There Is No Dichotomy Between Effectiveness and Efficiency in Keyword Search Over Databases

Vahid Ghadakchi [1], Arash Termehchy [2]

School of Electrical Engineering and Computer Sciencec
Oregon State University
Corvallis, Oregon, USA
[1] ghadakcv@oregonstate.edu
[2] termehca@oregonstate.edu

I. Introduction

Ordinary users, such as scientists, are not familiar with the concepts of schema and query language and rely on query interfaces that can handle keyword queries or natural language [1]. Moreover, users who know query languages such as SQL, do not often know the content and/or structure of the data sufficiently well to precisely specify their queries. As imprecise queries do not exactly reflect the users' intents, the foremost challenge of a database system is to discover the relevant answers to these queries. Roughly speaking, the database system finds a list of plausible interpretations for these queries using different IR, NLP and machine learning techniques, runs these interpretations, and returns their results. The returned results are significantly less effective than what users want: they contain too many non-relevant answers and few relevant answers, i.e., low precision and recall [2]. Furthermore, to run a keyword query, the database system has to process a considerable number of interpretations for each input query. Each interpretation is a rather complex database query with potentially large number of joins [3] which makes the keyword query processing to become extremely time-consuming. The rapid increase in the amount of data and size of databases only exacerbates the mentioned problems. These challenges poses the following question: can we redesign the main components of a database system to improve both effectiveness and efficiency of answering keyword queries over a large database? We show that it is possible to achieve this goal by presenting our effort on finding and caching an optimal subsets of a database to improve both effectiveness and efficiency of answering keyword queries.

II. Finding Optimal Subset of The Database

To address the aforementioned problems, we investigate to see if there is a subset of the database over which the query interface can answer queries more effectively and efficiently than over the entire database. It is known that accessing data items in a database, generally follows a power-law distribution: a large number of queries target relatively few, frequently accessed data items. If the query interface conducts the search over frequently accessed data, it is likely that it finds some of the relevant answers within this subset. More importantly, it is generally easier to find the relevant answers within a smaller subset of candidate answers [4]. Thus, by submitting the queries to a subset of frequently accesses tuples, one might achieve a higher precision in answering queries. We confirm this hypothesis by conducting extensive experimental studies on real world data and queries. While finding a few relevant answers over a smaller subset of the database is easier, this subset may not contain all the relevant answers to most queries. More accurately, using the subset may result in a reduced recall (i.e. fraction of returned relevant answers by the system over all relevant answers). In section III, we show that our method increases the precision while maintaining a reasonably high recall.

In order to find an *optimal* subset of a database, one has to find the right size of the frequently accessed tuple set: a small set may not contain the relevant answers to many queries and a large one may make the search as difficult and time-consuming as the entire database. We learn the right size of this set using a probabilistic model learned over a sample of input queries. More precisely, given a database, access counts of its tuples and a sample of its query workload, we build a subset of the database that contains N most frequently accessed tuples of the database. To determine the value of N, we use a small sample of queries and do a grid search over different subset sizes and pick the one that results in the highest effectiveness. We explain the details of this approach in Section III.

While the proposed method is reminiscent of using a traditional main-memory cache to improve the running time of queries over a database, it has two major differences with the caching systems. 1) In traditional caching scenarios, the size of the cache is an input of the problem. It is determined based on the available resources and is fixed. A larger cache has a better performance. However in our case, the size of the subset should be determined by the system and a larger subset does not necessarily have better effectiveness. 2) Traditional caching is intended to just improve the efficiency of answering queries, whereas the main goal of using an optimal subset is to increase effectiveness. Furthermore, since the subset is significantly smaller than the database, it will have a better efficiency in answering queries. One may further improve this efficiency by storing the subset in the main memory.

III. Experimental Results

We perform several experiments using a real-world query workload from MSN with 7000 queries and a benchmark INEX query workload with 160 queries to test our approach. We submit the queries to a snapshot of Wikipedia from 2013 with more than 11 million articles and their access counts. Note that these access counts are obtained independent of the queries in our query workloads. The results indicate that by caching a small part of the database, we achieve significant improvements in effectiveness of answering queries. To evaluate the effectiveness of query answering, we use *reciprocal rank*, *precision-at*-20 (p@20), and *recall* [4]. We compute the average of these metrics over the queries of each query workload. Consider database I with tuples $t \in I$ and access counts $w(t)$. We build $I_i \subset I$ by picking the top $i\%$ tuples of I based on the value of their access counts. It follows from the definition that if $i < j$ then 1) $I_i \subsetneq I_j$, 2) the access counts of the tuples in I_i are greater than or equal to the access counts of the tuples in I_j.

