q}(o, k) : o \in C\}$ of top-k relevant sets is available for each object in a given subset $C \subseteq S$, for some fixed $k \in [1, |S|]$. If each of the relevant sets is maintained as a list of objects sorted from most relevant to least relevant, the collection of relevant sets $\mathcal{C}(C, j)$ is also readily available for all $1 \leq j < k$. We will accordingly refer to $\mathcal{C}(C, j)$ as a *sub-cache* of $\mathcal{C}(C, k)$.

For a given $u \in S$, reverse relevant sets for u can also be defined with respect to the cache $\mathcal{C}(C, k)$ as follows:

$$\mathrm{Q}_C^{-1}(u, k) \triangleq \mathrm{Q}^{-1}(u, k) \cap C = \{o \in C : u \in \mathrm{Q}(o, k)\}.$$

The collection of all such reverse lists taken over all choices of $u \in S$ will be referred to as the *reverse cache* corresponding to $\mathcal{C}(C, k)$, and will be denoted by $\mathcal{C}^{-1}(C, k) \triangleq \{\mathrm{Q}_C^{-1}(o, k) : o \in C\}$. For the remainder of the paper, we will use the terms *forward cache* and *forward relevant set* to refer to the original cache $\mathcal{C}(C, k)$ and its lists, and the term *cache* loosely to refer to the set C taken together with its forward and reverse relevant sets. For any object having a forward relevant set available in the cache, the forward list can serve as the result of future similarity queries, as in any traditional caching approach. For an object without a forward relevant set but having a reverse relevant set available in the cache, our proposed algorithm will synthesize a query result from the cache. If neither a forward relevant set nor a reverse relevant set is available for an object, the results of any similarity queries based at this object will be fetched from the database. Figure 1 shows a simple example of a cache structure with set size of 10, cache size of 3, and reverse lists constructed for the objects stored in the forward lists.

2.2 Partial-Order Based Approach

We now describe our method for active caching of recommender system results. The method does not make use of actual similarity values, instead relying only on the rank order

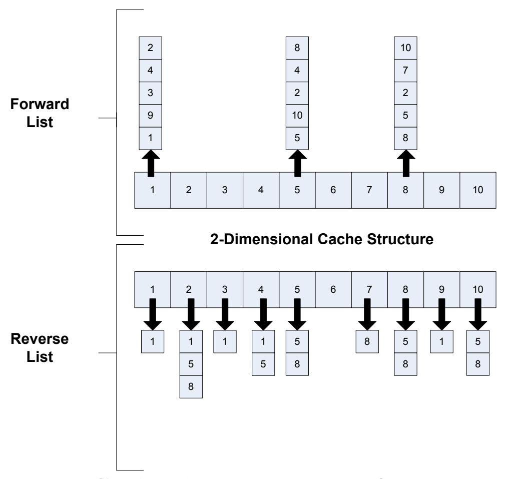

Figure 1: Cache data structures for a set of 10 objects in the 2-D plane. Top-5 lists are cached for 3 objects and reverse lists constructed accordingly.

information of query result lists. Our approach uses partial-order list characteristics to compute results for queries that are related to others explicitly stored in the cache. In order to assess the impact of using rank information instead of distance information for active caching, we also describe and implement a distance-based variant of our method.

Let D be an object domain and l a ranked list objects drawn from D. Let $rank(l, o)$ denote the rank of the object o in the list l. Given a query object o for which no result is cached, our active cache method first generates two partial orderings from each cached query result list containing o:

- the *suffix list* $suff(l)$, defined as the sublist of l consisting of items with ranks strictly higher than $rank(l, o)$; and
- the *prefix list* $pref(l)$, defined as the sublist of l consisting of items with ranks strictly less than $rank(l, o)$, taken in reverse order.

With respect to these two partial orderings, we define the rank $rank_o(l, u)$ of object $u \in l$ with respect to o to be the rank that u holds in either $pref(l)$ or $suff(l)$ — note that u cannot simultaneously be contained in both. More precisely, this rank is defined to be the difference

$$rank_o(l, u) \triangleq |rank(l, u) - rank(l, o)|.$$

We next show how a query result can be synthesized from the cached results of other queries. Let l be a cached forward list with $rank_o(l, o_1) > rank_o(l, o_2)$, for some pair of objects $o_1, o_2 \in D$. If for a top-k query-by-example on o an active cache method generates a ranked result list l' containing both o_1 and o_2, then ideally we would expect the ranks of these objects in the synthesized result to satisfy $rank(l', o_1) > rank(l', o_2)$. The active caching method proposed in this paper resolves conflicts in the partial order information by aggregating the rank information across all suffix lists and prefix lists available from the cache. The aggregation can be performed with respect to such standard operations as *min*, *max*, and *avg*.

