Analysing Compression Techniques for In-Memory Collaborative Filtering

Saúl Vargas, Craig Macdonald and Iadh Ounis
{firstname.lastname}@glasgow.ac.uk
School of Computing Science, University of Glasgow

ABSTRACT

Following the recent trend of in-memory data processing, it is a usual practice to maintain collaborative filtering data in the main memory when generating recommendations in academic and industrial recommender systems. In this paper, we study the impact of integer compression techniques for in-memory collaborative filtering data in terms of space and time efficiency. Our results provide relevant observations about when and how to compress collaborative filtering data. First, we observe that, depending on the memory constraints, compression techniques may speed up or slow down the performance of state-of-the-art collaborative filtering algorithms. Second, after comparing different compression techniques, we find the Frame of Reference (FOR) technique to be the best option in terms of space and time efficiency under different memory constraints.

Categories and Subject Descriptors: H.3.3 [Information Storage & Retrieval]: Information Filtering

Keywords: Recommender Systems, Collaborative Filtering, Index Compression.

1. INTRODUCTION

With the decrease of memory costs, having servers with hundreds of GBs of main memory is nowadays an affordable option [10]. With such infrastructure, performing in-memory data processing has become a common and feasible option in both single and multi-node environments. This trend can be observed in different areas of computing systems such as databases [12], search indices [2] and recommendation engines [7], more specifically in recommendation engines relying on collaborative filtering (CF) techniques. Indeed, keeping CF data in memory is a usual practice, particularly in the publicly available datasets for research purposes. In real-world datasets, however, CF data is usually several orders of magnitude larger than the public datasets, and thus efficient representations of in-memory CF data may be needed. As CF data is commonly represented as lists of numerical ids of users and items, integer compression techniques [1, 3, 4, 6, 9] can be used to significantly reduce the amount of memory required for representing them.

In this work, we study the use of integer compression techniques to compress CF data. Although there has been prior work studying the benefits of compression techniques for CF data [5], that work focused on a scenario where the data is mainly stored on disk. In that setting, data transfer between disk and memory can be identified as a bottleneck in the computation of recommendations and, consequently, compression techniques consistently help in speeding up the recommendation algorithms. In our fully in-memory setting, depending on the available memory, compression techniques may speed up computing time as well, but may also slow it down. Furthermore, we explore a wider range of compression techniques, finding that the Frame of Reference (FOR) [6] technique offers the best solution in terms of space and time efficiency among the compared approaches.

RecSys 2015 Poster Proceedings, September 16-20, 2015, Vienna, Austria.

2. EXPERIMENTAL SETUP

In order to analyse the effect of compression techniques for CF data, we have conducted a series of experiments with two well known datasets: the dataset from the Netflix prize, containing 100 million ratings from 480,000 users to 17,770 films, and the Yahoo Music dataset, containing 717 million ratings from 1.8 million users to 136,736 songs. In both datasets, ratings are given in a scale between 1 to 5 stars. These are, to our knowledge, two of the largest CF datasets available for academic research purposes.

Both datasets use numerical ids for identifying users and items. In the case of the Netflix dataset, we map the original user ids consecutive ids determined by the numerical order of the original ids. More elaborate id re-assignation techniques, which may lead to further compression efficiency [2, 11], are left for future work. The preferences of each user/item are represented with a list of sorted ids of items/users and a list of numerical ratings. The lists of ids are compressed using a variety of compression techniques: fixed-length coding (fix-len), where each id is coded by using the minimum number of bits required to store the largest id, γ coding [4], Elias-Fano (EF) [3], Rice [9], ζ_3 coding [1] and Frame of Reference (FOR)[1] [6]. With the exception of fixed-length and Elias-Fano, id arrays are stored with *delta-gaps*. Rating values in the 1-5 scale are simply compressed with fixed-length coding, that is, using 3 bits to represent each rating. We use RankSys[2], a Recommender Systems framework written in Java and, on top of it, we use the implementation of FOR in the `JavaFastPFOR`[3] package and the `dsiutils`[4] package for the rest of techniques.

In order to measure the performance of the different compression techniques in terms of time efficiency with respect to uncompressed representations of the CF data, we generate

[1]Results with more sophisticated variations of FOR, such as PFOR, are omitted as they do not differ significantly from those of the simpler, original FOR.
[2]`http://ir-uam.github.io/RankSys/`
[3]`https://github.com/lemire/JavaFastPFOR`
[4]`http://dsiutils.di.unimi.it/`

	Netflix	Y Music
none	1,608	11,486
fix-len	506	4,051
γ	298	2,871
EF	249	2,190
Rice	**241**	**2,130**
ζ_3	266	2,396
FOR	273	2,354

Table 1: Memory usage in MB of the Netflix and Yahoo Music datasets with different compression techniques for user and item ids. Best results in bold.

recommendations for 1,000 users randomly selected in both datasets with the user-based nearest neighbours algorithm [8] provided by RankSys. For the purpose of simulating different memory constraints and their effect on the time efficiency of the recommendations, we selected for both datasets three different settings for heap size (via the `-Xmx` parameter in Java): a heap size in which the uncompressed datasets fit in memory without problem (4.8 GB for Netflix and 32 GB for Yahoo Music), a heap size slightly higher that what is required to keep the dataset in memory (2.4 GB and 16 GB) and, finally, a heap size slightly higher than what is required to allocate the data with fix-len coding but where the uncompressed data cannot be allocated (0.8 GB and 6 GB). Moreover, recommendations were generated in a multi-threaded environment using 8 parallel threads. This simulates a realistic high-demand environment where many of the benefits of caching are lost and there is a high demand of memory for auxiliary data structures for the CF algorithm.

