Monitoring the Dynamic Web to respond to Continuous Queries

Sandeep Pandey
Computer Science and Engineering
Indian Institute of Technology
Powai, Mumbai-400076, India
pandey@cse.iitb.ac.in

Krithi Ramamritham
Computer Science and Engineering
Indian Institute of Technology
Powai, Mumbai-400076, India
krithi@cse.iitb.ac.in

Soumen Chakrabarti
Computer Science and Engineering
Indian Institute of Technology
Powai, Mumbai-400076, India
soumen@cse.iitb.ac.in

ABSTRACT

Continuous queries are queries for which responses given to users must be continuously updated, as the sources of interest get updated. Such queries occur, for instance, during on-line decision making, e.g., traffic flow control, weather monitoring, etc. The problem of keeping the responses current reduces to the problem of deciding how often to visit a source to determine if and how it has been modified so that a user response can be updated accordingly. On the surface, this seems to be similar to the crawling problem since crawlers attempt to keep indexes up-to-date as users pose search queries. We show that this is not the case, both due to the inherent differences between the nature of the two problems as well as the performance metric. We also develop and evaluate a novel multi-phase (**C**ontinuous **A**daptive **M**onitoring) (CAM) solution to the problem of maintaining the currency of query results. Some of the important phases are: The *tracking phase*, in which changes, to an initially identified set of relevant pages, are tracked. From the observed change characteristics of these pages, a probabilistic model of their change behaviour is formulated and weights are assigned to pages to denote their importance for the current queries. During the next phase, the *Resource Allocation* phase, based on these statistics, resources, needed to continuously *monitor* these pages for changes, are allocated. Given these resource allocations, the *scheduling* phase produces an optimal achievable schedule for the monitoring tasks. An experimental evaluation of our approach compared to prior approaches for crawling dynamic web pages shows the effectiveness of our approach to monitoring dynamic changes. For example, by monitoring just 5% of the page changes, CAM is able to return 90% of the changed information to the users. The experiments also produce some interesting observations pertaining to the differences between the two problems of crawling—to build an index—and the problem of change tracking—to respond to continuous queries.

Categories and Subject Descriptors

H.4.m [**Information Systems**]: Information Storage and Retrieval; D.2 [**Mathematics of Computing**]: Probability, Linear Optimization

General Terms

Continuous Queries, Performance, Allocation policies

WWW2003, May 20–24, 2003, Budapest, Hungary.
ACM xxx.

1. INTRODUCTION

The World Wide Web consists of an ever-increasing collection of decentralized web pages that are modified at unspecified times by their owners. Current search engines try to keep up with the dynamics of web by crawling it periodically, in the process building an index that allows better search for pages relevant to a topic or a set of keywords. Clearly, any good crawling technique needs to consider the change behaviour of web pages. But, the algorithms used for crawling and the typical frequency of crawling are insufficient to handle a class of queries known as *Continuous Queries* (for example, see[11]) in which the user expects to be continuously updated as and when new information of relevance to his/her query becomes available. For example, consider a user who wants to monitor a hurricane in progress with the view of knowing how his/her town will be affected by the hurricane. Obviously, a system which responds taking into account the continuous updates to the relevant web pages will serve the users better than another which, say, treats the query as a *discrete query*, i.e., returns an answer only when the query is submitted.

Not surprisingly, the problem of keeping track of the dynamics of the web becomes inherently different for the *continuous query* case compared to the *discrete* query case. We use the term *monitoring* to explicitly account for the differences from the classical crawling problem. A *monitoring task* fetches a web page, much like a crawler does, but with the goal of fetching new information relevant to one or more queries while a *crawl* is not done with any specific user request in mind. The work involved in handling continuous queries is portrayed in Figure 1. For *continuous* queries, since the system should maintain the *currency* of responses to users, the problem translates to one of (a) knowing which pages are relevant, (b) tracking the changes to the pages, to determine the characteristics of changes to these pages, and from these, (c) deciding when to *monitor* the pages for changes, so that responses are current. The last problem is the focus of this paper, and has several subproblems: allocating the resources needed for monitoring the pages, scheduling the actual monitoring tasks, and then monitoring. Specifically, in this paper, we address the problem of distributing a given number of *monitoring tasks* among the pages whose changes need to be tracked so as to respond to a set of *continuous* queries. In Figure 1, the feedback arcs from the monitoring phase to the earlier phases indicate that observations made during the monitoring phase can be used to adjust subsequent decisions.

It could be argued that *discrete* queries posed every so often can be considered to be equivalent to *continuous* queries but the following reasons should help dispel this misconception: First, determining the next time when the *discrete* query should be posed

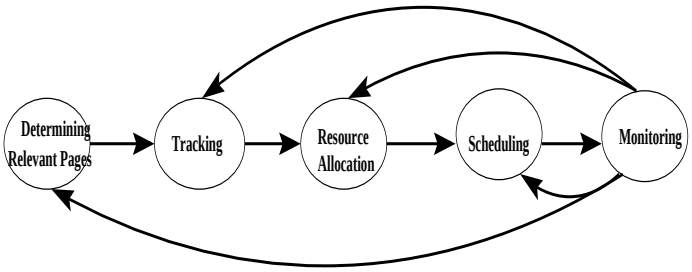

Figure 1: Different phases of our approach

by the user is highly non-trivial. If the time-interval is kept small then it may induce unnecessary load on the system, particularly when the updates are not frequent. If we set the time-interval to be large, it may lead to loss of information if updates are more frequent than expected. Second, *continuous* queries have a non-zero lifetime and so a query system can study a query's characteristics carefully and can answer it more efficiently than in the case where *discrete* queries, which have zero lifetime, are continuously posed. Furthermore, unlike in the case of discrete queries, the time taken to provide the system's first response to a *continuous query* may not be as important as the maintenance of currency during all the responses. This discussion makes it clear that not only the nature of the crawling problem but optimization goals also become different when we move from *discrete* to *continuous* query case. To this end, our optimization metric minimizes the information loss compared to an ideal *monitoring* algorithm which monitors every change of a page.

To our knowledge, no earlier work has focused on the aspect of *monitoring* relevant web pages to respond to a set of *continuous queries*. In this paper, we introduce (**C**ontinuous **A**daptive **M**onitoring) (CAM), a technique to monitor changes. The goal of CAM's resource allocation algorithm is to allocate the monitoring resources across pages so as to minimize the information loss compared to an ideal *monitoring* algorithm which monitors upon every change of a page. This goal also differentiates crawling from *monitoring*. Whereas most of the earlier crawling strategies assume a *Poisson* update process, the CAM approach is more practical and robust as it is not designed with any basic assumptions. Instead, it tracks the changes to pages and evolves the statistics relating to these changes as pages are *monitored*. We show the optimality of CAM's resource allocation algorithm under specific change scenarios and formally prove that, in the *continuous* query case, that Proportional *allocation policy*, in which the pages with high frequency of change are allocated more *monitoring tasks*, works better than *Uniform policy* which allocates equal number of *monitoring tasks* to each page independent of its change frequency. On the surface, this seems to contradict a prior result that *Uniform* allocation of crawling resources produces better results than its *Proportional* counterpart [3]. We justify this surprising behaviour and also give the intuition behind it. This shows that nature of problem of *monitoring* of dynamic web pages for answering *continuous* queries is different from the problem of devising optimal crawling techniques addressed in earlier studies.

