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

Arash Termehchy 1, Vahid Ghadakchi 2

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

I. INTRODUCTION

Ordinary users such as scientists, are not familiar with the concepts of schema and query languages and can use only inherently vague and imprecise query interfaces, such as keyword query interfaces [1]. Moreover, with the rapid increase in the amount of data, users who know database query languages, e.g., SQL, do not often know the content and/or structure of the data sufficiently well to precisely specify their queries and have to explore the database via a series of imprecisely written queries. As imprecise queries do not exactly reflect the users’ intents, the foremost challenge of a database system is to discover the right and relevant answers to these queries, i.e., answer them effectively. However, the components of current database systems, such as caching and indexing, are designed to run queries efficiently and do not take effectiveness into account. Hence, database systems discover the relevant answers for imprecise queries at the query interface level: roughly speaking, they find a set of plausible interpretations of the input query using schemes developed in information retrieval, natural language processing, or machine learning communities, run these interpretations, and return their results.

The experience gained from developing keyword query interfaces over relational databases indicate that the returned results are significantly less effective than what users expect [2]. It also shows that it is very time-consuming to run keyword queries over large databases because no matter how intelligent the query processing algorithm is, it has to process a considerable number of interpretations for each input query where each interpretation is a rather complex database query, e.g., SQL queries with a large number of joins [3]. Clearly, such an architecture will not scale to Big Databases with billions of tuples over which running a single SQL query may take a long time. Also, due to massive number of plausible interpretations for a query over such a large database, it is very hard to find the relevant answers. These challenges poses the following question: can we redesign the main components of a database system to improve both effectiveness and efficiency over a large database? We show that it is possible to achieve this goal by presenting our effort on leveraging caching mechanism in database systems to improve both effectiveness and efficiency of answering keyword queries.

II. PROPOSED APPROACH

To address the aforementioned problem, 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 set. More importantly, it is generally easier to find the relevant answers within a smaller set of candidate answers. However, one has to find the right size of the frequently accessed tuples: 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 the cache using a probabilistic mode 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 would like build a subset of the database that contains $N$ most frequently accessed tuples of the database. To determine the value of $N$, we submit the sample queries to subsets of the different size and pick the subset with highest effectiveness.

We have performed several experiments using real-world query log from MSN and INEX ad-hoc search query workload to test our approach. The results indicate that by caching a small part of the database, we achieve significant improvements in effectiveness of answering queries. Furthermore, since the queries are being processed over a subset of the database, the efficiency of the system is much higher than using the full database. 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 cache is an input of the problem and is determined based on the available resources. 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 effective 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

Consider database $I$ with tuples $t \in I$. We denote the users’ access counts (access counts for short) of $t$ as $w(t)$. We build $I_i \subset I$ by picking the top $i\%$ tuples of $I$ based on the value of their access count. It follows from the definition that if $i < j$ then 1) $I_i \subseteq I_j$, 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 mean reciprocal rank MRR of answering queries over $I_2, I_4, \ldots, I_{100}$. The $x$ axis shows the database subset $I_i$ over which we answer the input 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. Furthermore, the average time of querying the subset $I_2$ is 0.27 seconds whereas for the entire database this number is 0.79.

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 $I_2$. The latter group of tuples reduces the MRR of answering queries as the retrieval system may incorrectly select some of them as answers for the input queries. Thus, although other subsets other than $I_2$, such as $I_{100}$, have more tuples than $I_2$, it is harder to find the relevant answer of most queries over them than on $I_1$. The average $p@10$ and $p@20$ for the Bing query workload leads to similar results as the ones of MRR. Furthermore, since most queries in this workload have a single answer, the recall of each query is 0 or 1 and the trend of average recall will be similar to MRR.

In this section, we repeat the same experiment using INEX queries. Since each query in the INEX query workload has more than one relevant answer, we report the average $p@20$ and recall for these queries in Figure 2. As it is shown in these figures, 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 Bing 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. We go beyond a single best subset and find a range of subsets that are significantly better than the full database. The results of t-test show that the subsets $S_2 = \{I_4 \ldots I_{54}\}$ and $S_3 = \{I_{12} \ldots I_{54}\}$ have significantly better average $p@20$ and recall respectively compared to the full database.

IV. CONCLUSION

REFERENCES