3. EXPERIMENTAL EVALUATION

The results of the experimental setup previously described are summarised in Table 1 and Table 2. The results in terms of memory usage in Table 1 indicate similar trends for both datasets: while a simple fix-len encoding is capable of reducing notably the size of the CF data in memory (by one third in both datasets), the rest of compression techniques are able to further reduce the usage of memory (below 20% in most cases), being EF and Rice the most space efficient among the compared alternatives.

In terms of time efficiency, Table 2 illustrates the change in performance when different memory constraints are considered. Again, the observed trends are equivalent in both datasets. Under the lightest memory constraints, it can be observed that the fastest option is working with uncompressed data. However, with the intermediate memory constraints, all the compression techniques are faster than the uncompressed data. Finally, in the setting with the heaviest memory constraints, the fix-len option heavily suffers the scarcity of available memory, whereas the rest of more space-efficient coding alternatives suffer a milder time penalty. Among the compression approaches, FOR stands out as the best one, being the fastest approach in every setting and only slightly slower than the uncompressed option in the setting with high memory availability.

To understand better the slowness of the uncompressed data and the simple fix-len coding under tight memory constraints, we observed in detail the performance characteristics of the system, and noticed two factors: the higher time spent in the garbage collector and the inability to make full use of the multi-threading capabilities of the system.

On the one hand, these previous observations contrast with prior work [5], in which CF data was primarily stored on classic disk search indices. In that work, the use of compression techniques always represented a speed up in access to CF data. As we observe, in the case of in-memory CF data this situation does not necessarily happen, provided

	Netflix			Y Music		
	4.8 GB	**2.4 GB**	**0.8 GB**	**32 GB**	**16 GB**	**6 GB**
none	**43.96**	109.43	-	**100.74**	267.66	-
fix-len	58.68	60.51	743.05	119.07	122.22	458.50
γ	63.03	64.00	71.13	131.64	128.27	131.49
EF	64.51	67.71	74.20	134.44	130.74	137.54
Rice	69.82	69.73	77.42	144.15	135.53	137.85
ζ_3	63.91	66.65	74.81	129.71	145.08	142.80
FOR	48.79	**50.43**	**55.36**	111.33	**106.80**	**108.92**

Table 2: Execution time in seconds of the user-based nearest neighbours algorithm in the Netflix and Yahoo Music datasets with different compression techniques for user and item ids. Best results in bold.

that the remaining memory is large enough to support the auxiliary data structures and the variables required for the computation of the algorithms. On the other hand, when comparing the performance of the different compression techniques, our results are in line with those of [2] for search index compression, in which the FOR technique is also found to be the best solution in terms of time efficiency.

4. CONCLUSIONS AND FUTURE WORK

In this paper we have conducted a study of the performance of compression techniques for keeping CF data in the main memory. We find that compression techniques in this case, as opposed to when the data is primarily stored on disk, may actually slow down the processing time when the remaining memory available for the processing of CF algorithms is large enough. Under more severe memory constraints, we find that compression techniques are indeed able to speed up the generation of recommendations. Finally, we find that the FOR technique is the best compression approach as its space efficiency is at the same level of the rest of compared alternatives while its time efficiency is clearly better than these and close to that of using no compression.

As part of future work, we envisage the usage of hybrid representations of CF data, in which the system may selectively compress part of the data in order to maximise the performance of the system under different memory constraints.

5. REFERENCES

[1] P. Boldi and S. Vigna. Codes for the world wide web. *Internet Mathematics*, 2(4), 2005.

[2] M. Catena, C. Macdonald, and I. Ounis. On inverted index compression for search engine efficiency. In *ECIR*, 2014.

[3] P. Elias. Efficient storage and retrieval by content and address of static files. *J. ACM*, 21(2), 1974.

[4] P. Elias. Universal codeword sets and representations of the integers. *IEEE Trans. Inf. Theory*, 21(2), 1975.

[5] V. Formoso, D. Fernández, F. Cacheda, and V. Carneiro. Using rating matrix compression techniques to speed up collaborative recommendations. *Inf. Ret.*, 16(6), 2013.

[6] J. Goldstein, R. Ramakrishnan, and U. Shaft. Compressing relations and indexes. In *ICDE*, 1998.

[7] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh. WTF: The who to follow service at Twitter. In *WWW*, 2013.

[8] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *CSCW*, 1994.

[9] R. Rice and J. Plaunt. Adaptative variable-length coding for efficient compression of spacecraft television data. *Trans. Communication Technology*, 19(6), 1971.

[10] A. Rowstron, D. Narayanan, A. Donnelly, G. O'Shea, and A. Douglas. Nobody ever got fired for using Hadoop on a cluster. In *HotCDP*, 2012.

[11] F. Silvestri. Sorting out the document identifier assignment problem. In *ECIR*, 2007.

[12] H. Zhang, G. Chen, B. Ooi, K. Tan, and M. Zhang. In-memory big data management and processing: A survey. *IEEE TKDE*, 27(7), 2015.