The rest of this paper is structured as follows: In Section 2, we define the problem formally and also provide an overview of our Continuous Adaptive Monitoring (*CAM*) approach for supporting continuous queries. Resource allocation and scheduling are the subject of Sections 3 and 4 respectively. Results of performance evaluation are presented in Section 5. Conclusions are related work are found in Section 6.

2. OVERVIEW OF THE CAM APPROACH

Consider a user who is worried about a hurricane in progress and wants to keep abreast of the hurricane-related updates. To achieve this, he poses a continuous *m*-keyword query $q = \{w_1, w_2, \ldots w_m\}$. In this section, we present an overview of the major ingredients of the CAM approach.

Identifying Pages Relevant to a Set of Queries: Based on the keywords specified by a user, we first identify pages relevant to this query. The query is fed to a classical search engine which in turn returns a set of pages relevant to the queries. We find, say, that *the National Hurricane Center*, *National Weather Organization*, and other tropical cyclone sites as well as news sites are relevant. The relevance of a page to a query can be measured by standard IR techniques based on the *Vector-Space* model (see Appendix B for details).

Tracking the Changes to Relevant Pages to Characterize Changes: Once relevant pages have been identified, by visiting each page at frequent intervals during a tracking period, changes to these pages are tracked, update statistics collected, and the relevance of the changes, vis a vis the queries, is assessed. This is used to build a statistical model of the changes to the pages relevant to a set of queries. These statistics include page update instances, page change frequency, and relevance of the changes to the pages for current queries.

Let Q denote the set of all queries submitted in the system and ω_i denote the importance of i^{th} query. These are input to the system. Let P denote the set of web pages relevant to the continuous queries, Q_{p_i} be the set of queries for which i^{th} page is found to be relevant, and $r_{i,j}$ be the estimated relevance of i^{th} page for j^{th} query. It is positive for all queries q$\in Q_{p_i}$ and zero for all $q \in Q$ - Q_{p_i}. These relevance measures are initially calculated during the tracking period (and get updated, as explained later, after every monitoring epoch).

It is clear that not all pages will be equally important for each query in the system. So we rank the pages by assigning a *weight* to each page using its relevance for queries. The *weight* of a page, computed as $\sum_{j \in Q}(\omega_j r_{i,j})$, denotes the value of current version of the page. If the page gets updated before its current version is *monitored*, we assume that we incur a loss of W_i.

Considerations underlying the Monitoring of Changes:

CAM does its monitoring in epochs, each epoch is of duration T time units. The purpose of the resource allocation phase is to decide how to allocate monitoring resources for an epoch and the goal of scheduling is to decide when a monitoring task should execute, given the resource allocation decisions. Monitoring is done by a monitoring task where the task includes fetching a specified page from its source and determining if it has changed and if so applying the changes to return new results for those queries for which that page is relevant.

Let C denote the total number of *monitoring tasks* that can be employed in a single monitoring epoch. C is derived as an aggregation of the resources needed for monitoring, including CPU cycles, communication bandwidth, and memory[1].

λ_i is the estimated number of changes that occur in page i in

[1]For example, the authors of [8] report that with two 533 MHz Alpha processors, 2 GB of RAM, 118 GB of local disk, a 100 Mbit/sec FDDI connection to the Internet, and *Mercator* under srcjava, their crawler crawled at an average download rate of 112 documents/sec and 1,682 KB/sec. Similarly the capabilities of a given infrastructure can be mapped to the number of *monitoring tasks* that it is capable of on average.

T time units. Henceforth we will call it the *change frequency* for page i. Suppose U_i denotes the sequence of time instances $u_{i,1}$, $u_{i,2}$.....u_{i,p_i} at which the tracking phase determines that possible updates occur to page i. We assume $0 \leq u_{i,1} \leq u_{i,2}$.....$u_{i,p_i} \leq T$ and $u_{i,0} = 0$ and $u_{i,p_i} = T$. p_i is the total number of update instances for i^{th} page during T, i.e., cardinality of sequence U_i (p_i=$|U_i|$). Note that a page may not be updated at these time instances and so there is a probability $\rho_{i,j}$ associated with each time instance $u_{i,j}$ that denotes the chances of i^{th} page being updated at the j^{th} instance. The overall goal of the resource allocation and scheduling phases is to monitor in such a way that the monitoring events occur just after updates are expected to take place. The number of missed updates is an indication of the amount of lost information and minimizing this is the goal of the system.

With these considerations in mind, decisions are made about the allocation of a given number of *monitoring tasks* among a set of relevant pages while also deciding *when* these allocated *monitoring tasks* should ideally occur within an epoch. The basic idea is that these monitoring epochs of length T repeat every T units of time and we will make decisions pertaining to the monitoring tasks to be carried out in one monitoring epoch using both new data and the results from the previous epochs.

Resource Allocation Phase:

It should be clear that if we decide to *monitor* at some instance, then it should be at the potential update time instance only because there is no reason to delay it beyond when a update might occur. If number of *monitoring tasks* allocated for a page is equal to the number of update instances, then we can always maintain a fresh version of this page by *monitoring* at all possible update instances. But in practice we will not be able to perform as many monitoring tasks as the number of update instances. So we need to pick a set of update instances at which the page is to be monitored and not at others. Hence with every update time instance, we associate a variable $y_{i,j}$ where

$y_{i,j} = 1$ if *monitoring* of i^{th} page is done at time $u_{i,j}$, 0, otherwise

if we *monitor* the i^{th} page x_i times, then $\sum_{j=0}^{p_i}(y_{i,j}) = x_i$ holds.

The resource allocation phase decides $y_{i,j}$ values, that is, the time instances at which *monitoring* should be done given that the tracking phase has identified the time instances when changes may occur.

Scheduling the Monitoring Tasks: In the *scheduling* phase, we take the $y_{i,j}$ values as inputs and prepare a feasible schedule to meet our optimization measures.

In practice, we have a set of M parallel monitoring processes which continuously perform these monitoring tasks. Now our goal is to map a *monitoring task* to one of these M parallel monitoring processes and determine its time of invocation. While determining any schedule, our aim is to minimize the total delay occurring between the ideal time instances and the actual scheduled time instances. This, the scheduling step involves taking the ideal timings for the monitoring of each page and obtaining an optimal achievable schedule out of it. We map this problem to *flow-shop* scheduling problem [12] with the goal of minimizing the average *completion time*. Next we *monitor* these pages according to the designed schedule and at the end of this *monitoring epoch* update the statistics of these pages on the basis of the observations made in the *preceding epoch*.

