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, in order to update earlier responses accordingly. On the surface, this seems to be similar to the crawling problem since crawlers attempt to keep indexes up-to-date as pages change and 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 propose, 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 behavior 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 CAM for 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**]: Linear Optimization

General Terms

Continuous Queries, Performance, Allocation policies

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 behavior of web pages. However, the algorithms used for maintaining a crawled collection, and the typical frequency of (re-)crawling are insufficient to handle a class of queries known as *Continuous Queries* (for example, see[12]) 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 freshness or *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 by the user is highly non-trivial. If the time-interval is kept small then it may induce unnecessary load on the system, particularly

WWW2003, May 20–24, 2003, Budapest, Hungary.
ACM 1-588113-680-3/03/0005.

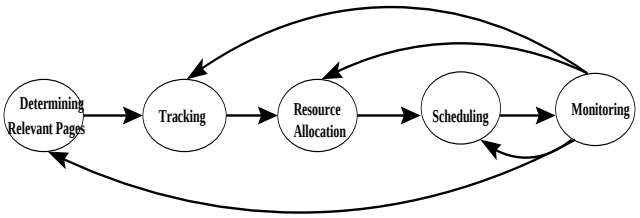


Figure 1: Different phases of our approach

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, in contrast to discrete queries which have zero lifetimes, continuous queries have a non-zero lifetime, enabling a query system to study a query's characteristics carefully and thereby answer it more efficiently. 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.

The above 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 queries. To this end, our optimization metric minimizes the information loss, compared to an *ideal* monitoring algorithm which can monitor every change of a page.

Our contributions. 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 limited monitoring resources across pages so as to minimize the information loss compared to an *ideal* monitoring algorithm which can monitor every change of a page. This goal also differentiates crawling from monitoring. Whereas most of the earlier crawling strategies assume a *Poisson* update process, CAM makes no such assumptions; it tracks the changes to pages dynamically and evolves the statistics relating to these changes, making CAM more practical and robust.

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**, in which the pages with high frequency of change are allocated more monitoring tasks, works better than **uniform allocation**, which allocates an equal number of monitoring tasks to each page, independent of its change frequency. On the surface, this seems to contradict a prior result [3] that the uniform allocation of crawling resources produces better results than its proportional counterpart. We provide an intuitive explanation for this apparently surprising behavior. This also emphasizes why CAM is distinct from earlier work on crawl maintenance for answering discrete queries.

Map. 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 CAM

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. Each query is submitted to a Web search engine which returns a set of pages relevant to the queries. The returned URLs determine the domain of pages which we will monitor. Continuing our earlier example, we may 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 and characterizing relevant page changes. By visiting each relevant page (identified via a search engine) 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 data 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 continually, as explained later).

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.

Change monitoring considerations. CAM does its monitoring in **epochs**. Each epoch has duration T. The purpose of *resource allocation* is to decide how to allocate a limited number of monitoring tasks for an epoch and the goal of *scheduling* is to decide *when* each allocated task should execute, given the resource allocation decisions. Monitoring is done by a monitoring task which fetches a specified page from its source, determines if it has changed and, if so, propagates the effects of these changes to responses of affected queries.

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. E.g., Heydon and Najork [9] 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, their crawler (Mercator) downloaded 112 documents/sec and 1,682 KB/sec (on an average).