Algorithm Query
Input: query object o, result size k;
Output: ranked top-k query-by-example result list *Result*.

```
1. Initialization:
   (a) Assign list $L \leftarrow \mathrm{Q}_C^{-1}(o,k)$. $L$ refers to the cached
       forward lists containing object $o$.
   (b) Initialize result object candidate set $candset \leftarrow \emptyset$,
       and final result object set $Result \leftarrow \emptyset$.
2. For all lists $l \in L$ do:
   (a) For all objects $u \in l$ do:
       i. If $u \notin candset$ then insert $u$ into $candset$, and
          initialize rank value multiset $rankset(u) \leftarrow \emptyset$.
       ii. Insert an instance of the rank value $rank_o(l,u)$
           into the multiset $rankset(u)$.
3. For all objects $u \in candset$ do:
   (a) Generate a score $s$ for $u$ by aggregating the rank
       values stored in $rankset(u)$ using the chosen aggregation
       function (for example, max, min, avg, etc.).
   (b) Insert the object-score pair $(u,s)$ into the result
       list Result.
4. Sort the entries in the list Result according to their
   score values, and return the objects of top $k$ object-score
   pairs. Ties can be broken arbitrarily, with the
   exception that $o$ is given priority over any other object
   $w \neq o$ in $D$.
```

If the object domain D is embeddable in a metric space M with distance metric d, and these distance values are readily computable, a distance-based variant of the proposed query algorithm is possible: each instance of the rank $rank_o(l,u)$ can simply be replaced by the distance value $d(o,u)$. The use of distance values in place of rank values can reasonably be expected to lead to better performances in practice; however, as has been previously noted, a distance-based formulation may not always be possible, especially in the context of collaborative filtering systems.

3. EVALUATION

The performances of the proposed method is evaluated in terms of three measures: *hit ratio*, *recall* and *efficiency*. Given a schedule of queries, the hit ratio is defined to be the proportion of queries that can be answered from the cache. For our method, a hit occurs for any query based at an object for which a forward list or reverse list is cached. Consider now the item set retrieved by any given top-k query operating on the cache. The *recall* of the query is defined as the proportion of this result that would also appear in a top-k query applied to the full database. The *efficiency* is measured in terms of execution time to process all the queries in the dataset, whether answered from the forward cache, the reverse cache or the database.

3.1 Datasets

We used three datasets to test the performance of our proposed solution. The first dataset is based on the Amsterdam Library of Object Images (ALOI)[4]. The full dataset consists of 110250 images of 1000 common objects taken from a number of different angles under different lighting conditions (for a detailed description of how the vectors were produced, see [2]). We used the SASH approximate similarity search structure [6] with Euclidean distances to compute a k-NN matrix for this image collection. The Reuters Corpus Volume 1 (RCV1) is an archive consisting of 802,352 newswire articles [8]. Again, we used a SASH to compute a k-NN matrix using the cosine similarity measure. Both the ALOI and RCV1 datasets were used to represent content-based recommender systems. As an example of collaborative filtering, we used the MovieLens dataset, which consists of 100,000 ratings (from 1 to 5) for 1682 movies from 943 users [5]. We opted for a model-based approach to generate recommendations, and used MultiLens [12] to generate user-to-user and count models for providing recommendations.

Both the distance-based and rank-based active caching methods were implemented in Microsoft C♯, and tested on IBM desktop with processor speed of 3.0 GHz and 2 GB of RAM. Microsoft SQL Server was used to store forward and reverse cached lists.

3.2 Experimental Results

For the following experiments, various proportions of the datasets were selected uniformly at random for inclusion in the cache: for each item selected, its top-k similarity query result was generated and stored. The experimental results observed for the distance-based approach was in all instances comparable to that of the rank-based approach. In this version of the paper, we provide results only for the rank-based active caching method using the *max* aggregation function.

3.2.1 Hit Ratio

The partial-order based active caching results achieved a substantially-high hit ratio when compared with traditional caching results, for all datasets tested. Figure 2 shows a hit ratio comparison between active caching approach for each recommendation technique and for traditional caching with various data loads. Much higher hit ratios were achieved for the content-based cases than the collaborative filtering cases. This may be due to the small size of the MovieLens dataset in both the number of users and the number of movies, or to the high variability of the numbers of movie ratings provided by the users.

3.2.2 Recall

For the various datasets and caching techniques, average recall rates were calculated by posing similarity queries based at every object of the dataset, with the exception of the Reuters dataset where only the first 10,000 objects served as the basis of queries. The tests were conducted for several different choices of the proportion of cached items. The query size and cached list size were both fixed at $k = 10$. Our proposed active caching solution attained high recall values across the various cache sizes, as shown in Figure 3. Again, the ALOI and RCV1 datasets showed better performance, possibly due to their larger sizes and more uniform distribu-

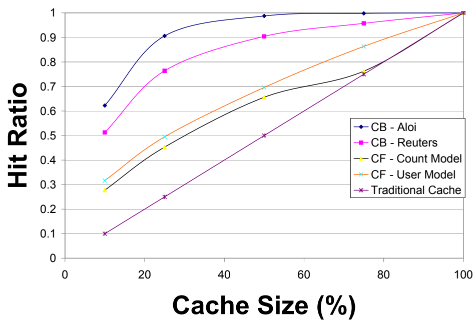

Figure 2: Hit ratio comparison using various proportions of cached data.