Figure 1 shows the effectiveness of answering MSN queries over $I_2, I_4 \dots I_{100}$. The x axis shows the database subset I_i over which we answer the input queries and the y axis shows the value of mean reciprocal rank (a.k.a. MRR) and average recall of the queries. MRR has its maximum value, 0.62, on I_2 that has 2% of the tuples of the original database. The full database (I_{100}) provides the lowest value of MRR, 0.25. This indicates that I_2 has sufficiently many relevant answers for most queries in the query workload. Adding more tuples to I_2 may increase the number of relevant answers to the input queries, but it puts significantly more tuples that are not relevant to any query or are relevant to very few queries in I_2. Furthermore, this figure shows that the value of recall over subset is very close to the value of recall over full database. Since most queries in this workload have a single answer, we do not report p@20 for this query workload.

Figure 2 shows the effectiveness of answering INEX queries over $I_2, I_4 \dots I_{100}$. As it is shown in this figure, the maximum p@20 and recall happen at I_{12} and I_{28} respectively, which shows that using proper subsets of a database to answer queries will significantly increase the quality of ranking and recall of the results. Unlike the results for MSN query workload, in this experiment the smallest subsets do not have the best average p@20 and $recall$. The reason is that these subsets do not contain enough relevant tuples. Even though it is easy to find the relevant tuples in these subsets, the limited number of relevant tuples decreases the average p@20 and $recall$. Thus based on the type of query workload, our proposed method is able to pick an effective subset of the database.

To evaluate the robustness of the system, we repeat the experiments using 10% of each query workload to find the optimal dataset and test it using the rest of the queries. The results of this evaluation is similar to the results presented in the previous parts.

Lastly, we present the efficiency results of our system. The average time spent on each MSN query over I_2 and I_{100} is 0.07 and 0.36 seconds respectively. Furthermore, the average time spent for each INEX query over I_{12} is 0.27 seconds whereas for the entire database, the average spent time per query is 0.79 seconds per query. These results show our system can reach high effectiveness and efficiency simultaneously. Note that on average, MSN queries have less keywords than INEX queries and that's why MSN queries take less time to be executed.

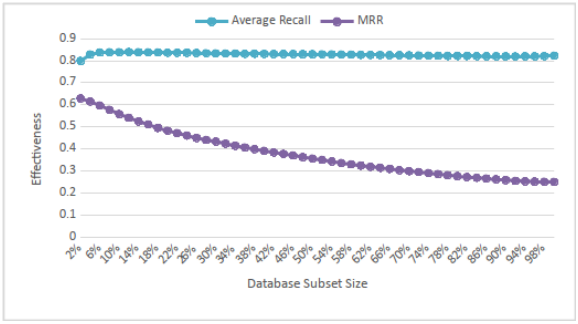

Fig. 1. Effectiveness of MSN queries over different subsets

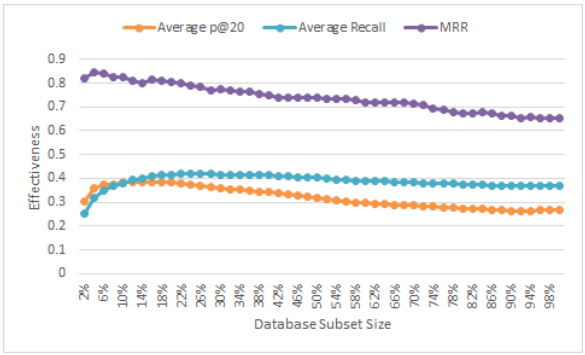

Fig. 2. Effectiveness of INEX queries over different subsets

IV. Conclusion

In this paper, we show how a small subset of a database can answer queries more effectively and efficiently. Furthermore, we show how one can pick the right subset based on a sample of the query workload. The provided technique can be applied to any database with an imprecise query interface such as keyword query interface. The proposed system comes with one caveat: It can not effectively answer rare queries on rarely accesses tuples. As a future work, we are planning to extend the system to help rare queries, while maintaining the high average effectiveness for all queries.

References

[1] Y. Chen, W. Wang, Z. Liu, and X. Lin, "Keyword search on structured and semi-structured data," in *SIGMOD*, 2009.

[2] J. Coffman and A. C. Weaver, "A framework for evaluating database keyword search strategies," in *Proceedings of the 19th ACM international conference on Information and knowledge management*. ACM, 2010, pp. 729–738.

[3] J. Coffman and A. Weaver, "An empirical performance evaluation of relational keyword search techniques," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 30–42, 2014.

[4] C. Manning, P. Raghavan, and H. Schutze, *An Introduction to Information Retrieval*. Cambridge University Press, 2008.