Keyword Search over Dynamic Categorized Information

Manish Bhide [#1], Venkatesan T. Chakaravarthy [#2], Krithi Ramamritham [*3], Prasan Roy [%4]

[#]*IBM India Research Lab., New Delhi, India.* { [1]abmanish, [2]vechakra}@in.ibm.com

[*]*IIT Bombay, Mumbai, India.* [3]krithi@iitb.ac.in

[%]*Aster Data Systems, CA, USA.* [4]prasan.roy@aster-data.com

***Abstract*— Consider an information repository whose content is categorized. A data item (in the repository) can belong to multiple categories and new data is continuously added to the system. In this paper, we describe a system, CS*, which takes a keyword query and returns the relevant top-K categories. In contrast, traditional keyword search returns the top-K documents (i.e., data items) relevant to a user query. The need to dynamically categorize new data and also update the meta-data required for fast responses to user queries poses interesting challenges. The brute force approach of updating the meta-data by comparing each new data item with all the categories is impractical due to (i) the large cost involved in finding the categories associated with a data item and (ii) the high rate of arrival of new data items. We show that a sampling based approach which provides statistical guarantees on the reported results is also impracticable. We hence develop the CS* approach whose effectiveness results from its ability to focus on a strategically chosen subset of categories on the one hand, and a subset of new data on the other. Given a query, CS* finds the top-K categories with high accuracy even in time-constrained situations. An experimental evaluation of the CS* system using real world data shows that it can easily achieve accuracy in excess of 90%, whereas other approaches demand at least 57% more resources (i.e., processing power), for providing similar results. Our experimental results also show that, contrary to expectations, if the rate of arrival of data items doubles, whereas CS* continues to provide high accuracy without a significant increase in resources, other approaches require more than double the number of resources.**

I. Introduction

In this paper, we describe a system, CS* (Category Search), which takes a keyword query and returns the relevant top-K categories, given an information repository containing categorized content. Its key features include:

- Unlike current search engines which respond with (just the identity of) the top-K documents that are relevant to the keywords specified, CS* returns the top-K relevant categories. The framework underlying CS* is capable of handling any information categorization mechanism.
- CS* is appropriate for domains in which new information arrives at a high rate or where new categories are specified by users.
- CS* is able to find the correct top-K categories with high accuracy even in time-constrained situations.

Motivation: Consider a scenario where PC, a presidential candidate, has recently announced his education manifesto. PC's campaign manager is anxious to assess the reaction to the manifesto among different categories of potential voters. So the manager examines postings on blogs, forums, wikis, etc., by firing a keyword query "PC education manifesto" over a popular blog search engine. The results returned consist of a large number of blog posts, with the blog entries from highly ranked blogs being returned at the top of the search result. Unfortunately, the campaign manager finds it tedious to read through all the results and from that evolve a consolidated reaction focusing on specific categories of voters. What would have been preferable are search results which, for example, inform the campaign manager that among the reactions from various groups of interest to the campaign, the most relevant can be found in postings about (i) K-12 education and (ii) about high school students' interest in science. Reading a sample set of recent postings from each of these top categories would help the campaign manager understand the real issues raised by these important groups. For instance, the manager might find out that the education policy is perceived as not adequately addressing the concerns of K-12 school teachers. This would help the campaign manager to take immediate follow up action.

Notice that using traditional search, grouping the results into different categories (using say, document clustering techniques) and ordering the categories based on the number of posts in each category, will not provide the desired results. Given the keyword query "PC education manifesto", such a system might return the category "Postings of users from America" at the top, as majority of the posts would be from people in America. The crucial factor that it fails to capture is that postings about PC's education policy are likely to constitute a minuscule fraction of the postings from America. This difference can only be captured by a system that categorizes updates along specific categories and provides keyword search over such categories. Such a system would be able to identify that majority of the blog posts about K-12 schools refer to PC's education policy and hence would place them at the top of the rank order.

Consider a second application, one from the financial domain. In a stock exchange, transaction information is categorized based on the stock(s) associated with the transaction, buyer/seller profiles, etc. Example categories could be "Transactions made by retail customers","Transactions made by high value customers", etc. Now consider an analyst who

wants to investigate recent sudden jumps in the price of IBM and Microsoft stocks. Using CS*, the analyst can issue a keyword query "IBM Microsoft" to find the top-K categories of buyers/sellers of these stocks. The results could show the following categories at the top (i) Transactions made by Bank of America customers and (ii) Transactions made by high value customers. The analyst uses this to do further investigation and realizes that there was a tip issued by Bank of America to its customers about IBM and Microsoft stocks. This is a classic example of Real Time Business Intelligence. Notice that the regular keyword search using existing techniques would not be meaningful as this would return individual transactions (rather than the categories) which may not be of direct use to the analyst.

Problem Definition: In this paper we study the problem of top-K keyword search over categories, where each category is associated with, or is mapped to, one or more pieces of information, referred to simply as *data items*. The data items in the above examples include the blog posts, forum postings and stock transactions.

Consider a continuously growing information repository of *data items* d_1, d_2, $d_3, \cdots$. Each data item d is associated with a set of attributes $A(d)$ and a multi-set of terms $T(d)$ drawn from a universe of terms $\mathcal{T}$. Each data item belongs to one or more categories from a set of *categories* $\mathcal{C}$. Each category c is associated with a boolean predicate $p_c(\cdot)$ that takes as input a data item d and tells whether d belongs to category c. $p_c(\cdot)$ is evaluated over $A(d)$ and $T(d)$. E.g., the predicate for the category "Forum postings about high school students' interest in science" would be realized by a text classifier which would take the posting as input and decide whether the posting belongs to the category. On the other hand, the predicate for the category "Blog post of people from Texas" would make use of the attributes of the data item (i.e., the blog author's profile) to decide the categorization. Thus the predicate is domain dependent and will be provided as input to CS*.

In this paper, for ease of explanation, we do not measure time in absolute terms, but in terms of *time-step*. Updates to the information repository with one or more data items causes the time-step to be incremented proportionately (i.e., equal to the number of data items added). Thus, there is a one-to-one mapping between a time-step and the data item added to the information repository in that time-step. Let D_s denote the set of all data items till time-step s, then: $D_s = \{d_i : 1 \leq i \leq s\}$. The ***data-set*** ($M_s(c)$) of a category c, till time-step s, is defined to be the set of all data items added till time-step s that map to c: $M_s(c) = \{d \in D_s \ : \ p_c(d) = 1\}$.

Given D_s and the set of categories $\mathcal{C}$, our goal is to find the set of categories most relevant to a keyword query given by a user at some time-step s. A keyword query $\mathcal{Q}$ consists of a set of keywords $\{t_1, t_2, \ldots t_\ell\}$. We need to evaluate the keyword query with respect to data items added till time-step s, i.e., the set D_s.

The top-K categories are found using a scoring function. Let the score of a category c with respect to a keyword query $\mathcal{Q}$ be represented by $\mathrm{Score}(c, \mathcal{Q})$. This would depend on the score of c for each keyword present in $\mathcal{Q}$. Let $\mathcal{F}$ be the function that takes as input a category c and a keyword t and computes the score of the category with respect to t. Further, let $\mathcal{G}$ be a function that combines the $\mathcal{F}(c, t)$ values for each $t \in \mathcal{Q}$ to arrive at $\mathrm{Score}(c, \mathcal{Q})$. Then,

$$\mathrm{Score}(c, \mathcal{Q}) = \mathcal{G}(\mathcal{F}(c, t_1), \mathcal{F}(c, t_2), \ldots, \mathcal{F}(c, t_\ell)) \quad (1)$$

Given a user query $\mathcal{Q}$ issued at a time-step s, the goal is to find the categories having the top-K scores among all the categories defined in the system. Here, K is an input parameter. The query answering process makes use of ***meta-data*** which helps it to find the value of $\mathrm{Score}(c, \mathcal{Q})$ for each c. But $\mathrm{Score}(c, \mathcal{Q})$ is bound to change as D_s and hence $M_s(c)$ changes with the passage of time s. This implies that, to compute the score accurately, we must keep the meta-data up-to-date as data gets added to the repository. If data arrival rate is high, this is a challenging task.

A simple strategy to solve this problem is to eagerly update the meta-data as soon as new data items arrive. The update-all strategy, described next, uses such an approach.