In general, based on the results of monitoring tasks of an epoch, scheduling, resource allocations, change statistic computations, and page relevance can all be revisited. These, as mentioned earlier, correspond to the arcs going from the monitoring phase to the earlier phases of Figure 1.

3. RESOURCE ALLOCATION IN CAM

As noted earlier, we need to distinguish between pages on the basis of two metrics. One is the nature of page change behaviour and the other is the importance of a page for one or more queries. Page change behaviour is studied during a *tracking* phase and is characterized by associating a probability of change with every potential update instance. Next we show the way in which pages can be ranked by assigning *weights* to them using relevance measures. These relevance measures are determined for each page for each query during the *tracking* period.

3.1 Goals of the Resource Allocation Phase

CAM aims is to minimize the weighted importance of changes that are not reported to users, that is,

$$min \sum_{i \in P} (W_i E_i)$$

where E_i denotes expected number of lost changes for i^{th} page. We assume that each update instance is independent of others, that is, with each update of a page, information from the preceding update is completely lost. Also, one update is assumed to be independent of another. That is, the number of lost updates is an indication of the amount of lost information. While these may not always be true, they give us a simple way to state the goal to be accomplished. Thus,

$$E_i = \sum_{j \in U_i} \rho_{i,j}(1 - y_{i,j})$$

Resource constraint is given by

$$\sum_{i \in P} \sum_{j \in U_i} y_{i,j} = C,$$

where C denotes the total number of available *monitoring tasks*. Implicitly, we assume that during the monitoring epoch of length T, the relevance of each updated version of page for queries remains the same as estimated during *tracking* period. At the end of a *monitoring epoch*, we update these relevance measures on the basis of the monitored information. So unless T is very large or page updates are very erratic, our assumption is practical.

It is important to point out that it is relatively easy to accommodate the following extensions to the above model. In certain cases, we may have more information about specific changes than the case described above. For example, if we can measure change behaviour of i^{th} page with respect to j^{th} query, then it would be possible to allocate resources even more efficiently. For example, suppose we get to know during the tracking period that a particular news site mainly declares health updates only once at the start of day and in the rest of the time, it remains mainly concerned about political and sports updates, then we can better characterize the change behaviour of this page with respect to queries concerned with sports, medical and political domain.

Suppose $\rho_{i,j,k}$ denotes the probability of change of i^{th} page at j^{th} update instance where this change is relevant for query k. Then E_i, the weighted expected number of lost changes for i^{th} page is,

$$\sum_{k \in Q} \omega_k . r_{i,k} . \sum_{j \in U_i} \rho_{i,j,k}(1 - y_{i,j})$$

If we can extract even more information by measuring not only the probability of change of i^{th} page at update instance $u_{i,j}$ but also

the average importance of change at this time instance, then it can make better resource allocation. For example, suppose we find that a particular research site compiles and announces all its previous day's research updates daily at 10:00 a.m. in the morning and in rest of day, it updates its page only when some new research breakthrough takes place. Then it is clear that visit to this page at 10:00 a.m. is certainly more fruitful than any other visit to this page.

3.2 The Resource Allocation Algorithm

The formulated resources allocation problems are discrete, separable and convex.

1. *Discrete*: because variable $y_{i,j}$ can take only discrete values. Our problem is inherently discrete due to discrete nature of monitoring. Either a monitoring task is allocated to a page or it won't be. There can't be anything between these.

2. *Separable*: because optimizing function could be expressed in terms of $y_{i,j}$ only.

3. *Convex* : due to convex nature of optimizing function.

Discrete, Separable and Convex problems have been well-studied [9]. Formally it can be stated as minimizing $\sum_{i=1}^{G} F_i(x_i)$ with resource constraint: $\sum_{i=1}^{G} x_i = Z$, where $x_i's$ are discrete and $F_i's$ are convex. A *greedy* algorithm exists for the discrete case [6]. There is a faster algorithm also for our problem, due to Galil and Megiddo, which has complexity $O(G(logZ)^2)$. The fastest algorithm is due to Frederickson and Johnson [7] and it has complexity $O(max\{G, Glog(Z/G)\})$. In our case, the output of these algorithms is a set of $y_{i,j}$'s. This set in turn gives us the number of *monitoring tasks* allocated to a page (x_i=$\sum_{j=0}^{P_i}(y_{i,j})$) as well as the ideal time instances, (namely, the js for which $y_{i,j}$ is 1), at which these allocated *monitoring tasks* should be executed.

Given the design of the above resource allocation algorithm, the $y_{i,j}$s that result are optimal, i.e., maximize the value of the returned information, when the updates are quazi deterministic, i.e., occur at specific time instances and the actual occurrence of a change is associated with a probability.

4. SCHEDULING OF MONITORING TASKS

As mentioned earlier, our goal is to schedule the allocated *monitoring tasks* among M parallel monitoring processes with the aim of minimizing the total delay between the ideal time instances and the actual scheduled time instances when a monitoring task must be executed.

Let page P_i be allocated x_i number of *monitoring tasks* in an optimal resource allocation. Also the time instances at which these x_i *monitoring tasks* should be employed are $t_1, t_2, t_3....t_{x_i}$, as identified in the resource allocation phase. Let $fetch_i$ be the average fetching time for the i^{th} page. The scheduling problem can be easily mapped to parallel shop scheduling problem.

In this problem, each *job* has to be processed on exactly one of M identical *machines*. Each *monitoring task* could be regarded as a *job* whereas the monitoring processes are equivalent to *machines*. Suppose there are a total of n such jobs. In scheduling problems, the time at which a job becomes available for processing is called the *release time(rel_j)* and the time for which it needs a machine is called the *processing time*. So in our case, ideal *monitoring* time instances $t_1, t_2, t_3.....t_{x_i}$ would be the *release times* and fetching times of pages correspond to *processing times (p_j)* for jobs. Our goal is to minimize the delay d_i between ideal *monitoring* time instance (rel_i) and actual time instance s_i of scheduling.

