Monitoring the Dynamic Web to respond to Continuous Queries

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 multiphase solution to the problem. Some of the important phases are: The *monitoring 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 *probe* these pages for changes, are allocated. Given these resource allocations, the *scheduling* phase produces an optimal achievable schedule for the probings. An experimental evaluation of our approach compared to prior approaches for crawling dynamic web pages leads to some interesting observations pertaining to the differences between the two problem of crawling—to build an index—and the problem of change tracking—to respond to continuous queries.

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. Even in the best-case scenario, a crawler visits a web site only once in a few hours. This type and frequency of crawling is insufficient to handle a class of queries known as *Continuous queries* [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.

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

Not surprisingly, the problem of keeping track of the dynamics of the web becomes inherently different for the *continuous query* case compared to classical(*discrete*) query case. 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) monitoring the pages, to determine the characteristics of changes to these pages, and from these (c) deciding when to *probe* the pages for changes, so that responses are current. The last problem has several subproblems: allocating the resources needed for probing the pages, scheduling the actual probings, and then carrying out the probing. The work involved in handling continuous queries is portrayed in Figure 1. Here, the feedback arcs from the probing phase to the earlier phases indicate that observations made during the probing phase are used to tune subsequent decisions. We

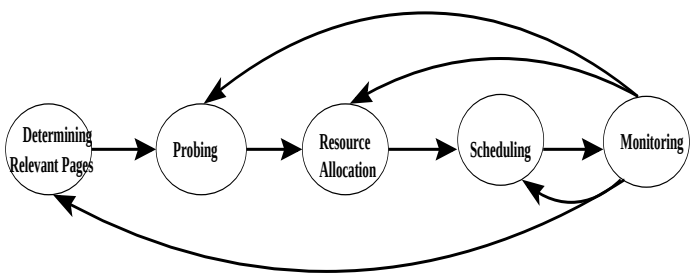


Figure 1: Different phases of our approach

use the term *probings* to explicitly account for the differences from the classical crawling problem. A *probe* fetches a web page, much like what 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. Specifically, in this paper, our problem reduces to the distribution of a given number of *probings* among the pages whose changes need to be tracked to respond to a set of *continuous* queries.

It could be argued that *discrete* queries asked periodically with some time interval can be considered to be equivalent to *continuous* queries but the following reasons should help in dispelling this misconception: Firstly, 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 when the updates are not frequent. If we set the time-interval to be large, it may lead to loss of information if more frequent updates, than expected, take place. Secondly, *con-*

tinuous queries have a non-zero lifetime and so a query system can study a query's characteristics carefully and can answer it better 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. To this end, our optimization metric is defined to minimize the information loss compared to an ideal *probing* algorithm which probes upon every change of a page.

So it should be clear by now that not only the nature of the crawling problem but optimization goals also become different when we move from *discrete* to *continuous* query case.

The rest of this paper is structured as follows: In Section 2, we define the problem formally and also provide an overview of the solution ???

2. OVERVIEW OF OUR SOLUTION

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 $q = \{w_1, w_2, \ldots w_m\}$.

Identifying Pages relevant to a set of queries: Based on the keywords specified by a user, we need to first identify pages relevant to this query. The query is considered as a bag of terms and is fed to a classical search engine which in turn returns back a set of pages it finds relevant to queries using *inverted index*. 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: 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.

Monitoring the Relevant Pages to Characterize Changes: Once relevant pages have been identified, by visiting each page at frequent intervals during a monitoring period, changes to these pages are monitored, 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 pages for queries.

With this information in hand, we can move to the next set of phases where resource allocation and scheduling decisions are made concerning the probings to be done for an interval of length T. The basic idea is that these scheduling intervals of length T repeat every T units of time and we will make decisions pertaining to the probings to be carried out in one scheduling interval using both new data and the results from the previous scheduling intervals.

In general, based on the results of probings carried out for T time units, scheduling, resource allocations, change statistic computations, and page relevance can all be revisited. These, as mentioned earlier, correspond to the arcs going from the "probings" phase to the earlier phases of Figure 1.

Let C denote the total number of *probings* that can be employed in a single scheduling interval. C is derived as an aggregation of the resources needed for probing, including CPU cycles, communication bandwidth, and memory. [1] With this information in hand, decisions are made about the allocation of a given number of *probings* among a set of pages while also deciding *when* these allocated *probings* should ideally occur.