Update-all Strategy: This strategy ***refreshes*** all the categories whenever a new data item is added. This involves evaluating the boolean predicate (p_c) of each category on each new data item and updating the meta-data of those categories whose predicate evaluates to true. This strategy could be very time consuming as the process of evaluating the boolean predicate of each category on the new data item is likely to be a costly operation. For the presidential candidate scenario, this would involve running a text classifier on the blog entry (data item). If the text classifier can classify the blog entry on an average in say 25 milliseconds, then with 1000 categories 25 seconds will be required to refresh all categories using one data item. Similarly, in the stock exchange scenario, evaluating the boolean predicate would require firing a SQL query on a database of stock transactions. These queries could involve costly joins with the company or user profile. Hence while a data item is being processed more data items could be added to the system. This is particularly true for both the examples given above as the number of transactions in a stock exchange is very high and in the blog world, according to a 2006 estimate, more than 13 blog entries are generated per second [1]. Thus by the time all the categories are refreshed using a single data item, more data items would have been added. Therefore, as time progresses, such a meta-data update strategy would start lagging behind, resulting in an increase in the number of unprocessed data items. As a result, the meta-data required for answering the keyword queries becomes stale, leading to inaccurate top-K results. Clearly what is required, is a way by which meta-data of a select set of categories is refreshed using a select set of data items. Incorporating this idea, we have developed the *selective update strategy* which is used by the CS* system.

The CS* Approach: As new data items are added, the selective update strategy of CS* first identifies a subset of the categories that are deemed 'important' and then identifies the subset of data items which can provide the maximum impact

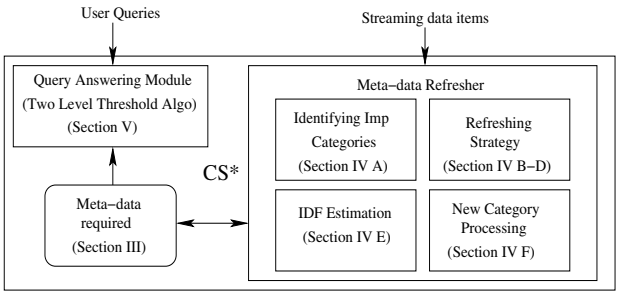

Fig. 1. Overview of CS*

in terms of update to the meta-data and hence to the accuracy of the reported top-K categories. These data items are used to refresh the selected important categories. This way, CS* utilizes the processing time effectively, by only focusing on the important categories and important data items.

The CS* system which uses this novel approach consists of two components: (i) meta-data refresher module and (ii) query answering module. The Meta-data Refresher Module uses the selective update strategy for updating the meta-data as new data items are added. The meta-data updated by this module consists of an inverted index which maps each keyword t, to the set of all categories that contain t in their data-set. The index also contains information needed to compute the functions $\mathcal{F}$ and $\mathcal{G}$ (see Equation 1). Section IV presents the details of this module. The Query Answering Module finds the top-K categories for a given keyword query Q issued at time-step s^* using the meta-data. Since it is impossible to keep the meta-data always up-to-date, it is infeasible to find the exact top-K categories. Thus, providing results with a high level of accuracy is a challenge. We address this issue by carefully maintaining estimates of the actual statistics. Another challenge is to efficiently find the top-K answers in terms of running time. We have developed an innovative two-level threshold algorithm for this purpose, described in Section V. The **research contributions** of our work include:

- We formulate the problem of keyword search over dynamically categorized information. We believe that this problem has a lot of practical significance as indicated by the examples described in this paper.
- We present resource-efficient techniques for maintaining the indexes and statistics for answering keyword queries with high levels of accuracy. These involve a number of interesting subproblems for which we develop systematic solutions.
- We present a novel two-level threshold algorithm for answering keyword queries in an efficient and highly accurate manner.
- We present an experimental study of the CS* system using real world data. Our analysis shows that CS* is able to provide an accuracy of over 90%. In comparison, the update-all technique requires atleast 57% more resources for providing similar results.

Paper organization: Section II presents a sampling based approach which provides theoretical guarantees on the results. We show the impracticality of such an approach and then present an overview of the statistics maintained by the CS* system in Section III. The meta-data refresher is outlined in Section IV and Section V presents the details of the query answering module. An experimental evaluation of our system is given in Section VI. Related work is summarized in Section VII and Section VIII concludes the paper.

II. An Approach that provides Statistical Guarantees

A high rate of arrival of data items leads to staleness in the computed meta-data which in turn can affect the accuracy of the top-K results. Hence what we need is a technique which can guarantee a bounded error in the computed meta-data and hence in the top-K results. In this section we present such a sampling based technique which finds the top-K categories using a scoring function with bounded error. In order to explain this technique, we use the standard scoring technique used in information retrieval, based on term frequencies (tf) and inverse document frequency (idf) [2]. We first define the $tf \cdot idf$ based scoring function for our setup and then outline the sampling based approach.

A. tf and idf based scoring functions

The *term-frequency* of a term t in a category c at time-step s, is defined to be the number of times t occurs in the data-set of c normalized by the total number of terms present in it. Recall from Section I, that there will be s data items in the system at time-step s. Further, let $f(d,t)$ denote the number of times t appears in d. Then,

$$tf_s(c,t) = \frac{\sum_{d\ :\ d \in M_s(c)} f(d,t)}{\sum_{t' \in \mathcal{T}} \left(\sum_{d\ :\ d \in M_s(c)} f(d,t') \right)}$$

Notice that the numerator counts the number of times t occurs in the data-set of c and denominator counts the total number of terms in the data-set. This size normalization helps to avoid bias towards categories with large number of terms.

The inverse document frequency seeks to scale down those terms that occur more frequently across different categories. At a time-step s, the idf of a term t is calculated as follows:

$$idf_s(t) = 1 + log\left(\frac{|\mathcal{C}|}{|\mathcal{C}'|}\right) \tag{2}$$

where $\mathcal{C}$ is the set of all categories in the system and $\mathcal{C}'$ is the set of categories whose data-set contain t, i.e., $(\exists d \in M_s(c))[f(d,t) \geq 1]$.

For a keyword query $\mathcal{Q} = \{t_1, t_2, \ldots, t_\ell\}$ issued at time-step s, the score of a category c is obtained by summing the individual $tf \cdot idf$ scores of each keyword present in the query.

$$\text{Score}_s(c, \mathcal{Q}) = \sum_{t_i \in \mathcal{Q}} (tf_s(c,t_i) \times idf_s(t_i)) \tag{3}$$

One way of providing a guarantee on the top-K results is to compute the $\text{Score}_s(c, \mathcal{Q})$ value with a bounded error and ensuring that there is no overlap in the computed $\text{Score}_s(c, \mathcal{Q})$ values (due to the error bounds) for the top-K categories. We

now explain a technique to compute the $\text{Score}_s(c, \mathcal{Q})$ value (i.e., the tf and idf value) with a bounded error.

B. Estimating idf Values with Bounded Error

In order to find the idf value of all terms with a bounded error, we pick up a uniform random sample of the categories and refresh them (continuously) using all the data items. The idf value is computed using these sampled categories. The accuracy of the computed idf value will depend on the number of sampled categories. We use Chernoff bounds [3] to find a correlation between the size of sample and the error bounds on the computed idf values.

We can have a good estimate of the $idf(t)$ value for a term t, if we have a good estimate on the value of $\frac{|\mathcal{C}'|}{|\mathcal{C}|}$ (see Equation 2). Let $\tau = \frac{|\mathcal{C}'|}{|\mathcal{C}|}$ and let us sample n categories out of the total of $\mathcal{C}$ categories. Let the random variable $X_i = 1$ if the i^{th} category contains the term t. Let the random variable $\mathcal{X}$ represent the number of categories in the sample that contain the term t. The expected value of $\mathcal{X}$ is given as: $E[\mathcal{X}] = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = n\tau$.