In our case all the jobs are equally important as there is no weight assigned with each *monitoring*. So our problem can be formulated in scheduling notation as $R|M|rel_j \geq 0|\sum_j Cm_j$ meaning that R jobs of non-trivial release times are available for scheduling at M machines with goal of minimizing the average completion time. Here Cm_j denotes completion time for job j. Minimizing the average completion time leads to minimization of average delay time because *Total Completion Time*

$$\begin{aligned} &= \sum_{i=1}^{R}(Cm_i) \\ &= \sum_{i=1}^{R}(s_i + p_i) \\ &= \sum_{i=1}^{R}(rel_i + d_i + p_i) \\ &= \sum_{i=1}^{R}(d_i) + \sum_{i=1}^{R}(rel_i) + \sum_{i=1}^{R}(p_i) \end{aligned}$$

As $\sum_{i=1}^{R}(rel_i)$ and $\sum_{i=1}^{R}(p_i)$ are constants, minimizing average completion time is same as minimizing delay time. Note that Cm_i is the same as s_i + p_i because of *non-preemptive* scheduling. Unfortunately even the simpler problem $R|1|rel_j \geq 0|\sum_j Cm_j$ do not have any polynomial time algorithm and has been proved to be *NP-Complete* [10]. So we have to look for *approximation algorithms*. For completion time problem there is an 1.58-approximation algorithm [10] which we used in our experiments.

5. EXPERIMENTAL EVALUATION

In this section, after explaining the setup for the experiments, we describe the results.

5.1 Experimental setup and performance metric

Comparison with Alternative Algorithms: In previous sections, we presented the optimal *resources allocation policy* incorporated in CAM for monitoring changes in pages relevant to *continuous* queries. Here we evaluate our policy by comparing it with some classical policies using a synthetic data set. These policies [3] are:
Uniform in which resources i.e., *(monitoring tasks)* are allocated uniformly across all pages, and
Proportional in which resources are allocated proportional to change-frequencies of pages respectively.
As suggested in [15], it would be fair to compare with the weighted version of these policies than the unweighted ones: In the *Weighted Uniform* scheme, the number of *monitoring tasks(x_i)* allocated to a page depends on the *weights(W_i)* associated with the page but is independent of its change frequency(λ_i): $x_i \propto W_i$. In the *Proportional* scheme, $x_i \propto (\lambda_i W_i)$.
Parameters of the Experiment: As mentioned earlier, each page has an estimated change frequency(λ_i) associated with it which denotes the expected number of changes that occur in a page in T time duration. Also there is a sequence of update instances$(u_{i,j})$ for each page(U_i) which enumerates the time instances at which changes can occur in a page. With each update instance$(u_{i,j})$, there is an associated probability$(\rho_{i,j})$ which denotes the probability with which a change can occur at this instance. In our experiments, to make it simple, we make this sequence of update instances(U) the same for each page. On the first sight, it seems to be in contrast with the model we described in Section 2. There we said that each page will have its own sequence of U_i and it will vary from page to page. But note that fixing of update sequence(U) doesn't create any difference if we choose it to be the *union* of U_i of each page. This is because of the following two reasons: Firstly universal sequence(U) does contain all possible update instance of all the pages, so no update instance of any page is lost. Secondly if a page doesn't have some update instance in its U_i which is in U, then we can always correct it by making the probability associated with

this update instance in U for this page zero. So model for change behaviour of each and every page remains unaffected. Other parameters are decided as below :

1. N_q : number of queries submitted in the system. It is set to 500.

2. N : number of pages found relevant the for the queries submitted. It is also set to 500. This implies that set of relevant pages for queries have common elements in them.

3. C : number of *monitoring tasks* available. It is varied from 1000 to 50000 in our experiments. If T is set to be 15 minutes (for example, the Google-News site is said to use crawlers which visit relevant sites every 15 minutes), then 50000 *monitoring tasks* would require a downloading speed of 56 documents/sec approximately.

4. *Change frequency distribution* : The change frequencies(λ_i's) are chosen according to *Zipf* distribution with parameters N and θ. θ varies from 0 to 2. Such distributions run the spectrum from highly skewed (when θ is 2) to uniform (when θ is 0). Unless otherwise specified, θ is set to 2 in experiments.

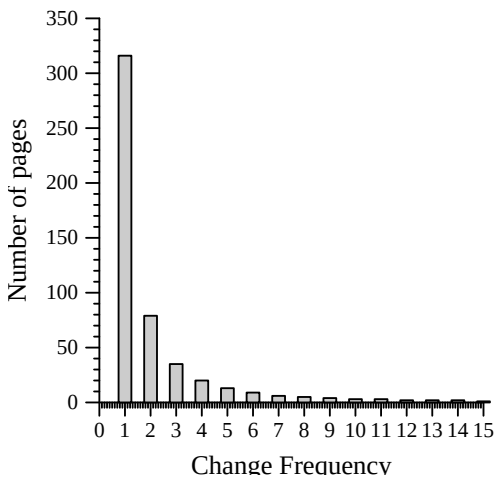

Figure 2: Change frequency distribution

5. *Update probability distribution* : Update instances of U are assumed to be uniformly distributed throughout the duration T. In our experiments, we have divided T in 480 update instances. Probabilities ($\rho_{i,j}$) associated with these update instances($u_{i,j}$) are varied between 0 and 0.3 and follow a *Zipf* distribution. Henceforth we will refer to this distribution as *update probability distribution*. *Zipf* is chosen because of the fact that most of the web pages have time durations when they are updated with greater probabilities in comparison to the rest of the time durations. News sites can have multiple hot time durations and that can be modeled by generating many "humps" in their *update probability distribution* with probability varying in the vicinity of every hump in *Zipf* fashion. Note that the probabilities($\rho_{i,j}$) for all update instances of a page should sum up to expected change-frequency(λ_i) of that page. Also note that we vary the *zipf* parameter of this *update probability distribution* from 0 to 2 in our experiments and so we get a corresponding *update probability distribution* for a page in T varying from a uniform to a highly skewed distribution. This makes our experiments free from a priori assumptions about page change behaviour and helps in evaluating our policies for real scenarios.

6. *Weight of queries* : All queries are assigned the same *importance measure*(ω_i). It means that there is no distinction made among queries and they are defined to have equal importance.

7. *Page Weight Distribution* : Recent studies [14] show that popularity of pages vary in *zipf* fashion as shown in Fig 3. Drawing an analogy, we choose $r_{i,j}$, the relevance of page

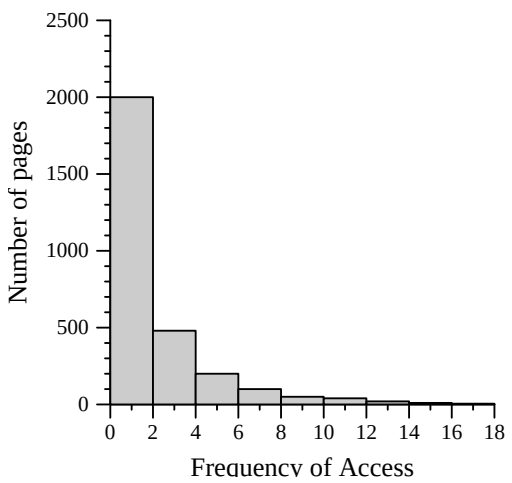

Figure 3: Observed popularity distribution

j for a query i, from a *zipf* distribution. Also the more dynamic a page, the more more popular it is too, as shown in [16]. So we make the the more dynamic more have higher relevance in our experiments to a page in a *biased* random manner. The summation of relevance measures of a page for all the queries gives us the weight(W_i) for this page as discussed in Section 3. The distribution according to which W_i varies is referred to as *page weight distribution*.

