A lightweight distributed order and duplication insensitive algorithm for approximate top-k queries using order statistics

Vinay Deolalikar
Hewlett Packard Labs
1501 Page Mill Road
Palo Alto, CA 94304
vinayd@hpl.hp.com

Kave Eshghi
Hewlett Packard Labs
1501 Page Mill Road
Palo Alto, CA 94304
kave@hpl.hp.com

Hernan Laffitte
Hewlett Packard Labs
1501 Page Mill Road
Palo Alto, CA 94304
hernan@hpl.hp.com

1. APPROXIMATE TOP-K

Let $\{e_1, e_2, \ldots, e_l\}$ be a set of distinct records in a database, with unique IDs $\{id_1, id_2, \ldots, id_l\}$. Let $A_1, A_2, \ldots, A_p$ be a set of distinct attributes for each record. For every record e_i, the attribute A_j is zero or some positive value. We denote the value of the attribute A_j of record e_i by $A_j(e_i)$. The sum of the attributes of e_i is denoted by $N_i = \sum_j A_j(e_i)$. We would like to obtain the list of top k records, ordered by N_i. We present a highly configurable, lightweight, distributed algorithm to solve the above problem approximately, based on order statistics.

2. THE ALGORITHM

2.1 Phase One: Generating a list of random variables

A ticket is a triple $< ID, r, b >$ where id is a record, r is the value of a random variable, and b is a binary flag which can be set to either 1 or 0, respectively.

Each peer first generates an exponential random variable for the record e_i with rate given by $A_j(e_i)$. At the end of this phase, each peer will have a list of random variables that is as long as the number of records. The list has two columns: the first column has the record ID and the second column has the random variable value.

2.2 Phase Two: Pruning the list

Each peer thresholds the list of random variables they have generated. Rows in the list whose second column (random variable value) is below a threshold T are discarded.

2.3 Phase Three: Exchanging lists

In the third phase of the algorithm, each peer sends their pruned list of (record ID, random variable value) to their neighbors. This is the information passing phase of the algorithm.

2.4 Phase Four: Merging lists by maximum

Each peer now has lists from other peers. Each peer now merges these lists by keeping only the maximum of the values of the random variables for each record.

2.5 Phase Five: Cropping merged lists

Each peer now sorts his merged list in descending order of random variable value, and crops it to have only L topmost records.

Now the algorithm proceeds by looping through phases Two through Five for a fixed number of iterations. Experimental results indicate that 5 iterations suffice for a stabilization of lists.

2.6 Phase Six: Running algorithm multiple times and merging results

Phases One through Six are run a 'run count' of times. At the end of each run, a list emerges. Now, a final list is obtained as follows. If a record occurs in at least 'merge count' out of the total 'run count' number of lists, then it is included in the final output of the algorithm as a top k record.

As with any approximate algorithm, we may merge results of multiple runs of the basic algorithm outlined above, in order to increase accuracy.

3. EXPERIMENTAL RESULTS

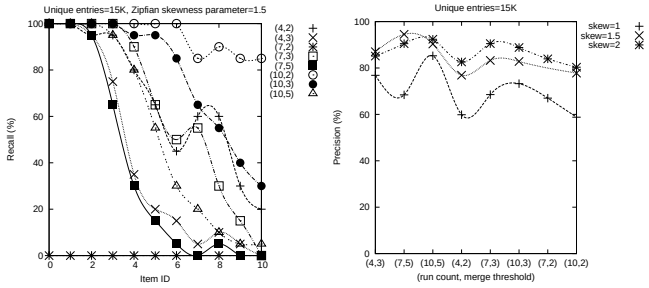

Figure 1: Tradeoff between (run count, merge threshold) and precision vs. recall on Zipfian distribution over 15K distincts. Item id indicates frequency of item: Item 0 is most frequent, and so on.

We have validated our algorithm extensively on a wide array of multi-parameter Zipfian datasets, varying the skewness of the distribution of records (only one parameter choice shown in Fig. 1). We report strong performance of the algorithm over a wide range of parameter values, and study the trade-offs involved in setting the tunable parameters of the algorithm in order to obtain the precision and recall that is desired.

The 18th International Conference on Management of Data (COMAD), 14th-16th Dec, 2012 at Pune, India.