λ_i is the estimated number of changes that occur in page i in T time units. Henceforth we will call it the *change frequency* for page i. Suppose U_i denotes the sequence of time instances $u_{i,1}, u_{i,2}, \ldots, u_{i,p_i}$ at which the tracking phase determines that possible updates occur to page i. We assume $0 \leq u_{i,1} \leq u_{i,2} \cdots \leq u_{i,p_i} \leq T$, $u_{i,0} = 0$, 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 the best instant when these allocated tasks should 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 instant, then it should be at the potential update time instant only because there is no reason to delay it beyond when a update might occur. If the number of monitoring tasks allocated for a page is equal to the number of update instants, then we can always maintain a fresh version of this page by monitoring at all possible update instants. But in practice, we will not be able to perform as many monitoring tasks as the number of update instants. So we need to pick a set of update instants at which the page is to be monitored and not at others. Hence with every update time instant, we associate a variable $y_{i,j}$ where $y_{i,j} = 1$ if the i^{th} page is monitored at time $u_{i,j}$, and 0 otherwise. If we monitor the i^{th} page x_i times, then $\sum_{j=0}^{p_i} y_{i,j} = x_i$. The resource allocation phase decides $y_{i.j}$ values, that is, the time instants at which monitoring should be done given that the tracking phase has identified the time instants 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 instants and the actual scheduled time instants. 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 [15] with the goal of minimizing the average *completion time*. Next we monitor these pages according to the designed schedule and, at the end of the current monitoring epoch, update the statistics of these pages on the basis of the observations made during the current 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 behavior and the other is the importance of a page for one or more queries. Page change behavior is studied during a *tracking* phase and is characterized by associating a probability of change with every potential update instant. 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: $\sum_{i \in P} W_i E_i$. Here E_i denotes expected number of lost changes for i^{th} page.

We make two assumptions about updates. First, inter-update intervals are independent of each other. Second, an update results in complete loss of information from previous updates. 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,

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

The constraint on resources is given by

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

where C denotes the maximum available number of monitoring tasks (which we assume are all used up). 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 the 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 acceptable.

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 behavior 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 behavior 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 instant where this change is relevant for query k. Then E_i, the weighted expected number of lost changes for i^{th} page is,

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

If we can extract even more information about changes to the ith page (by measuring, say, not only the probability of change at time $u_{i,j}$, but also the average importance of change at this time) 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, but throughout the rest of the day, it updates its page only if/when some new research breakthrough takes place. Then it is clear that visit to this page right after 10:00 a.m. is more likely to be fruitful than any other visit to this page.

3.2 The Resource Allocation Algorithm

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

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.

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

Convex owing to the convex nature of the objective function.

Discrete, separable and convex problems have been studied intensively [10]. 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_is are discrete and F_is are convex. A *greedy* algorithm exists for the discrete case [7]. There is a faster algorithm also for our problem, due to Galil and Megiddo, which has complexity $O(G(\log Z)^2)$. The fastest algorithm is due to Frederickson and Johnson [8] of complexity $O(\max\{G, G\log(Z/G)\})$. 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 instants, (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 quasi-deterministic, i.e., occur at specific time instants and the actual occurrence of a change is associated with a probability.

4. SCHEDULING 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 instants and the actual scheduled time instants when a monitoring task must be executed.

Let page i be allocated x_i number of monitoring tasks in an optimal resource allocation. Also, suppose the time instants at which these x_i monitoring tasks should be employed are $t_1, t_2, t_3, \ldots, t_{x_i}$, 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 instants $t_1, t_2, t_3, \ldots, t_{x_i}$ 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 instant (rel_i) and actual time instant 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 $R|M|\text{rel}_j \geq 0|\sum_j \text{Cm}_j$ (Cm_j denotes the completion time for job 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. Minimizing the average completion time leads to minimization of average delay time, because the total completion time is

$$\sum_{i=1}^{R} \text{Cm}_i = \sum_{i=1}^{R} (s_i + p_i)$$

$$= \sum_{i=1}^{R} (\text{rel}_i + d_i + p_i) = \sum_{i=1}^{R} d_i + \sum_{i=1}^{R} \text{rel}_i + \sum_{i=1}^{R} p_i.$$

As $\sum_{i=1}^{R} \text{rel}_i$ 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 $R|1|rel_j \geq 0|\sum_j Cm_j$ has been proved to be *NP-Complete* [11]. So we have to look for *approximation algorithms*. For completion time problem there is an 1.58-approximation algorithm [11] which we used in our experiments as a heuristic for optimizing average delay time (an approximation for average completion time does not necessarily translate to the same approximation for average delay time; the latter is harder to approximate).