8. *Monitoring-change ratio*: denotes the ratio of the total number of *monitoring tasks* to, ($\sum_{i \epsilon P} \lambda_i$), i.e., the number of actual changes expected in time T.

Performance Metric: Returned Information ratio: Proportion of the changed *information* returned by a given number of monitoring tasks is called as *Returned Information Ratio*. From section 3,

$$\frac{\sum_{i \in P} W_i . \sum_{j \in U} (\rho_{i,j} . y_{i,j})}{\sum_{i \in P} (W_i . \lambda_i)}$$

Note that the maximum possible value of *returned information ratio* is 1 and it is attained when all those $y_{i,j}s$ are made 1 for which corresponding $\rho_{i,j}s$ are non-zero. This is the performance metric on the basis of which we compare various allocation policies in our experiments.

5.2 Comparison of Resource Allocation Policies

In this experiment, we evaluate the aforementioned resource allocation polices and also observe the effects of *update probability distribution* and *page weight distribution* on their performance.

5.2.1 *Uniform page weights and update probabilities*

We make both these distributions uniform and set the *Zipf* parameter of *change frequency distribution* to 2 as shown in Fig. 2. Uniform *page weight distribution* means that all pages have equal importance while uniform *update probability distribution* leads to equal probability of change to a page at any update instance in T. Fig. 4 shows the performance of different resource allocation policies. There are two important observations.

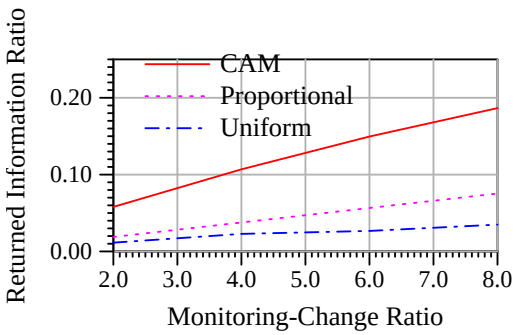

Figure 4: Performance under uniform page weight and update probability distribution

1. *Proportional* policy performs better than its *Uniform* counterpart. This is very surprising as earlier studies showed the reverse to be true [3] [15]. The reason becomes clear when we delve into the nature of the crawling vs. the monitoring problem. In our case, we answer *continuous* queries and our aim is to detect as many changes as possible. So when all other parameters (page-weight and update probability distribution) are uniform, one would certainly expect more benefits by *monitoring* those pages which have high change frequency (λ_i) because these pages have considerable chances of changing. This is what *Proportional* policy does and so it performs better than *Uniform* policy. Earlier studies solved the problem for answering discrete queries and aimed to maximize *freshness* of page which is found to be of a *convex* nature. So the performance of *Uniform* becomes better than *Proportional* in their case. We offer a formal proof of why *Uniform* does not work as well as *Proportional* for *continuous* queries in an Appendix.

2. Optimal policy also allocates more *monitoring tasks* to more dynamic pages but it does it even more aggressively than *Proportional*. Fig. 5 shows that our CAM approach allocates all its *monitoring tasks* to only a few pages (for clarity, for the sake of this graph, 50 pages of consecutive page indices have been grouped into a bin) and delivers most of the *information* to queries from these pages. Again the pages *monitored* are those which have high probability of actually changing. *Proportional* too does this but it allocates *monitoring tasks* in a proportional manner only while CAM does it in a more biased way and so it gets even better performance. It is evident from the graph that optimal policy performs 300% better than *Proportional* policy and around 600% better than *Uniform* policy!

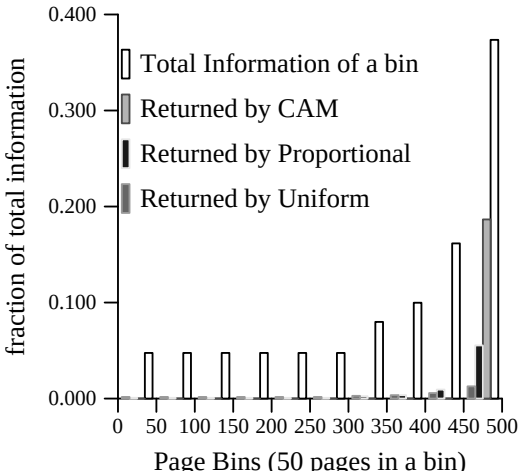

Figure 5: Characteristics of Resource Allocation Policies

If we decrease the skewness, i.e., the *zipf* parameter of the *change frequency distribution* the policies start coming closer and in the extreme case, they all become the same when frequencies are made to be distributed in a uniform manner (*Zipf* parameter set to 0).

5.2.2 *Skewed page update probabilities*

We skew the *update probability distribution* with *zipf* parameter set to 1. So pages are still of equal importance but for each page, the update instances are no more equi-probable, in changing. Fig. 6 shows the performance. Again, CAM performs best leaving other allocation policies far behind. It is 12 times better than *Uniform* policy. But in this case, the pages which are *monitored* by CAM turn out to be quite diversified as pages with even lesser change frequency have some update instances with a good chance of actually changing as shown in Fig. 7.

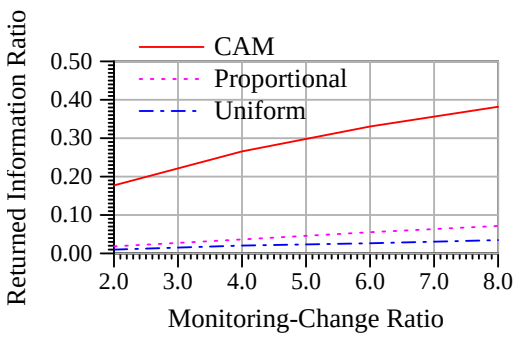

Figure 6: Under skewed update probability and uniform page weight distribution

5.2.3 *When page weights are skewed*

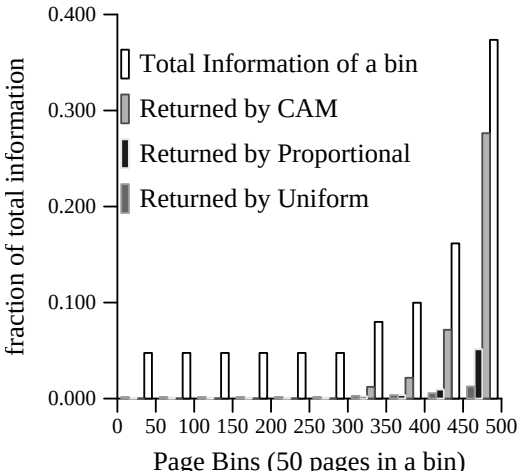

Figure 7: Characteristics of resource allocation policies

If we make *page-weight* distribution skewed while keeping *update probability distribution* uniform, we find that optimal again performs far better than others as shown in Fig. 8. Also, now it allocates *monitoring tasks* to those pages which have high importance and a higher probability of getting changed.