For any positive constant, $0 \leq \epsilon \leq 1$, the Chernoff Bounds provide information on how close the actual occurrence of a term in the sample is to its expected count. Applying the Chernoff Bound formula to $\mathcal{X}$:

$$\begin{aligned} P(\mathcal{X} \leq (1-\epsilon)n\tau) &\leq e^{\epsilon^2 n\tau/2} \\ P(\mathcal{X} \geq (1+\epsilon)n\tau) &\leq e^{\epsilon^2 n\tau/3} \end{aligned} \tag{4}$$

ϵ in the above formula measures the accuracy of the sample. The bounds also tell us the probability that the sample of size n will have a given accuracy. This is called the confidence of the sample and is defined as 1 minus the expression on the right hand side of the equations. If we are willing to tolerate a confidence ($Conf = 1 - \rho$) of say 90% (i.e., $\rho = 0.1$), then we can equate the RHS of Equation 4 with ρ. Doing this (for the first of the two equations) we obtain: $\rho = e^{\epsilon^2 n\tau/2} \implies n = -\frac{2log_e(\rho)}{\epsilon^2 \times \tau}$.

If we want an accuracy of 99%, then $\epsilon = 0.01$. Substituting the values of ϵ and ρ, we get: $n = \frac{46051.7}{\tau}$. For $|\mathcal{C}| = 1000$, the value of τ can be of the order of 0.001. In such a case, the number of samples that would be needed will be 46051700. As the number of categories will be smaller than this, such an approach translates to maintaining all the categories up-to-date with respect to all the data items. This is same as the update-all technique which has a huge overhead. Hence the sampling based approach for finding the $\text{Score}_s(c, \mathcal{Q})$ value with a bounded error is not practical.

C. Estimating tf Values with Bounded Error

In order to find the tf value with a bounded error, we keep picking a uniform random sample of the data items (as they are added) and use it to update the tf values. Using arguments similar to those used for idf estimation, we can show that the number of data items that need to be sampled will be very large. Hence such approaches fall behind the data arrival rate and do not provide good results as is corroborated by our experimental evaluation.

In summary, sampling based approaches that provide guaranteed accurate results are not practicable in environments with high data arrival rates. Hence there is a need for a technique that may not have guaranteed accuracy, but provides high accuracy in practise. In the next three sections we present details of the CS* system which has these properties.

III. Statistics maintained by CS*

CS* maintains statistics (i.e., meta-data) to find the value of $\text{Score}(c, \mathcal{Q})$ given by Equation 1. We use the standard information retrieval technique of scoring based on term frequencies (tf) and inverse document frequency (idf) for explaining this feature of CS*. The scoring function using $tf \cdot idf$ is given in Equation 3. The statistics that need to be maintained for this scoring function includes the term frequency and the inverse document frequency. In our system, we ensure that the statistics of the categories are refreshed contiguously. This means that whenever the statistics of a category c are refreshed using a data item d_s (the item added at time-step s), then it is ensured that they have also been refreshed using all data items added till the time-step s i.e., the set of data items $d_1, d_2, \ldots d_{s-1}$. This contiguous property improves the efficiency of the algorithms used by the meta-data refresher (details in Section IV). For a category c, let s' be the largest time-step such that statistics of c have been refreshed using $d_{s'}$; we refer to s' as the *last refresh time-step* of c and denote it by $\text{rt}(c)$. Thus, when the statistics of c are refreshed again, $d_{s'+1}$ will be the first data item to be considered. As a result, the statistics contains the term frequency information for c till $\text{rt}(c)$, namely $tf_{rt(c)}(c, t)$ is available for any term t.

Suppose a keyword query $\mathcal{Q} = \{t_1, t_2, \ldots, t_\ell\}$ is presented at the current time-step s^*. In order to accurately find the top-K categories, we need the term frequencies $tf_{s^*}(c, t_i)$, for each category $c \in \mathcal{C}$ and each keyword $t_i \in \mathcal{Q}$. However, only the term frequencies $tf_{\text{rt}(c)}(c, t_i)$ are available in the statistics. We estimate $tf_{s^*}(c, t_i)$ by analyzing how the term frequency of t_i in c changed in the recent past. This is based on the principle of temporal locality which assumes that the term frequencies do not change dramatically.

Suppose for a term t and a category c, *rate of change of term frequency* is denoted by $\Delta(c, t)$ and is an estimate of the change in the term frequency per data item added to the system. Notice that this would be dependent on two factors: (i) The selectivity of the category c, i.e., out of say p data items added to the information repository, how many are likely to belong to c's data-set and (ii) the change in the term frequency for each new data item added to c's data-set. The system could keep track of these two factors to predict the value of $\Delta(c, t)$. Thus the value of $\Delta(c, t)$ over a sequence of time-steps can be seen as a time series and we are required to predict its value at a time-step in the future based on its values in the past. Given such a predicted value of $\Delta(c, t)$, we estimate $tf_{s^*}(c, t_i)$ as follows:

$$tf^{est}_{s^*}(c, t_i) = tf_{\text{rt}(c)}(c, t_i) + \Delta(c, t_i) \times (s^* - \text{rt}(c)) \tag{5}$$

In the sum on the RHS, the first component is the known term frequency till time-step $\mathrm{rt}(c)$ and the second component is the estimated change in its value due to new data items added between $\mathrm{rt}(c)$ and s^*. The Δ values corresponding to a category c are updated whenever the category c gets refreshed.

For the sake of completeness, we provide one example technique for deriving the Δ values. However, note that our system is independent of the exact mechanism used for this purpose. The Δ value can be computed by looking at the changes in the term frequencies in the recent past and using an exponentially smoothed value. Let c be a category and let its last two refresh time-steps be s_2 and s_1 (with $s_2 > s_1$). For a term t, let $\Delta_{s_1}(c,t)$ be the Δ value computed at time-step s_1. At time-step s_2, we compute Δ_{s_2} as follows:

$$\Delta_{s_2}(c,t) := Z \cdot \frac{tf_{s_2}(c,t) - tf_{s_1}(c,t)}{s_2 - s_1} + (1 - Z) \cdot \Delta_{s_1}(c,t)$$

where Z $(0 \leq Z \leq 1)$ is a smoothing constant. By setting $Z > 0.5$, we can give more importance to the recently added data items. While answering a query at time-step s^*, in order to get an estimate $tf^{est}_{s^*}(c,t_i)$ on the exact value of $tf_{s^*}(c,t_i)$, we use Equation 5 with $\Delta_{\mathrm{rt}(c)}(c,t)$ as the Δ value.

Turning to inverse document frequencies, ideally, one would like to maintain $idf_{s^*}(t)$ for each term t. However, this requires refreshing the statistics whenever a data item is added, because adding a data item can change the idf of every term contained in it. So, here-again, we shall only maintain an estimated idf (i.e., $idf_{est}(t)$), for each term t. The details are presented while discussing the meta-data refresher module (Section IV).

IV. Meta-Data Refresher

The Meta-data Refresher uses the selective update strategy for maintaining and refreshing the statistics as accurately as possible. During each invocation it first identifies the set of 'important' categories (of size N) and then identifies the set of data items (of size B) which can provide the maximum impact to the selected categories in terms of update to statistics. The meta-data refresher then refreshes the chosen categories using the selected data items. Once it has finished refreshing, the meta-data refresher is invoked again and the process is repeated.

The first and foremost task for the meta-data refresher is that of finding the importance of a category. As we shall see in Section IV-A, the meta-data refresher uses the predicted query workload to compute the importance and selects the N categories with maximum importance. Given the N important categories the issue that needs to be addressed next is: How to find the right set of data items that would provide the maximum impact? Answering this question requires us to first define the benefit (or impact) of a set of data items for a category. This is the focus of Section IV-B. The benefit is then used by our dynamic programming algorithm (details in Section IV-C) to find the optimal set of data items to refresh the selected important categories.

The above discussion assumes some value of N and B. These values are computed during each invocation of the meta-data refresher based on the system's performance as explained in Section IV-D. The meta-data refresher also has the additional responsibilities for maintaining the idf values and for handling new categories added to the system. These details are outlined in Section IV-E and Section IV-F respectively.

A. Determining Important Categories

We find the importance of a category based on the predicted query workload. The meta-data refresher uses the keyword queries seen in the past to predict the query workload. Thus the predicted query workload $\mathcal{W}$ is simply a multi-set of keywords that were queried in the recent past. This can be easily computed by keeping track of the queries posed to the system.