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 *resource 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 (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 [18], 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 time T. Also, there is a sequence U_i of update instants ($u_{i,j}$) for each page i which enumerates the time instants at which changes can occur in it. With each update instant ($u_{i,j}$), there is an associated probability ($\rho_{i,j}$) which denotes the probability with which a change can occur at this instant. To simplify our experiments, we make the sequence of update instants (U) the same for each page. At first sight, it seems to be in contrast with the model we described in Section 2. There we said that each page i may have its own distinct sequence U_i. But note that there is no loss of generality in fixing all update sequences to a common U, if U is chosen as the *union* of U_is of all pages, for the following two reasons. First, the universal sequence U contains all possible update instants of all the pages, so no update instant of any page is lost. Second, we can mask off every update instant in U which does not occur in U_i by associating with it a update probability of zero.

Other experimental parameters are set as described below.

1. N_q : The number of queries submitted in the system, set to 500.

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

3. C: The 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 a *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.

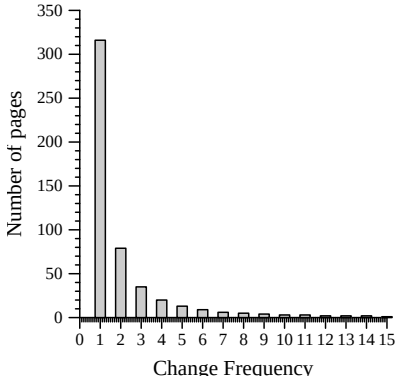


Figure 2: Change frequency distribution

5. *Update probability distribution*: In our experiments, we have 480 update instants in U at uniform gaps throughout the duration T. For each experiment, we need to associate a probability $\rho_{i,j}$ with each update instant $u_{i,j}$. An actual update (or absence of an update) for page i at the jth update instant is materialized by tossing a coin with bias $\rho_{i,j}$. For our experiments, the probability values $\rho_{i,j}$ are themselves generated from a Zipf distribution, clipped between 0 and 0.3. Using a Zipf distribution over $\rho_{i,j}$ biases them toward lower values and ensures that relatively few potential update instants are likely to be instantiated into an actual update for any page. Henceforth we will refer to this distribution as *update probability distribution*. News sites are often updated in a *quasi-deterministic* manner: at specific intervals, changes occur with some probability. This means the inter-update time distribution has multiple modes, which 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 instants of a page should sum up to expected change-frequency (λ_i) of that page. Also note that we vary the Zipf skewness parameter of the update probability distribution from 0 to 2 in our experiments and so we get a corresponding update probability distribution for a page varying from a uniform to a highly skewed distribution. All this makes our experiments free from a priori assumptions about page change behavior and helps in evaluating our policies for real scenarios.

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

7. *Page weight distribution*: Recent studies [17] show that popularity of pages vary in a Zipfian fashion as shown in Fig 3. Drawing an analogy, we choose $r_{i,j}$, the relevance of page j for a query i, from another Zipf distribution. There is some evidence [19] that the popularity of a page is positively related to its rate of updates. Therefore, in our experiments, we make the more dynamic pages have higher relevance. 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*.

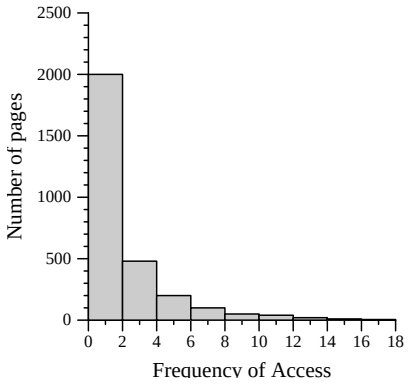


Figure 3: Observed popularity distribution

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

Returned information ratio. : The total expected information change over all pages is $\sum_{i\in P} W_i\lambda_i$, and if the allocation algorithm selects suitable (0/1) values of $y_{i,j}$, the expected fraction of information change made visible to the user(s) is $\sum_{i\in P} W_i.\sum_{j\in U}\rho_{i,j}.y_{i,j}$. We define the ratio of the latter to the former as the **returned information ratio** (RIR), our main performance metric.

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

Note that the maximum possible value of RIR is 1 and it is attained by setting $y_{i,j}$ to 1 for every i, j having $\rho_{i,j} > 0$. 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 Figure 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 instant in T.

Figure 4 shows the performance of different resource allocation policies. There are two important observations.