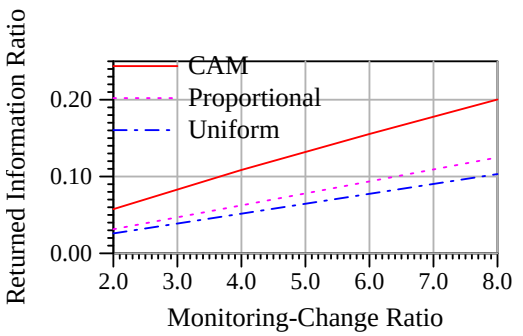

Figure 8: Performance under skewed page weight and uniform update probability distribution

In summary, CAM's resource allocation approach performs better than the previously proposed *Uniform* and *Proportional* approaches across a wide spectrum of distributions. In particular, the more skewed the distributions, the more pronounced the performance improvement.

5.3 Effect of varying the skewness of the update probability distribution

Fig. 9 compares performance of different resource allocation policies when *monitoring-change ratio* is kept at 9 and *page weight distribution* is uniform. It is clear from the curves that for this data set, CAM's resource allocation policy always performs better than the other resource allocation policies. To emphasize the difference, we varied the *update probability distribution* keeping other parameters same as before. As is evident from Fig. 9, CAM's resource allocation policy starts performing even much better than other policies as *update distribution* is made more and more skewed. It exhibits a 5-fold improvement over *Uniform* resource allocation policy at zero *zipf* parameter but when *zipf* parameter is set to 1.5, its performance sees a 10-fold improvement.

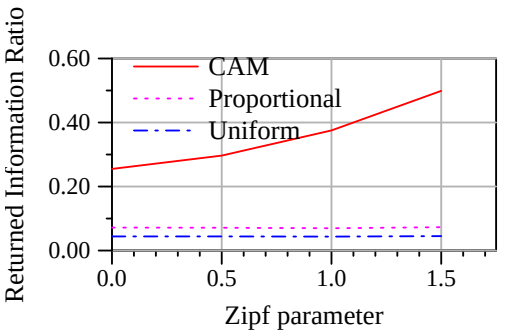

Figure 9: Performance under varying skewness of update probability distribution

This is because of the fact that in CAM's resource allocation policy, *monitoring* is done at those update instances which have a high probability of returning relevant information and so, as *update probability distribution* is made more and more skewed, it responds to the skewness of data by selecting the most beneficial instances for *monitoring* and performs even better than before. But this is not the case with *Uniform* and *Proportional* policies as they do not take into account the granularity of update instances and decide to *monitor* based on weight and change frequency. (Note that when *zipf* parameter is set to zero, it does not mean that update probabilities become uniformly distributed, instead of this it means that all update probability values occur equal number of times.)

5.4 Identifying the Parameters to get even Better Results for Continuous Queries

In the previous experiment, we observed that even when we have 9 times more *monitoring tasks* available than expected number of changes in T, the loss of information remains significant. This is

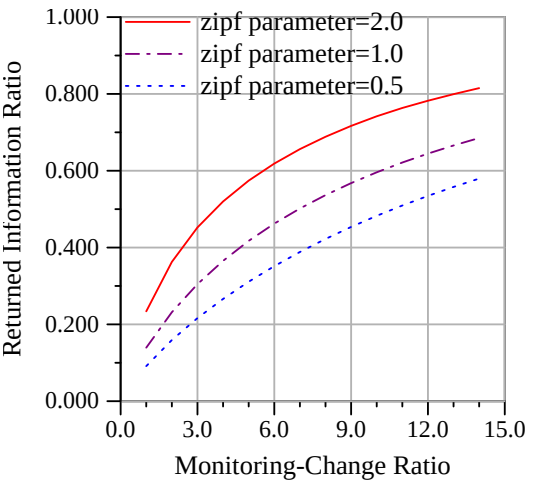

Figure 10: Performance with varying skewness of update probability distribution

because of the distributed and uncertain nature of page change behaviour which make the number of *monitoring tasks* required for good performance very large (Section 5.2.1). In the ideal case, we will require continuous monitoring of web pages and so even a large number of *monitoring tasks* (until they become comparable the to number of update instances) will not be of much help.

Fig. 10 shows how performance varies with the *update probability distribution* of page change behaviour. It is also evident that pages with almost uniform *update probability distribution* will require more monitoring than the skewed case. We find that *page*

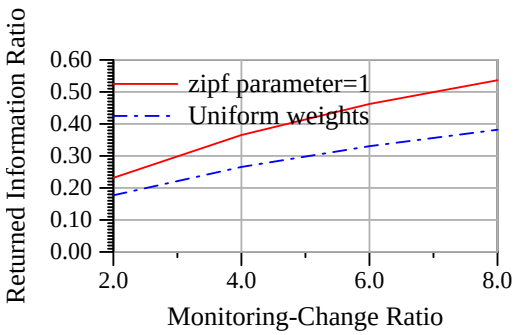

Figure 11: Performance with different skewness of page weight distribution

weight distribution also affects the performance in a significant way. This is intuitive: if we can somehow figure out during *tracking phase* that a particular set of pages is serving a major part of reportings to users for answering query, then we can improve our performance by assigning them a major share of *monitoring tasks*. Fig. 11 shows the effect of *page-weight* distribution on the performance of allocation techniques. In general, *Continuous* queries can be responded to even more efficiently by extracting meta-information about the change behaviour of web pages.

5.5 Effect of Monitor-change Ratio on Continuous Queries

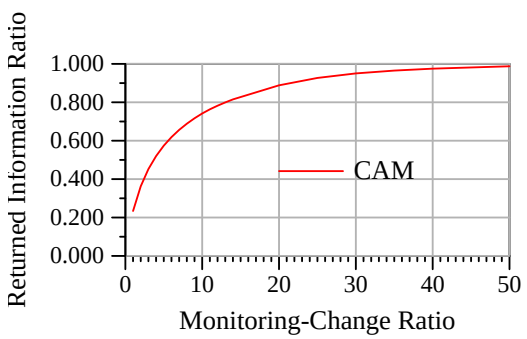

Figure 12: Performance of optimal policy

Here we evaluate the practical application of our proposed scheme. As evident from Fig. 12, 90% of the Information is returned in our technique when *monitor-change ratio* is 20. Without using CAM, retrieving 90% of the information would require *monitoring* of at least 432 monitoring instances while CAM needs only 20 *monitoring tasks* (5% of the maximum needed monitoring tasks).