For the predicted query workload $\mathcal{W}$, we use the following principle in computing the importance of a category: *a category that is highly relevant to a large number of keywords in $\mathcal{W}$ should have higher importance; a category is relevant to a keyword, if the category is one among the top-ranked categories for the keyword (query).* Based on the above principle the importance of a category is defined as follows. For a keyword $t \in \mathcal{W}$, we define its *candidate set* to be the set of top-$2K$ categories for t. The candidate set can be easily computed by the Query Answering module while answering the keyword query (details in Section V). Our formula for calculating the importance also takes into account the frequency of occurrence of the keywords in $\mathcal{W}$. We define the *weight* of a keyword $t \in \mathcal{W}$ to be the number of times it occurs in the predicted workload $\mathcal{W}$. Then the importance of a category c is calculated as the sum total of the weights of all the keywords in whose candidate sets the category c appears:

$$Importance(c) = \sum_{t \in \mathcal{W} \wedge c \in CandidateSet(t)} weight(t) \quad (6)$$

The above formula ensures that the categories that appear in the candidate set of multiple keywords have higher importance. The $Importance(c)$ so calculated can be thought of as a measure of the likelihood of the category being used for answering some query in the future. Thus, if there is no drastic change in the query workload, refreshing a category with high importance would help us to improve the accuracy of the query results by a large amount.

In CS*, the meta-data refresher chooses the N categories with maximum importance. We represent these categories by $\mathcal{IC}$. The size of $\mathcal{W}$ in Equation 6 is set to U which is called as the query workload prediction window.

B. Strategy for Refreshing Categories

Given the set $\mathcal{IC}$, the next task is to choose the set of B most beneficial data items to refresh the N categories. In order to solve this problem we need to find answers to four important questions, namely: (i) When can a data item be useful for a category? (ii) How can we define the benefit of a data item for a category? (iii) How do we reduce the number of data items that we need to consider (from among those not considered so far)? and (iv) How to find the optimal set of data items

to refresh the selected important categories? We provide an answer to each of these questions in this section.

As mentioned in Section III, we ensure that the categories are refreshed contiguously. In order to ensure the contiguous property, our algorithm selects the needed B data items in the form of ranges. For $i \leq j$, let the range $[i, j]$ represent the set of all data items added between time-steps s_i and s_j, namely the set $d_i, d_{i+1}, \ldots, d_j$. Let $width([i, j])$ denote the number of data items in the range. Our goal is to choose a set of ranges $\mathcal{R}$ with the constraint that the sum of the width of the ranges in $\mathcal{R}$ should be at most B. Notice that selecting two overlapping ranges (i.e., $[i_1, i_3]$ and $[i_2, i_4]$ such that $i_1 \leq i_2 \leq i_3$) is equivalent to selecting the combined range (i.e., $[i_1, i_4]$).

Once a set of ranges $\mathcal{R}$ is chosen, the categories in $\mathcal{IC}$ are refreshed with respect to data in these ranges. In order to determine if a category $c \in \mathcal{IC}$ can be refreshed using a range $[i_1, i_2] \in \mathcal{R}$, we need to consider the following three cases:

1) $\mathrm{rt}(c) > i_2$: Here, the range is of no use to c, as c has already been refreshed using the data items in this range.
2) $i_1 \leq \mathrm{rt}(c) \leq i_2$: Here, c can be refreshed using the data items in the range $[\mathrm{rt}(c), i_2]$.
3) $\mathrm{rt}(c) < i_1$: Here, we cannot refresh c using the data items in this range, because it would lead to the violation of the contiguous refreshing property.

Our next task is to design a mechanism for measuring the benefit of a range $[i_1, i_2]$, denoted by $Benefit([i_1, i_2])$. This is calculated using the benefits that this range provides for each category in the set $\mathcal{IC}$. Consider a category $c \in \mathcal{IC}$. The benefit that the range $[i_1, i_2]$ provides for c is simply the number of data items in this range that can be used to refresh c. This benefit is denoted by $Benefit([i_1, i_2], c)$ and is calculated as below:

1) If $\mathrm{rt}(c) > i_2$, $Benefit([i_1, i_2], c) = 0$.
2) If $i_1 \leq \mathrm{rt}(c) \leq i_2$, $Benefit([i_1, i_2], c) = i_2 - \mathrm{rt}(c)$.
3) If $\mathrm{rt}(c) < i_1$, $Benefit([i_1, i_2], c) = 0$.

The overall benefit of $[i_1, i_2]$ is calculated based on the benefit it offers to each category in $\mathcal{IC}$, taking also into account the importance of the categories:

$$Benefit([i_1, i_2]) = \sum_{c \in \mathcal{IC}} Importance(c) \times Benefit([i_1, i_2], c)$$

The above formula is designed to ascribe more benefit to a range that is beneficial to an important category. Extending the above formula, we define the benefit of a set of ranges $\mathcal{R}$ as the sum of benefits of all the ranges in it.

Let us now analyze the set of ranges that we need to consider. Notice that any range in the interval from 1 to the current time-step s^* is a possible candidate; this means that $\binom{s^*}{2}$ ranges are possible. This being a very large number, we try to reduce the number of candidate ranges so as to improve the efficiency of our algorithm (for finding the optimal set of ranges), without having a large impact on its accuracy.

In order to do this, we first arrange the categories in $\mathcal{IC}$ in ascending order of their last refresh time. Let this ordering be $c_1, c_2, \ldots, c_N$. Consider a range $[i_1, i_2]$ which is contained in between two consecutive refresh times, i.e., for some k, $\mathrm{rt}(c_k) < i_1 < i_2 < \mathrm{rt}(c_{k+1})$. The benefit of this range is 0 and so, such ranges can be ignored. Next consider any range $[i_1, i_2]$ such that i_1 does not coincide with the last refresh time of any category in $\mathcal{IC}$. This means that for some k, $\mathrm{rt}(c_k) < i_1 < \mathrm{rt}(c_{k+1})$ and by previous observation, $i_2 > \mathrm{rt}(c_{k+1})$. Notice that we can shift the starting point of the range to $\mathrm{rt}(c_{k+1})$ without incurring any loss in benefit, i.e., $Benefit([\mathrm{rt}(c_{k+1}), i_2]) = Benefit([i_1, i_2])$. Hence, we can ignore such ranges. Even after the above pruning of ranges, we are still left with all the ranges that start at $\mathrm{rt}(c_k)$, for some $c_k \in \mathcal{IC}$; the number of such ranges can be seen as $\Sigma_{i=1}^{N}(s^* - \mathrm{rt}(c_i))$. The above number is a function of s^* and hence, it is large. We wish to keep the number of candidate ranges as a function of N and we achieve this by considering only those ranges that end at some $\mathrm{rt}(c_k)$, for some k [1]. Let $[i_1, i_2]$ be a range such that $\mathrm{rt}(c_k) < i_2 < \mathrm{rt}(c_{k+1})$, for some k. By shifting the ending point to $\mathrm{rt}(c_k)$, we would lose a small amount of benefit; but this allows us to reduce the number of ranges to be considered drastically. To summarize, we shall only consider the $\binom{N}{2}$ ranges that start and end at some last refresh time. We call such ranges as *nice ranges*. The above discussion serves as the motivation of the *range selection problem*.

C. Range Selection Problem

The input to the range selection problem is a sequence of categories $\mathcal{IC} = c_1, c_2, \ldots, c_N$ sorted in the ascending order of their last refresh times and a bandwidth B. The goal is to find a set of non-overlapping nice ranges having a total width of at most B, such that the total benefit is maximized. In this section we present a dynamic programming algorithm for the range selection problem.

For $1 \leq i < j \leq N$, let NR_{ij} denote the (nice) range $[\mathrm{rt}(c_i), \mathrm{rt}(c_j)]$. The input to the algorithm is the set of all nice ranges represented by $\mathcal{NR}$. Our goal is to find a subset $\mathcal{R} \subseteq \mathcal{NR}$ of non-overlapping nice ranges such that $width(\mathcal{R}) \leq B$ and $Benefit(\mathcal{R})$ is maximized. The algorithm constructs an $N \times B$ matrix E, whose (k, b) entry contains the maximum benefit that can be achieved by using only the categories $c_1, c_2, \ldots c_k$ and a bandwidth of b. Observe that $E[N, B]$ gives the benefit of the optimal solution to our original problem.

The entry $E[k, b]$ can be inductively computed from entries $E[k-1, \cdot]$. Consider the optimal solution $\mathcal{R}^*$ corresponding to the entry $E[k, b]$. The set $\mathcal{R}^*$ can either include a range of the form NR_{jk} (for some j) or not include any such range. In the latter case, $\mathcal{R}^*$ is also the optimal solution corresponding to the entry $E[k-1, b]$. On the other hand, suppose $\mathcal{R}^*$ includes a range of the form NR_{jk}, for some $1 \leq j < k$. Then the set $\mathcal{R}^* - \{NR_{jk}\}$ is the optimal solution corresponding to the entry $E[j, b - Width(NR_{jk})]$.