1. The proportional policy performs better than its uniform counterpart. This is very surprising as earlier studies showed the reverse to be true [3, 18]. 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 the proportional policy does and so it performs better than the uniform policy. Earlier studies solved the (convex optimization) problem of answering discrete queries and aimed to maximize *freshness* of pages. In that problem domain, the performance of the uniform policy was better than the proportional policy. We offer a formal proof of why uniform does not work as well as proportional for continuous queries in Appendix A.

2. The optimal policy also allocates more monitoring tasks to more dynamic pages but it does it even more aggressively than the proportional policy. Figure 5 shows that CAM 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 change information from these pages to CQs. The pages monitored are those which have high probability of actually changing. Proportional does this too, but, as its name indicates, it can only allocate tasks in a proportional manner, while CAM does it in a more biased fashion and get even better performance. It is evident from the graph that optimal policy performs 300% better than the proportional policy and around 600% better than the uniform policy.

If we decrease the skewness (the Zipf parameter of the change frequency distribution), the policies start approaching each other. 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 all pages are still of equal importance, but for each page, the probability that an update instant instantiates to an actual update is no longer the same for all update instants.

Figure 6 shows the performance in this setting. Again, CAM performs best leaving other allocation policies far behind. It is 12 times better than the uniform policy. But in this case, the pages which are monitored by CAM turn out to be quite diversified as

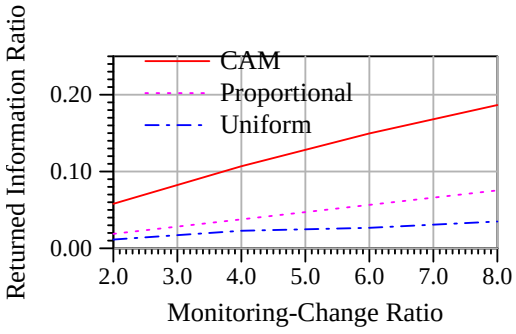


Figure 4: Performance under uniform page weight and update probability distribution

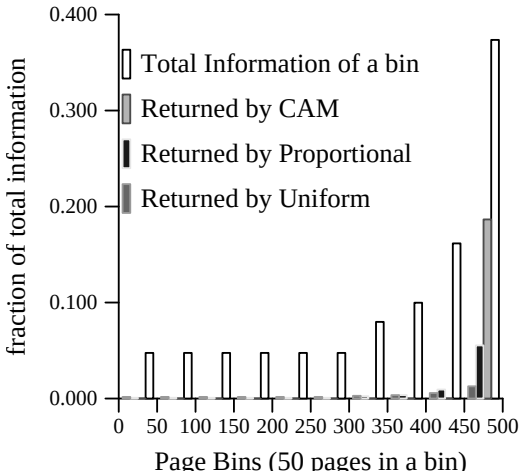


Figure 5: Characteristics of Resource Allocation Policies

pages with even lower change frequency have some update instants with a good chance of actually changing, as shown in Figure 7.

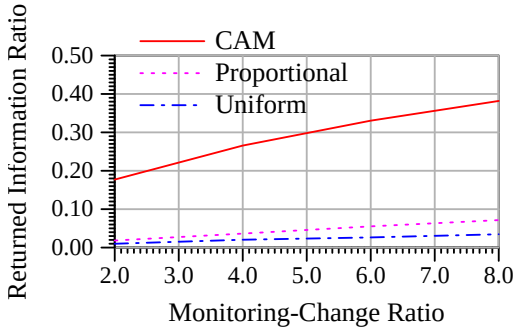


Figure 6: Under skewed update probability and uniform page weight distribution

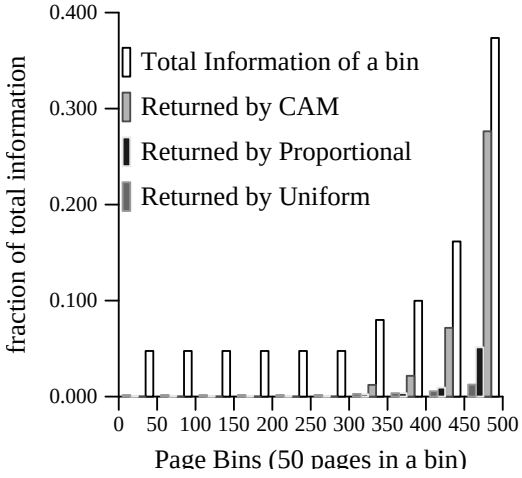


Figure 7: Characteristics of resource allocation policies

5.2.3 When page weights are skewed

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 Figure 8. Also, now it allocates monitoring tasks to those pages which have high importance and a higher probability of getting changed.