5.6 Reallocating Resources

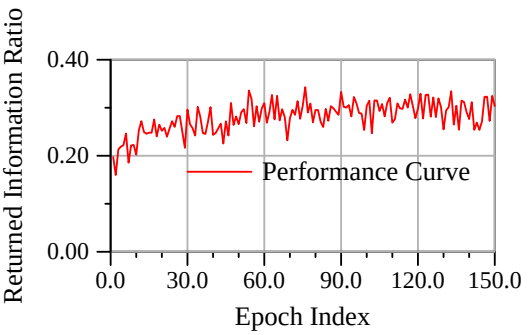

Figure 13: Effect of Resource Reallocation after every epoch

As we said while describing our CAM technique that after every epoch of length T, we update page change behaviour and accordingly modify resource allocations for next epoch. But this may become very expensive especially when T is small. So, we next study the effect of the the *resource allocation delay* to determine how often it might be beneficial to reallocate the resources.

We start with *page update probability distribution*'s *zipf* parameter being set to 1. Then we generate an actual event based on this *update probability distribution* by tossing a biased coin at every update instance and declaring a change at an instance if it falls head. Before the next epoch, we modify the *update probability distribu-*

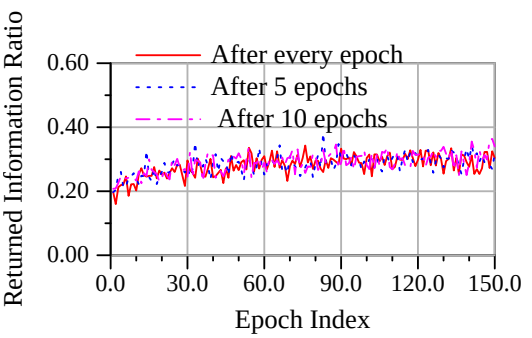

Figure 14: Effect of Varying the Resource Reallocation Delay

tion based on the recent epoch by modifying update probabilities($\rho_{i,j}$) of those update instances which get *monitored* in the epoch. We do this modification simply by estimating the average rate of occurrence of updates: if a page was updated five times in the last 10 monitoring instances, 0.5 is assigned as the probability of expecting a change at the next update instance. Then we reallocate resources accommodating this new *update probability distribution*. As Fig. 13 shows that performance of such a reallocation policy does increase in the initial epochs and then becomes steady. This is what one would expect because after a large number of epochs, the *update probability distribution* itself becomes steady. Also we

plotted 2 more graphs as shown in Fig. 14 to study the effect of delayed resource allocation: Here the statistics are updated after a set of epochs. We find that resource allocation is not required to be done after every epoch and can be delayed without incurring any significant loss provided the *tracking phase* was used to capture the initial page change behaviour. This study shows that it is possible to adapt to the change behaviour using the CAM approach and derive additional benefits in terms of the quality of information returned.

5.7 Performance of the Scheduling algorithm

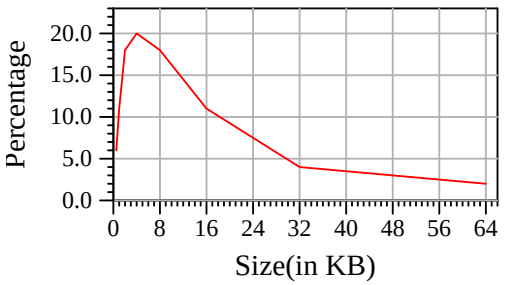

Figure 15: Size of web documents

In this experiment, we test our scheduling algorithm and show its performance. *Change frequency distribution*'s zipf parameter is set to 2 and *update probability distribution*'s zipf parameter to 1. Sizes of the documents are generated as shown in Fig. 15 as per [8]. Also, the more popular pages' sizes are set to be smaller [5]. We define average *monitoring capacity* as available bandwidth divided by average size of documents. As shown in Fig. 16, our *scheduling* algorithm performs very well and is almost *lossless* when the number of *monitoring tasks* is less than the average *monitoring capacity*. Even when the number of *monitoring tasks* required to be scheduled exceeds the average *monitoring capacity*, the loss of information incurred in the *scheduling* phase remains quite negligible in comparison of *resource allocation* phase. The two kinds of losses incurred are:

1. As the number of *monitoring tasks* to be scheduled becomes more than the average *monitoring capacity*, some *monitoring tasks* remain undone and so some loss of information occurs.

2. As number of monitoring tasks increases, the chances of exceeding the *monitoring capacity* at an instance also becomes high. So these *monitoring tasks* get delayed, leading to loss of information.

The experimental results lead to the following observations:

- CAM produces query results with higher coherency than either *Proportional* or *Uniform* can. Often the amount of returned information with CAM's approach is 5-10 times that returned by Uniform or Proportional. The more skewed the page update and page weight distribution, the better the improvement. Increased availability of monitoring resources (as indicated by the monitoring-change ration) also leads to a larger performance improvement with CAM than with the others.

- By deploying a relatively very small number of monitoring tasks, e.g., 5% of the total number of update instances, CAM's resource allocation and scheduling algorithms are able to return a very large proportion, in the above case, 90%, of the changed information.

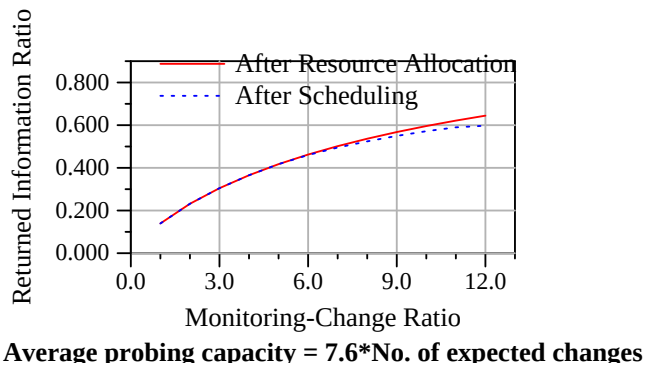

Figure 16: Allocation vs. Scheduling Decisions

- It is possible to improve performance further by updating the page change statistics using of the changes monitored during each epoch. We showed that in fact, it is possible to achieve considerable performance improvement even if such adaptation is not done after every epoch, but once after several epochs suffices.