Based on the above observations, we can derive the following recurrence for the entries of E. For $1 \leq k \leq N$ and

[1] The ranges can also end at s^* which can be handled by adding an imaginary category c_{img} with $\mathrm{rt}(c_{img}) = s^*$ to $\mathcal{IC}$.

$1 \leq b \leq B$,

$$E[k,b] = \max \begin{cases} E[k-1,b] \\ \max\limits_{1 \leq j < k} \{Benefit(NR_{jk}) + E[j, b - Width(NR_{jk})]\} \end{cases}$$

The dynamic programming algorithm uses the above recurrence relation to compute the matrix E. The algorithm is given in detail in [4].

The matrix E has $N \cdot B$ entries and it takes $O(N)$ time to compute each entry. Thus, the running time of the algorithm is $O(N^2 \cdot B)$. Notice that if we had not reduced the number of ranges to be considered from $\binom{s^*}{2}$ to $\binom{N}{2}$, the running time would have been $O((s^*)^2 \cdot B)$ where $s^* >> N$. This justifies our strategy of reducing the number of input ranges.

Justification for contiguous refreshing: Consider a system CS' where the categories are not refreshed in a contiguous manner. For such a system, $\mathrm{rt}(c)$ would represent the number of (non-contiguous) data items using which c has been refreshed. Due to the non-contiguous refreshing property the dynamic program will have to select a set of 'B' data items (and not ranges). Due to the absence of ranges, the size of the input to the dynamic program would be: $\sum_{c \in \mathcal{IC}} (s^* - \mathrm{rt}(c))$. This being a function of the current time-step s^*, it will be very large compared to the input size of $\binom{N}{2}$ in the CS* system. Such a large input size would adversely affect the number of categories that the meta-data refresher can refresh in one invocation, as we shall see in the next section. Hence the CS* system is designed to refresh the categories in a contiguous manner.

Parallelization of meta-data refresher: Once the meta-data refresher chooses the nice ranges $\mathcal{IC}$ of width B and the set of important N categories, the job of refreshing the categories can be executed in parallel over $B \times N$ processors. If the number of available processors p is less than this, then the meta-data refresher distributes it evenly among these p processors. Each of the processors updates the statistics stored at a central location from where it is accessed by the Query Answering module. In this paper we focus on the core algorithms used by the meta-data refresher and do not get into the synchronization issues, which are beyond the scope of this paper.

D. Selecting B and N Values to Keep Up with Data Addition

Let γ be the average units of time (including communication costs) required to refresh a single category using a data item per unit processing power. If the available processing power is p, then the update-all technique will require $\frac{\gamma \times |\mathcal{C}|}{p}$ units of time to refresh all the categories ($\mathcal{C}$) using one data item. Let the rate of arrival of data be α i.e., α data items are added per unit time. As $\frac{\gamma \times |\mathcal{C}|}{p} > \frac{1}{\alpha}$, the update-all technique starts lagging behind the data item addition rate.

The meta-data refresher accesses B data items and uses them to refresh N categories. Thus the time taken for one invocation of the meta-data refresher would be: $\frac{B \cdot N \cdot \gamma}{p}$. If a new data item is added before the invocation finishes, the meta-data refresher would start lagging behind the data item addition rate. Thus the maximum time available for a single invocation would be: $\frac{1}{\alpha}$. Thus,

$$\frac{B \cdot N \cdot \gamma}{p} = \frac{1}{\alpha} \implies N = \frac{p}{\alpha \cdot B \cdot \gamma} \tag{7}$$

Given a value of B, we can use the above equation to find the value of N and vice-versa. Consider the system at the first time-step i.e., when only one data item has been added. For such a system, the value of B will be 1 as we cannot refresh a category using say, a fraction of a data item. With $B = 1$, the value of N can be computed from Equation 7. If the data items in the set $\mathcal{IC}$ do not change over time, then these values of B and N would ensure that the statistics associated with the categories in $\mathcal{IC}$ are always maintained up-to-date i.e., their staleness ($L = \sum_{c \in \mathcal{IC}} (s^* - \mathrm{rt}(c))$) would be zero. However, in practice the categories in $\mathcal{IC}$ would change due to the change in query workload. The staleness of this new set of important categories would thus be greater than zero. If we continue to use the same value of B and N, then there would be no improvement in the staleness and the quality of query results would suffer. The query results can be improved by reducing the staleness of the important categories which would require an increase in the value of B. However, if we simply increase the value of B without changing the value of N, it would lead to an increase in the time taken by the meta-data refresher. Consequently, the meta-data refresher would start falling behind the data item arrival rate. This can be avoided by reducing the value of N such that the time taken by the meta-data refresher is constant. Thus given an increased value of B, the value of N can be computed using Equation 7.

Now the problem that needs to be addressed is to find the right increase in the value of B. We compute the value of B using the following feedback mechanism. During each invocation, the meta-data refresher first finds the staleness of the N important categories, where N is set to its value used during the previous invocation. It keeps track of the minimum and maximum value of the staleness of the top N categories seen in the past i.e., $[L_{min}, L_{max}]$. During the current invocation, if the staleness of the N categories is the maximum seen so far, then we need to focus on a smaller set of categories and reduce their staleness. Hence, we set $N = 1$ and compute B_{max} value using Equation 7. If on the other hand the staleness is the least seen so far, then the value of B is set to 1 and the value of N is computed accordingly. For other values of staleness (L'), we set the value of B in proportion to the quantity: $\frac{L' - L_{min}}{L_{max} - L_{min} + 1}$. This ensures that if say, the range $[L_{min}, L_{max}]$ is $[10, 20]$ and the value of L' is 14, then the value of B will be set to 40% of its maximum value (B_{max}). Hence if the staleness increases, the value of B will increase proportionately. Thus during each invocation, the meta-data refresher sets the value of N and B such that it does not start falling behind the data item arrival rate. Note that if the data item arrival rate slows down sufficiently, then CS* behaves like the update-all technique.

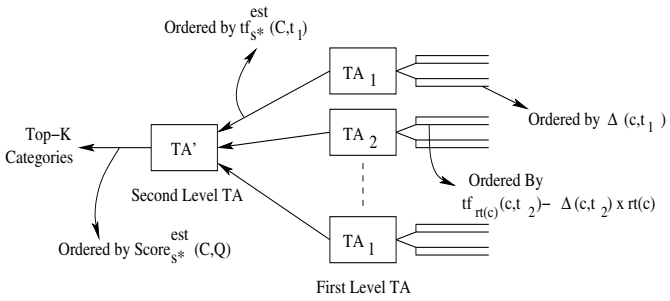

Fig. 2. Two-level Threshold Algorithm

E. Estimating idf

The meta-data refresher is also responsible for maintaining the estimated value of the inverse document frequency for each keyword (i.e., idf_{est}). The crucial difference between the term frequency and the inverse document frequency is that unlike the term frequency, the idf value does not change significantly with time. The meta-data refresher computes the $idf(t)$ value by finding the number of categories having $tf(\cdot, t) > 0$ (i.e., $|\mathcal{C}'|$ value in Equation 2). Over a period of time as the tf statistics get updated, the idf value approaches its correct value and it does not change significantly over time. Hence CS* uses the previous known value of the idf which turns out to be good estimate for the actual value.

F. Handling New Categories

We now consider the case when a new category is added to the system. Note that the rate of addition of a new category will be very low. Whenever a new category is added to the system, we refresh the category fully till current time-step s^* and update the statistics accordingly. The importance of the category can also be computed by analyzing the predicted query workload. Once these statistics have been computed, the new category gets integrated into the system.