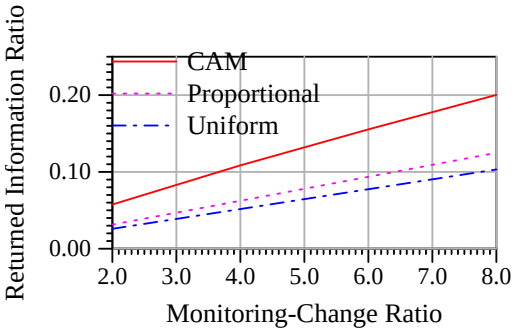


Figure 8: Performance under skewed page weight and uniform update probability distribution

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

Figure 9 compares performance of different resource allocation policies when monitoring-to-change ratio[1] is kept at 9 and the page weight distribution is uniform.

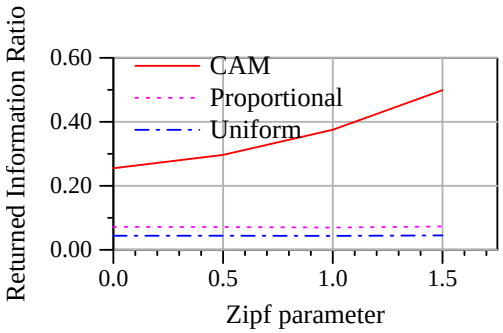


Figure 9: Performance under varying skewness of update probability distribution

It is clear 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 the same as before. As is evident from Figure 9, CAM's resource allocation policy starts performing even better than other policies as update distribution is made more and more skewed. It exhibits a 5-fold improvement over uniform resource allocation policy when the Zipf parameter is 0, and a 10-fold improvement when the Zipf parameter is 1.5. This is because CAM monitors pages preferentially at those update instants which have a high probability of returning relevant information. As the update probability distribution is made more and more skewed, CAM responds to the skewness by selecting the most beneficial instants for monitoring, thus performing even better than before. The uniform and proportional policies do not take into account the granularity of update instants and decide to monitor based on weight and change frequency. (Note that when the 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.)

[1]The ratio of available tasks to the expected number of changes over all pages.

5.4 Which settings affect CAM most?

In the previous experiment, we observed that even when we have nine times more monitoring tasks available than expected number of changes, the loss of information remains significant. This is because of the distributed and uncertain nature of page change behavior 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. Even a large number of monitoring tasks (until they become comparable the to number of update instants) will not be of much help.

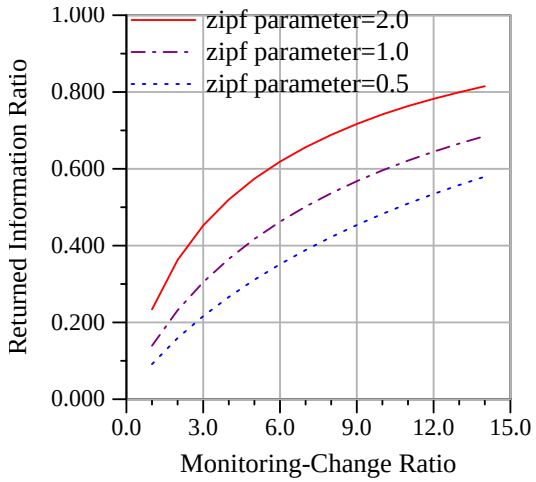


Figure 10: Performance with varying skewness of update probability distribution

Figure 10 shows how performance varies with the update probability distribution of page change behavior. It is also evident that pages with almost uniform update probability distribution will require more monitoring than the skewed case. Luckily, the Web is not updated with a flat distribution. A recent measurement study by Fetterly and others [6] showed that most web pages do not change much, and that a page's previous change rate is a good predictor of its future change rate. This means that the most favorable scenario for CAM is, in fact, reality.