6. CONCLUSIONS AND RELATED WORK

In this paper, we examined the problem of keeping responses to continuous queries current by focusing on the problem of dynamically monitoring the sources of information relevant to the queries. From the change characteristics of these pages—observed in a tracking phase, a probabilistic model of their change behaviour is formulated and weights are assigned to pages to denote their importance for the current queries. During the Resource Allocation phase, based on these statistics, resources, needed to continuously monitor these pages for changes, are allocated. Given these resource allocations, the scheduling phase produces an optimal achievable schedule for monitoring. We also presented experimental evidence for the effectiveness of our approach which offers several-fold improvement in the returned information, compared to the classical *Uniform* and *Proportional* techniques. In general, CAM performance improves even more under skewed page update and page weight distributions. We also showed that these techniques do not work well for answering continuous queries because of the specific nature of continuous queries. We formally proved that *Proportional* allocation works better than *Uniform* policy for the continuous query case. CAM's resource allocation algorithm is deisgned to be optimal: It maximizes the value of the returned information, when the updates are quazi deterministic, i.e., when updates occur at specific time instances and the actual occurrence of a change is associated with a probability. Similar resource allocation techniques can be developed to be optimal for other types of update behaviours.

There have been several studies of web crawling in an attempt of capturing web dynamics. The earliest study to our knowledge is by Brewington and Cybenko. In [1], they not only studied the dynamics of web but also raise some very interesting issues for developing better crawling techniques. They showed that page change behaviour varies significantly from page to page and so crawling them equal number of times is a fallacious technique. [3] and

[2] address a number of issues relating to the design of effective crawlers. In [4][15], authors approached the problem formally and devised an optimal crawling technique. (Some aspects of our formal are adopted from [15] and modified to suit our problem definition.) A common assumption made in most of these studies is that page changes are a *Poisson* or *memoryless* process. In fact it has shown to hold true for a large set of pages but it is also found in [1] that most of web pages are modified during US working hours, *i.e.*, 5 a.m. to 5 p.m. In our case, we go beyond these assumptions and present an optimal monitoring technique for answering continuous queries independent of any assumption about page change behaviour. Instead, we collect and build page change statistics during a *tracking* period and only on the basis of this collected information, we do *resource allocation*. Then we keep on updating this information after every T time units based on the result of the monitoring done. This makes our solution robust and adaptable in any web scenario.

It is important to mention the push-based alternative to answering *continuous queries*: information is *pushed* from web sources instead of users *pulling* it as is assumed in our scheme [13]. Here users register their queries with the sources and when the sources update the relevant pages they themselves propagate their changes to the users. This, of course, is applicable only to push-based Web sites, and even in that case, the onus of consolidating and aggregating the information returned from the sources is left to the end application.

7. REFERENCES

[1] B. E. Brewington and G. Cybenko. How dynamic is the Web? *Computer Networks (Amsterdam, Netherlands: 1999)*, 33(1–6):257–276, 2000.

[2] J. Cho and H. Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *Proceedings of the Twenty-sixth International Conference on Very Large Databases*, 2000.

[3] J. Cho and H. García-Molina. Synchronizing a database to improve freshness. *In Proceedings of 2000 ACM International Conference on Management of Data(SIGMOD)*, 30(1–7):161–172, 2000.

[4] E. Coffman, J. Z. Liu, and R. R. Weber. Optimal robot scheduling for web search engines. *Journal of Scheduling*, 1998.

[5] C. Cunha, A. Bestavros, and M. Crovella. Characteristics of World Wide Web Client-based Traces. Technical Report BUCS-TR-1995-010, Boston University, CS Dept, Boston, MA 02215, April 1995.

[6] B. Fox. Discrete optimization via marginal analysis. *Management Science*, 13(3):211–216, 1966.

[7] G. N. Frederickson and D. B. Johnson. The complexity of selection and ranking in x + y and matrices with sorted columns. *Journal of Computer and System Sciences*, 24:197–208, 1982.

[8] A. Heydon and M. Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, 1999.

[9] T. Ibaraki and N. Katoh. Resource allocation problems: Algorithmic approaches. *MIT Press, Cambridge, MA*, 1988.

[10] J.K.Lenstra, A. Kan, and P.Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[11] L. Liu, C. Pu, and W. Tang. Continual queries for internet scale event-driven information delivery. *Knowledge and Data Engineering*, 11(4):610–628, 1999.

[12] M.R.Garey, D.S.Johnson, and R.Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics Operation Research*, 1:117–129, 1976.

[13] C. Olston, B. T. Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *SIGMOD Conference*, 2001.

[14] J. Pitkow and P. Pirolli. Life, death, and lawfulness on the electronic frontier. In *Proceedings of the Conference on Human Factors in Computing Systems CHI'97*, 1997.

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

[16] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. R. Karlin, and H. M. Levy. On the scale and performance of cooperative web proxy caching. In *Symposium on Operating Systems Principles*, pages 16–31, 1999.

Appendix A: Proof that Proportional Policy is better than Uniform for Continuous Queries

Assumption : Update distribution is uniform.

We compare weighted *Uniform* and weighted *Proportional* policies.

Number of crawls allocated to i^{th} page in proportional policy is $\frac{C.W_i.\lambda_i}{\sum_i (W_i.\lambda_i)}$

where W_i and λ_i are weight and change frequency of i^{th} page respectively.

So Information gained for this page is equal to $\frac{C.{W_i}^2.\lambda_i.\rho_i}{\sum_i (W_i.\lambda_i)}$

where ρ_i is the update probability for i^{th} page at any update instance.

Information gained in case of *Uniform* Allocation for the same page is equal to $\frac{C.{W_i}^2.\rho_i}{\sum_i W_i}$

So ratio of performance of Proportion to *Uniform* policy over all pages becomes $\frac{\sum_i \lambda_i . \sum_i W_i}{\sum_i (W_i.\lambda_i)}$

As we know $\sum_i a_i . \sum_i b_i \geq \sum_i (a_i.b_i)$ for non-negative a_i's and b_i's, above ratio is always greater than 1.

This proves that *Proportional* policy always performs better than *Uniform* policy no matter how page weights and change frequencies are distributed.

Appendix B: Determining Relevant Pages Given a n-word document $a = \{w_1, w_2, \ldots w_n\}$ and a set of n recognized words, one can represent q and a each as a vector of word frequencies $\vec{q}$ and $\vec{a}$. A common measure of similarity between two word frequency vectors $\vec{a}$ and $\vec{q}$ weighted by inverse document frequency (idf) is the cosine distance between them:

$$score(\mathbf{q}, \mathbf{a}) = \frac{\sum_{w \in q,a} \lambda_w^2 \cdot f_q(w) \cdot f_a(w)}{\sqrt{\sum_{w \in q} (\lambda_w f_q(w))^2 \cdot \sum_{w \in a} (\lambda_w f_a(w))^2}},$$

where $f_d(w)$ is the number of times word w appears in the document d and λ_w is the inverse document frequency of the word w defined as:

$$\lambda_w = \log \left(\frac{|\mathcal{D}|}{|\{d \in \mathcal{D} : f_d(w) > 0\}|} \right)$$

where $\mathcal{D}$ is the document set in consideration.