V. Query Answering Module

Given a keyword query $\mathcal{Q} = \{t_1, t_2, \ldots, t_\ell\}$ issued at the current time-step s^*, the goal of the query answering module is to find the top-K categories using the following scoring function:

$$\text{Score}^{est}_{s^*}(c, \mathcal{Q}) = \sum_{t_i \in \mathcal{Q}} tf^{est}_{s^*}(c, t_i) \times idf^{est}_{s^*}(t_i) \qquad (8)$$

We build on ideas from the Threshold Algorithm (TA) [5] to efficiently find the top-K categories. Given a set of objects, the threshold algorithm finds the top-K objects with respect to a scoring function, where the function is made of ℓ components and the overall score of an object is determined by applying a monotone aggregation operator to the scores given by the ℓ components. In our case, the categories are the objects. The overall score of a category c with respect to a keyword query $\mathcal{Q}$, is a sum of the scores with respect to the ℓ keywords. Thus, the scores with respect of each keyword $t_i \in \mathcal{Q}$ (i.e., $tf^{est}_{s^*}(c, t_i) \times idf^{est}_{s^*}(t_i)$) is an individual component in the overall score.

The TA algorithm requires ℓ sorted lists, where the j^{th} list provides an ordering of the input objects on the j^{th} component of the overall score. Thus, in our case, for each keyword t_i, we need a sorted list σ_i that provides an ordering of all categories based on $tf^{est}_{s^*}(\cdot, t_i) \times idf^{est}_{s^*}(t_i)$. However, the main problem is that we cannot maintain the required sorted list σ_i. This is because, the value $tf^{est}_{s^*}(\cdot, t_i)$ is a function of s^* (see Equation 5) and hence, the sorted order could change with a change in s^* i.e., with the addition of each new data item. Hence, in principle, we need to recompute the sorted lists every time a keyword query is asked. We avoid this re-computation by employing a keyword level threshold algorithm.

An overview of the two-level threshold algorithm is given in Figure 2. For the user query $\mathcal{Q} = \{t_1, t_2, \ldots, t_\ell\}$, the query answering module sets up ℓ keyword level threshold algorithms, one for each keyword $t_i \in \mathcal{Q}$. The output of these ℓ keyword level threshold algorithms is fed to the query level threshold algorithm TA'. The keyword level threshold algorithm TA_i is responsible for outputting the categories according to the ordering σ_i and the query level threshold algorithm finds the top-K categories with respect to the overall score $\text{Score}^{est}_{s^*}(\cdot, \mathcal{Q})$. For the ease of exposition, we first discuss the scenario of single keyword queries ($\ell = 1$) which only requires the use of the keyword level TA and then explain the general case requiring the TA at two levels.

A. Answering Single Keyword Queries

For a query consisting of a single keyword t, the estimated score of a category c is $\text{Score}^{est}_{s^*} = tf^{est}_{s^*}(c, t) \times idf^{est}_{s^*}(t)$, where s^* is the current time-step. Since the idf is a common multiplicative factor across all categories, the ordering of the categories depends only on the estimated term frequency $tf^{est}_{s*}(\cdot, t)$. The tf^{est}_{s*} value depends on the current time-step s^* and hence keep changing; so it is difficult to maintain an ordered list of categories based on their $tf^{est}_{s^*}(\cdot, t)$ values. Here, we propose an efficient algorithm for computing the required sorted list at the current time-step.

Let us take a closer look at the function $tf^{est}_{s*}(c, t)$. Rewriting Equation 5, we see that:

$$\begin{aligned} tf^{est}_{s^*}(c, t) &= tf_{\text{rt}(c)}(c, t) + \Delta(c, t) \times (s^* - \text{rt}(c)) \\ &= tf_{\text{rt}(c)}(c, t) - \Delta(c, t) \times \text{rt}(c) + \Delta(c, t) \times s^* \end{aligned} \qquad (9)$$

We split the RHS into two components: (i) $tf_{\text{rt}(c)}(c, t) - \Delta(c, t) \times \text{rt}(c)$ and (ii) $\Delta(c, t) \times s^*$. The first component is independent of s^* and its value changes only when the category c is refreshed. On the other hand, the second component is dependent on s^*. But the interesting fact is that the value of the current time-step s^* is common for all the categories. Our solution exploits this insight as described below.

We maintain an inverted index that provides a mapping from each term to the set of categories containing the term. For each term t, the inverted index consists of two different sorted lists of categories: (i) the first list sorts the categories in the descending order of the first component (i.e., $tf_{\text{rt}(c)}(c, t) - \Delta(c, t) \times \text{rt}(c)$); and (ii) the second list sorts the categories in the descending order of their Δ values i.e., $\Delta(c, t)$. We use

the threshold algorithm [5] to merge these two lists to get the top-K categories. A sketch of this procedure is presented next.

Let O_1 and O_2 be the two orderings of the categories based on the value of the first component and the Δ value respectively. We do a parallel scan of the two lists iteratively using two cursors. We keep a buffer of all the top-K categories seen so far in the scan. Consider any iteration of the scanning process. Let c_1 and c_2 be the categories under the two cursors. We compute the term frequencies $tf^{est}_{s*}(c_1, t)$ and $tf^{est}_{s^*}(c_2, t)$ and choose the category having the higher term frequency. Without loss of generality, suppose c_1 is the chosen category. We want to see if c_1 is one among the top-K categories and if so, include it in the buffer. Towards that end, we perform the following test: Find the category $\bar{c}$ that has the lowest $tf^{est}_{s^*}(\cdot, t)$ score among all the categories in the buffer; if $tf^{est}_{s^*}(c_1, t) > tf^{est}_{s^*}(\bar{c}, t)$, then delete $\bar{c}$ from the buffer and add c_1. Next, we repeat the procedure for c_2. The above procedure ensures that the buffer always contains the top-K categories among those seen so far. The scanning process is terminated if $tf^{est}_{s^*}(\bar{c}, t)$ is larger than the maximum possible term frequency achievable by any category not seen so far. Namely, we terminate the scanning process if,

$$tf^{est}_{s^*}(c, t) \geq [tf_{\mathrm{rt}(c_1)}(c_1, t) - \Delta(c_1, t) \times \mathrm{rt}(c_1)] + \Delta(c_2, t) \times s^*.$$

If the process did not terminate, we advance the two cursors and go to the next iteration. Once the above procedure terminates, we simply output the categories in the buffer. It is fairly easy to see that the procedure outputs the top-K categories. A formal proof of correctness can obtained by adapting the arguments from [5].

B. Answering Multiple Keyword Queries

The algorithm presented in the previous section works for single keyword queries. We now address the issue of extending the technique to handle a multiple keyword query $\mathcal{Q} = \{t_1, t_2, \ldots, t_\ell\}$. Recall Equation 8, which provides the approximate score $\mathrm{Score}^{est}_{s^*}(c, \mathcal{Q})$ of a category c with respect to $\mathcal{Q}$. Our goal is to find the top-K categories according to this score.

Consider any keyword $t_i \in \mathcal{Q}$. Let σ_i be the ordering of all the categories based on the scores with respect to t_i (i.e., $tf^{est}_{s^*}(\cdot, t_i) \times idf^{est}_{s^*}(t_i)$). We can use the algorithm given in the previous section to obtain the sorted list σ_i for each keyword t_i. We then use the Threshold Algorithm (called the query level TA) to merge these ℓ lists to get the desired top-K categories. The pseudo code of the Two Level Threshold algorithm is given in [4].

VI. Experimental Evaluation

In this section, we discuss the experimental evaluation of CS*.

A. Experimental Setup

Dataset: We used data from the site *www.citeulike.org* for our experiments. This site is a free online service to organize academic papers from different disciplines. People can upload their academic papers on this site and can also post reviews on the posted papers. The site also offers a *"who-posted-what"* dataset which has details about all the postings made on the site along with the timestamp when they were posted. The dataset also has the tags associated with the posted article. Using the article ids present in the dataset we crawled the site to retrieve the documents. We thus created a trace of the documents posted on the site along with the time when they were posted and the tags associated with them. We used this trace in our experiments which consisted of 100,000 articles posted after 30-May-2007. The experiments were conducted by employing a trace replay. Recall from Section I, where in the blog example, 13 blog entries are generated per second. For a data generation rate of 13 articles per second, 100000 articles translates to 2.14 hours of data. In order to simulate a high rate of arrival of data items, we increased the speed of the clock during the trace replay to ensure that say, 13 articles are generated per second.

Categories: The categories were generated using the tags associated with the articles with each tag representing one category. Thus a category like say 'asthma' would have all the articles related to asthma. Hence, the dataset in our experiments can be considered to have been manually (pre)classified due to the presence of the tags with the articles. Our downloaded dataset had about 5000 distinct tags (i.e., categories).