We find that page weight distribution also affects the performance in a significant way. This is intuitive: if we can somehow figure out during the tracking phase that a particular set of pages is serving a major part of reporting to users for answering query, then we can improve our performance by assigning them a major share of monitoring tasks. Figure 11 shows the effect of page-weight

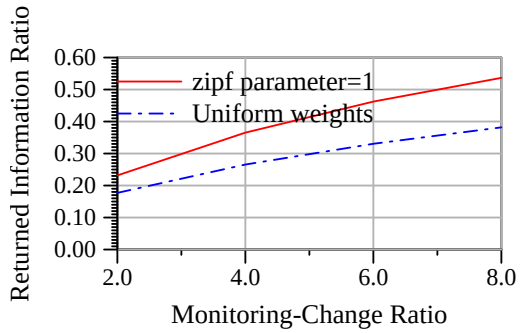


Figure 11: Performance with different skewness of page weight distribution

distribution on the performance of allocation techniques. Extending CAM to take advantage of further meta-information about the change behavior of web pages will help respond to CQs even more efficiently, and is left for future work.

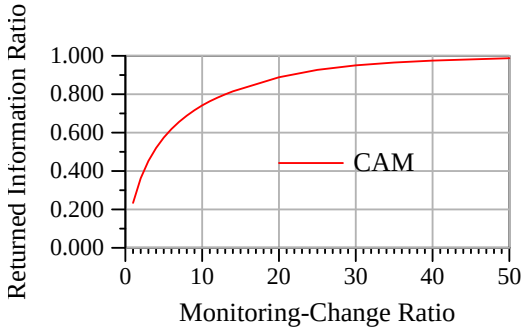


Figure 12: Performance of optimal policy

5.5 Effect of monitor-to-change ratio

Here we evaluate the practical application of our proposed scheme. As evident from Figure 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 instants while CAM needs only 20 monitoring tasks (5% of the maximum needed monitoring tasks) .

5.6 Reallocating resources

While describing CAM, we mentioned that after every epoch of length T, page change statistics are updated and resource allocation decisions reevaluated for the next epoch. This process may become very expensive, especially if T is small. Therefore, we next study the effect of 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 set to 1. Then we generate an actual event based on this update probability distribution by tossing a biased coin at every update instant and declaring a change at an instant if it turns up heads.

Before the next epoch, we modify the update probability distribution based on the recent epoch by modifying update probabilities ($\rho_{i,j}$) of those update instants which get monitored in the epoch.

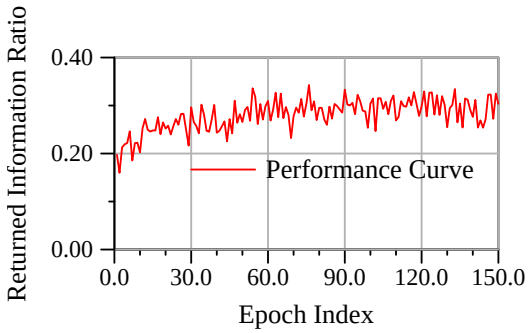


Figure 13: Effect of Resource Reallocation after every epoch

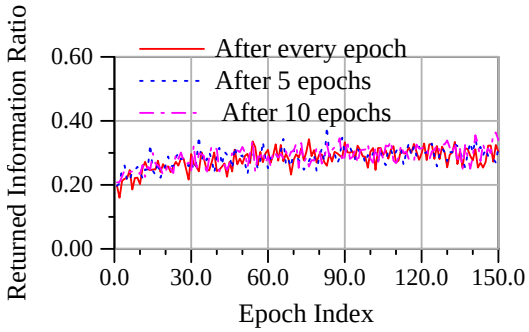


Figure 14: Effect of Varying the Resource Reallocation Delay

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 instants, the probability of expecting a change at the next update instant is simply set to 0.5.

Then we reallocate resources accommodating this new update probability distribution. As Figure 13 shows, 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.