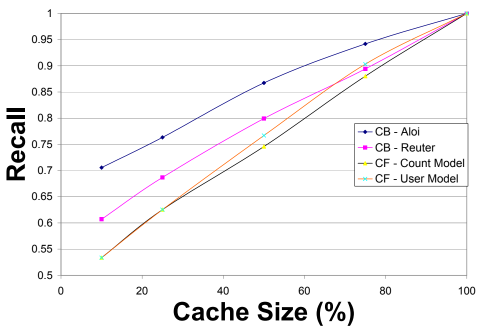

Figure 3: Recall test for content-based and collaborative filtering techniques using different data sets

tion of data. In practice, the recall can be even higher with popular (repeated) queries loaded in the cache.

3.2.3 *Efficiency*

To test the efficiency and cost of computation, we cached 25% of the ALOI dataset and executed similarity queries based at each of its 110,250 data objects. The execution time taken was computed for three methods: traditional caching, active caching, and retrieval from the database with no caching. Figure 4 shows the superior performance of our approach in terms of execution cost when compared with traditional caching and cacheless approaches.

4. CONCLUSION

This paper presents a novel technique for the active caching of top-k similarity queries for recommender systems. The caching mechanism imposes no restriction on the result set computation of the recommender system, as it relies solely on the rank order of the lists produced by the system. This approach is therefore suitable for metric as well as non-metric similarity measures. The experimental results discussed in the previous section show a substantial improvement in the cache hit ratio and computation costs as compared to traditional caching solutions. The test results also show reasonable effectiveness in terms of average recall values.

As with other caching solutions, our approach also incurs the overheads associated with disk accesses in the case of cache miss. However, this overhead is much lower for our method due to the greatly reduced number of cache misses. Active caching provides a substantial performance gain at the cost of a marginal processor overhead due to the processing of reverse lists.

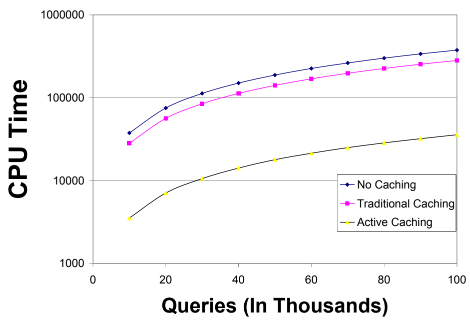

Figure 4: Computational costs of caching for the ALOI dataset, for top-10 similarity queries with cached (forward) lists of size 10. The CPU time is measured in milliseconds.

Acknowledgments

Partial support for this research was provided by the National Science Foundation under grant DUE-0434998.

5. REFERENCES

[1] M. Bhatt. Locality of reference. *Pattern Languages of Programming*, September 1997.

[2] N. Boujemaa, J. Fauqueur, M. Ferecatu, F. Fleuret, V. Gouet, B. L. Saux, and H. Sahbi. Ikona: Interactive generic and specific image retrieval. In *Intern. Workshop on Multimedia Content-Based Indexing and Retrieval (MMCBIR)*, 2001.

[3] P. Cao, J. Zhang, and K. Beach. Active cache: caching dynamic contents on the web. *Distributed Systems Engineering*, 6(1):43–50, 1999.

[4] J. Geusebroek, G. Burghouts, and A. Smeulders. The amsterdam library of object images. *Int. J. Comput. Vision*, 611:103–112, 2005.

[5] J. Herlocker, J. Konstan, A. Borchers, and J. Riedl. Framework for performing collaborative filtering. In *Proceedings of the 1999 Conference on Research and Development in Information Retrieval*, Aug 1999.

[6] M. E. Houle and J. Sakuma. Fast approximate similarity search in extremely high-dimensional data sets. In *International Conference on Data Engineering*, pages 619–630, 2005.

[7] T. M. Kroeger, D. D. E. Long, and J. C. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[8] D. Lewis, Y. Yang, T. Rose, and F. Li. A new benchmark collection for text categoriza-tion research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[9] Z. Li and I. Im. Recommender systems: A framework and research issues. In *Americas Conference on Information Systems (AMCIS)*, 2002.

[10] Q. Luo and J. F. Naughton. Form-based proxy caching for database-backed web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 191–200, September 2001.

[11] E. Markatos. On caching search engine query results. In *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, May 2000.

[12] B. N. Miller. Toward a personal recommender system. In *PhD Thesis*. University of Minnesota, 2003.

[13] S. Nagaraj. *Web Caching and Its Applications*. Kluwer, Norwell, 2004.

[14] J. Williams. Hot technologies with a purpose. *Library Journal*, 127(2):50, Feb 2002.