Query Workload: Various studies [6], [7] on query logs of different search engines suggest that they follow a a $Zipf$ distribution [8]. Hence we generated the query workload using a $Zipf$ distribution (with moderate skew i.e., $Zipf$ parameter $\theta = 1$) over the keywords present in all the documents in our corpus. Each query consisted of 1 to 5 keywords.

If a keyword query contains a keyword that appears in a large number of documents, a large number of categories will be relevant to the query. These categories coupled with the continuous addition of new data items will ensure that the ranking of the top-K categories will continuously keep changing. Hence the possibility of an incorrect result is maximum when the keyword query contains such a keyword. In order to do a thorough evaluation of CS*, we ensured that the frequency of occurrence of a keyword in the query workload was proportional to its frequency in the trace.

Processing Power: Consider a deployment of CS* on 10 machines i.e., the meta-data refresher would have 10 processors available for running the refresh process in parallel. If the data arrival rate is 15 data items per second and it takes 1 second to refresh a category using one data item on one machine (including communication costs), then in one second the 10 machines would have processed 10 data items and there would be 5 data items left unprocessed (if $|\mathcal{C}|$=1). We simulated this on a single machine by modelling time as follows: In 10 ticks of simulation time, 15 data items are added to the system. Thus at the end of 10 ticks of simulation time, the single machine would have processed 10 data items and 5 data items would be left unprocessed. Thus 10 ticks of simulation time will be similar to the one second of the actual setup.

Categorization Time: The total time required to determine all the categories a single data item belongs to is referred to as

categorization time. Our analysis using real classifiers (Naive Bayes Classifiers) showed that this can vary between 15 to 75 seconds for our setup (assuming some mutual exclusion between categories). In our dataset, as the data items were pre-categorized, we simulated the different type of classifiers by adding a delay proportional to the time that would be required to classify the data items. Our experiments were run on server grade windows machines with 4 GB RAM and 3.6 GHz processors and used the exponentially smoothed estimator given in Section III (with $Z = 0.5$) to estimate the value of Δ.

Accuracy: For a keyword query Q, let the top-K results returned by CS* be: $Re = \{c_1, c_2, \ldots, c_K\}$. Consider a system that has the refreshed statistics for all the categories for all data items till current time-step s^*. Let the results returned by such a system be: $Re^{'} = \{c_1^{'}, c_2^{'}, \ldots, c_K^{'}\}$. Then the accuracy of CS* for a keyword query is defined as follows: $Accuracy = \frac{|Re \cap Re'|}{K}$. E.g., if $Re = \{c_1, c_2, c_3\}$ and $Re^{'} = \{c_1, c_4, c_2\}$ and $K = 3$, then the accuracy of the system is 66%. The accuracy of CS* for a set of queries is simply the average of its accuracy across all the queries in the set. The correct query results ($Re^{'}$) were determined by using a system that refreshes all the categories every time a new data item is added. It is important to note that such a system takes a huge amount of time to update the statistics and answer the query.

Notice that for a top-K setup, this definition of accuracy is the same as that of precision used in IR literature. Similarly, the recall metric for a top-K setup measures the following quantity: Out of the documents which are actually in the top-K, how many are reported to the user. Notice that this is also same as the accuracy defined above.

B. Experimental Results

We conducted two sets of experiments. The first set of experiments was to measure the accuracy of the CS* system. These sets of experiments thus evaluate the efficiency of the meta-data refresher and are presented next. The second set of experiments evaluate the query answering module.

Evaluation of Meta-data Refresher: In this section we evaluate the effectiveness of the meta-data refresher by measuring the accuracy of the CS* system. The nominal value of the various parameters used in these experiments is summarized in Table I. Note that a processing power of 500 need not represent 500 processors. It could represent, say 50 machines, each of which can categorize a data item in $\frac{\gamma}{10}$ time units. The rate of creation of new blog posts suggested in [1] is 13 blog entries per second (i.e., α=13). However, this being a 2006 estimate the nominal value of α was set to 20 in our experiments.

Accuracy with varying processing power: We studied the effect of increasing processing power on accuracy. As is expected, the accuracy of both CS* and update-all technique improves with increase in processing power. However, Figure 3 shows that CS* comprehensively outperforms the update-all technique and is able to achieve an accuracy in excess of 90% using a processing power of 300. The update-all technique, on the other hand requires 65% more processing power to provide similar accuracy. *Thus CS* can achieve a user specified accuracy requirement by increasing the processing power – at a cost which is significantly lower than that of the update-all technique.* Note that the accuracy of update-all does not improve till the processing power is significantly increased (i.e., around 450), when it stops lagging behind the data item addition rate.

TABLE I

PARAMETERS RANGES AND NOMINAL VALUE USED IN THE EXPERIMENTS

Parameter	Range of Values Tested	Nominal Value
α	2 to 20	20
Categorization Time	15 to 75	25
Number of data items	25K to 100K	25K
Processing power	2 to 500	300
ℓ	1 to 5	3
U	10	10
K	10	10

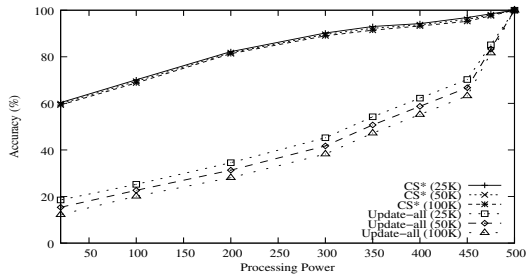

Fig. 3. Accuracy with varying processing power and number of data items

Scalability with respect to number of data items: Figure 3 also shows the effect of the number of data items on the accuracy. The accuracy of the update-all technique has a noticeable reduction with an increase in the number of data items. However, contrary to expectations, there is no such drop in the accuracy of CS*. At any time-step s', the update-all technique would have examined $\frac{s'p}{|\mathcal{C}|\gamma}$ data items. Hence an increase in the number of data items leads to a proportional increase in the staleness of the meta-data for all the categories. Hence the accuracy of the update-all technique suffers with an increase in the number of data items. CS* on the other hand focuses on the important categories and keeps them refreshed. Because CS* ignores the unimportant categories it is likely to lead to a small error for queries related to them. With an increase in the number of data items, the staleness of the meta-data of these unimportant categories increases. However, this does not lead to a decrease in the accuracy of our results (as these categories were being ignored even when the number of data items in the system were less). Thus, this showcases the scalability of the meta-data refresher module.

Accuracy with varying categorization time: In this set of experiments, we evaluated the accuracy of CS* by varying the categorization time using a processing power of 300. Figure 4 shows that even when the categorization time becomes very high, CS* is able to provide very good accuracy which is much better than that of the update-all technique. Notice that

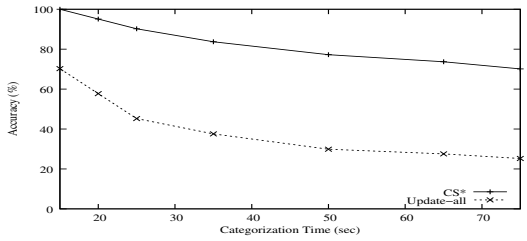

Fig. 4. Accuracy with varying categorization time

the accuracy of CS* can be further improved by increasing the processing power. The gain due to increase in the processing power is much more for CS* than that for the update-all technique as is evident from Figure 3.

Scalability with respect to arrival rate: We studied the effect of the rate of arrival of data items (α) on the accuracy. The CS* system and the update-all technique will provide 100% accuracy if they can refresh all the categories using a data item before a new one arrives. Thus the amount of processing power required will depend on the rate of arrival of data items. If α value doubles, then generally the processing power needs to double to maintain an accuracy of 100%. For each value of α, we first found out the processing power required to achieve 100% accuracy for the update-all technique. For each value of alpha, in this experiment, the processing power was set to 50% of its value required to achieve 100% accuracy (using update-all). The graph shows that contrary to expectations, as the rate of arrival of data items increases the accuracy of CS* also increases. *In other words, if the rate of arrival of data items doubles, then CS* will not require double the processing power to provide the same level of accuracy.* As is evident from the graph, the update-all technique on the other hand requires more than double the processing power to maintain its accuracy. This counter intuitive result is due to the fact that with the increase in the rate of arrival of data items and processing power, the important categories get refreshed using more number of data items. So the statistics of the important categories are maintained much better. This leads to an increase in the accuracy of CS* system.