We also plotted two more graphs, as shown in Figure 14, to study the effect of delayed resource allocation. Here the statistics are updated after *several* epochs. We find that it is not necessary to recompute resource allocation after every epoch; it can be delayed without incurring any significant loss provided the tracking phase was used to capture the initial page change behavior. This study shows that it is possible to adapt to the change behavior using the CAM approach and derive additional benefits in terms of the quality of information returned.

5.7 Performance of the Scheduling algorithm

In this experiment, we test our scheduling algorithm and show its performance. The Zipf parameter of the change frequency distribution is set to 2 and that of the update probability distribution is set to 1.

Sizes of the documents are generated as shown in Figure 15 [9]. Also, the more popular pages' sizes are set to be smaller [5]. We define the **average monitoring capacity** as the available bandwidth divided by average size of documents.

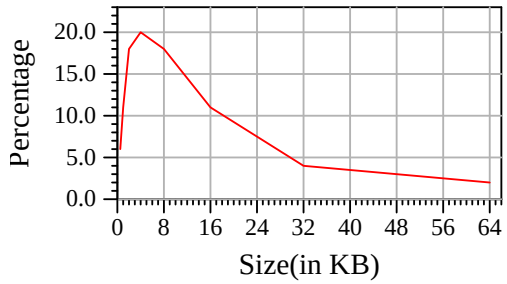
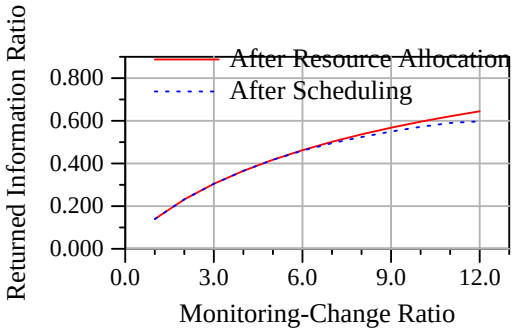


Figure 15: Size of web documents

As shown in Figure 16, our scheduling algorithm performs very well, usually accommodating all desired monitoring tasks recommended by the resource allocation phase, 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 to the 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 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 any time also becomes high. So these monitoring tasks get delayed, leading to loss of information.



Average probing capacity = 7.6*No. of expected changes

Figure 16: Allocation vs. scheduling decisions.

The experimental results lead to the following observations:

- CAM produces query results with higher coherency than either Proportional or Uniform policies. 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-to-change ratio) also leads to a larger performance improvement with CAM than with other approaches.

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

- 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

There are several systems developed for monitoring sources on the web. CONQUER[14], WebCQ[13] and C3 are the most related project to our work. The most evident differences between CAM and any of these related works is the approach of monitoring. The way source site are monitored in CAM is more efficient and optimal. CAM keeps 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 behavior 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 a near-optimal 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 allocation for CQs. CAM's resource allocation algorithm is designed to be optimal: It maximizes the value of the returned information, when the updates are quasi-deterministic, i.e., when updates occur at specific time instants 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 behaviors.

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 [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 behavior 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, 18], authors approached the problem formally and devised an optimal crawling technique. (Some aspects of our formal are adopted from [18] 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 behavior. 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 [16]. 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] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. Crawling towards light: A large scale study of the evolution of Web pages. In *First Workshop on Algorithms and Models for the Web-Graph*, Vancouver, Canada, nov 2002.

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

[8] 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.

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

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

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

[12] 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.

[13] L. Liu, C. Pu, and W. Tang. Webcq: Detecting and delivering information changes on the web. *In Proc. Int. Conf. on Information and Knowledge Management (CIKM)*, 2000.

[14] L. Liu, C. Pu, W. Tang, and W. Han. Conquer: A continual query system for update monitoring in the www. *International Journal of Computer Systems, Science and Engineering*, 1999.

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

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

[17] 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.

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

[19] 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. PROPORTIONAL VS. UNIFORM POLICY FOR CQS

Assumption : Update distribution is uniform.
We compare weighted *Uniform* and weighted *Proportional* policies.

Number of crawls allocated to i^{th} page in the 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 instant.
Information gained in case of the 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 the proportion to the 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* always performs better than *Uniform* no matter how page weights and change frequencies are distributed.

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.