Figure 5 also shows the accuracy provided by the sampling based refresher described in Section II. Such a refresher samples the data items and refreshes all the categories using it. For computing the idf value it uses a strategy similar to that used by CS*. The accuracy of this refresher is slightly better than that of the update-all technique. The difference between this technique and update-all is that, update-all refreshes the categories using all the data items in the order of their arrival. In the sampling based refresher this order is not maintained and some data items get skipped. In our experimental data, data items appearing in a time window would be similar to each other. E.g., papers posted in one day would be related to the conferences whose acceptance notification has arrived in the recent past. Hence, the sampling based refresher gets more diverse data items (due to skipping some of them) as compared to the update-all technique. This helps to update the statistics of more number of categories, which results in slightly better results.

TABLE II
SAMPLE PARAMETER COMBINATIONS THAT PRODUCE 90% ACCURACY

α	Categorization Cost	Processing Power		Extra Power Required
		CS*	Update-all	
20	25	300	493	64.33%
20	50	594	982	65.31%
10	25	155	244	57.42%

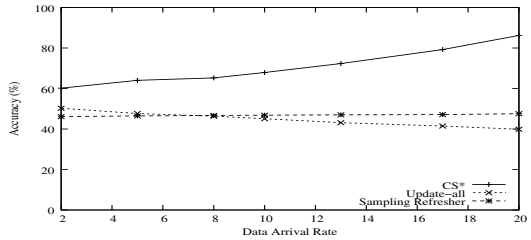

Fig. 5. Accuracy with varying rate of arrival of data items

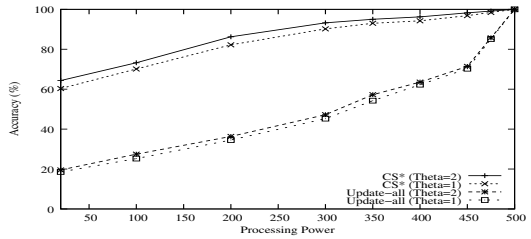

Fig. 6. Accuracy with varying skew in workload

Accuracy under changing query workload: In this set of experiments, we tried to find the effect of the skew in the query workload on the accuracy of CS*. The skew in the workload was increased by setting the zipf parameter θ to 2. This leads to an increase in the accuracy of CS* as seen from Figure 6. The increase in the skew leads to a lesser change in the set of important categories. This allows the meta-data refresher to focus on the important categories for longer periods of time which leads to an improvement in the accuracy.

Evaluation of Query Answering Module: We analyzed the efficiency of the two level threshold algorithm by measuring the number of categories considered by the algorithm in determining the top-K results. We found that the two-level threshold algorithm analyzes only 20% of the categories to find the top-K result. The running time of the query answering module was very small (of the order of milliseconds). Notice that in the absence of the two-level threshold algorithm, a normal query answering module will have to compute the current statistics of all the categories, sort them and then return the top-K categories. In the stock exchange scenario of Section I, this would require at least 80% more time (as we analyze only 20% of the categories), when analyst would expect quick answers.

In summary, the CS* system provides an accuracy significantly better than the update-all technique using a fraction of its resources. Table II summarizes the various parameter settings under which CS* can provide over 90% accuracy. It clearly shows that CS* outperforms the update-all technique and leads to a saving in processing power of more than 57%. These point out the effectiveness of the various aspects of decision making, built into CS*, making it a practical solution for the problem of keyword search over categorized data.

VII. Related Work

The problem of ranking categories which map to a set of dynamic data items broadly encompasses two areas (i) Ranking of documents in information retrieval and (ii) efficiently maintaining statistics related to dynamic data. There has been a large body of work on document ranking. Our work does not focus on developing the best ranking technique and we use the $tf \cdot idf$ [2] based ranking measure to describe our system. Notice that CS* can be easily made to work for other types of scoring functions such as cosine distance as it requires the maintenance of similar statistics.

In the information retrieval domain, the problem posed by dynamic documents has been looked at from a crawler perspective i.e., to develop better crawling techniques in the face of dynamic web content. [9], [10] present techniques for efficient retrieval of dynamic web-pages so as to capture the maximum number of changes to the web-page. However, our algorithms focus on updating the categories once the updates to the web-pages have been detected. Hence, we can make use of the techniques mentioned in the literature to detect the updates. Another related problem in information retrieval is that of clustering of search results [11], [12] using document clustering techniques. However, as noted in the introduction, focusing merely on the search results will not provide the correct top-K categories. By focusing only on the search results, document clustering techniques are unable to evaluate an important criterion; that of ranking the categories based on the percentage of documents in the category contained in the search result. This criterion is central to the ranking mechanism used in CS* which helps it return the correct top-K categories.

Dynamic data has been widely studied in the area of continuous queries [13]. Answering continuous queries generally involves building models for the dynamic data. Various techniques such as Discrete Time Markov Chains and Black-Scholes Differential Equation [14] have been used to model dynamic data in the literature. The Δ value required by our system can be estimated using these models.

Finally, the problem of answering top-K queries has been widely studied in database literature [15]. However, our problem is closer to the top-K ranking of documents than that of finding the top-K results in a database. In summary, to the best of our knowledge, this is the first attempt to address the problem of identifying the top-K categories for a keyword query where the categories map to one or more data items which are being continuously added to the system. We believe that this is a very important problem and has wide ranging applications beyond those mentioned in this paper.

VIII. Conclusion and Future Work

In this paper we introduced the CS* system which solves the important problem of enabling keyword search on categories which map to multiple data items, where new data items are continuously added to the system. An update-all technique which refreshes all the categories whenever a data item is added will have a prohibitive overhead. Our CS* system consists of two major components: (i) the *Meta-data Refresher* which is responsible for maintaining the statistics associated with the categories and (ii) the *Query Answering Module* which is responsible for using these statistics to answer the user query. The meta-data refresher keeps its attention focused on a dynamically chosen set of important categories and then refreshes them using data items which can provide the maximum benefit. The job of the query answering module is to accurately find the top-K categories related to a user query. We achieved this by proposing a two-level threshold algorithm which was able to find the top-K categories by only examining 20% of the statistics. We experimentally evaluated our system on real world data and showed that our system is able to achieve an accuracy in excess of 90% by using a fraction of the resources required by the update-all technique. Our experiments also showed that the CS* system was highly scalable and was able to easily handle large number of data items. One of the assumptions of the CS* system was that the data items are append only. We are currently exploring techniques to extend our work so as to handle in-place updates and deletions of data items.

References

[1] D. Sifry. State of the blogosphere. [Online]. Available: http://technorati.com/weblog/2006/04/96.html

[2] G. Salton and C. Yang, "On the specification of term values in automatic indexing," *Journal of Documentation*, vol. 29, no. 4, pp. 351–372, 1973.

[3] T. Hagerup and C. Rub, "A guided tour of chernoff bounds," in *Information Processing Letters*, 1990, pp. 305–308.

[4] M. Bhide, V. T. Chakaravarthy, K. Ramamritham, and P. Roy. Keyword search over dynamic categorized information – algorithms. [Online]. Available: http://www.cse.iitb.ac.in/~krithi/papers/CSStarAlgo.pdf

[5] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *Journal of Computer and System Sciences*, 2003.

[6] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *INFOCOM*, 1999.

[7] S. Glassman, "A caching relay for the world wide web," in *WWW*, 1994.

[8] W. Li, "Random texts exhibit zipf's-law-like word frequency distribution," *IEEE Transactions on Information Theory*, 1992.

[9] S. Pandey and C. Olston, "User centric web-crawling," in *WWW*, 2005.

[10] J. Wolf, M. Squillante, P.S.Yu, J. Sethuraman, and L. Ozsen, "Optimal crawling strategies for web search engines," in *WWW*, 2002.

[11] M. Steinbach, G. Karypis, and V. Kumar, "A comparison of document clustering techniques," in *Workshop on Text Mining, KDD*, 2000.

[12] O. Zamir, O. Etzioni, O. Madani, and R. Karp, "Fast and intuitive clustering of web documents," in *KDD*, 1997.

[13] S. Babu and J. Widom, "Continuous queries over data streams," in *ACM SIGMOD Record*, 2001.

[14] M. Bhide, K. Ramamritham, and M. Agrawal, "Efficient execution of continuous incoherency bounded queries over multi-source streaming data," in *ICDCS*, 2007.

[15] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. Ullman, "Computing iceberg queries efficiently," in *VLDB*, 1998.