Resource allocation phase: This phase decides the time instances at which these allocated *probings* should be done. In *scheduling* phase, we take these time instances as inputs and prepare a feasible schedule to meet our optimization measures. The overall goal of these two phases is to probe in such a way that the probes occur just after updates are expected to take place. The number of lost updates is an indication of the amount of lost information and minimizing this is the goal of the system.

We now give a summary of the *resource allocation* and *scheduling* phases using the following model (The basic general model is adopted from [16] and is modified to suit our problem definition.) We denote by N the total number of web pages that need to be monitored, which shall be indexed by i. Let P denote the set of web pages relevant to the continuous queries. λ_i is the estimated number of changes that occur in page i in T time units. Henceforth we will refer it as change frequency for page i. page may get updated.

Suppose U_i denotes the sequence of time instances $u_{i,1}, u_{i,2}.....u_{i,p_i}$ at which possible updates occur in page i. Here, p_i is the total number of update instances for i^{th} page, i.e., cardinality of sequence U_i (P_i=$|U_i|$). Note that it is not certain that a page would 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 getting this i^{th} page updated at the j^{th} instance.

We assume $0 \leq u_{i,1} \leq u_{i,2}.....u_{i,p_i} \leq T$ and $u_{i,0} = 0$ and $u_{i,p_i} = T$. It should be clear that if we decide to *probe* 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 *probings* allocated for a page is equal to the number of update instances, then we can always maintain a fresh copy of this page by *probing* at all possible update instances. But in practice we will not be able to perform as many probings as the number of update instances. So we need to pick a set of update instances at which only this page would be probed and not at others. Hence with every update time instance, we associate a variable $y_{i,j}$ where

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

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

Scheduling the probes: This involves taking the ideal timings for the probings of each page and obtaining an optimal achievable schedule out of it. We map this problem to *flow-shop* scheduling problem [13] with the goal of minimizing the average *completion time*. Though this problem has been proved to be *NP-Complete* [11], there are fast approximation algorithms [2] that provide an approximate bounded solution. Next we *probe* these pages according to the designed schedule and when this *scheduling interval* is finished, we go back to query system and update the characteristics

[1] For example, the authors of [9] report that with two 533 MHz Alpha processors, 2 GB of RAM, 118 GB of local disk, and a 100 Mbit/sec FDDI connection to the Internet, *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 *probings* that it is capable of on average.

of these pages on the basis of the observations done in the *preceding scheduling interval*.

3. OPTIMAL RESOURCE ALLOCATION

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 importance of page for queries. Page-change behaviour is studied during a *monitoring* phase and is denoted by associating a probability of change with every update instance. Next we would 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 *monitoring* period.

3.1 Importance of Specific Pages

Glossary

Q : Set of all queries submitted in the system.

Q_{p_i} : Set of queries for which i^{th} page is found to be relevant in *monitoring period*.

$w_{i,j}$: Estimated relevance of i^{th} page for j^{th} query. It would be positive for all queries $q \varepsilon Q_{p_i}$ and zero for all $q \varepsilon Q\ Q_{p_i}$. It would be computed during *monitoring* period.

ω_i : Importance of i^{th} query. This would be an input to the system.

W_i : Weight of i^{th} page. It is computed as shown below.

It is clear that not all pages would be equally important for each query in the system. So we would rank pages by assigning *weight* to each page using its relevance measures for queries. The *weight* of a page denotes the value of current copy of page and if page gets updated before the current copy is *probed*, we assume that it incurs us a loss of W_i value. Suppose there are Q queries submitted in the system. Each page would be relevant for a set of queries. Say the i^{th} page is relevant for a query set denoted by Q_{p_i}. Also the sequence of values measuring relevance of i^{th} page for each query is $w_{i,1}$, $w_{i,2}$.....$w_{i,|Q_{p_i}|}$. Note that these relevance measures would be calculated during monitoring period and after every T time units when a scheduling interval is finished, it would be updated based on the earlier values and values found in last scheduling interval.

Also let there be *Importance* measure associated with each query because of the classification of users submitting the query or classification of query domains, say ω_1, ω_2.....ω_Q . So now we can measure the *weight* W_i of ith page as

$$W_i = \sum_{j \varepsilon Q} (\omega_j w_{i,j})$$

where Q is a set of queries in the system,

ω_j is *importance* of jth query,

$w_{i,j}$ is relevance measure of ith page for jth query and is greater than 0 only if $q_j \varepsilon Q_{p_i}$

3.2 Goals of the Resource Allocation Phase

For *Continuous* queries, our aim is to minimize weighted importance of changes that are not reported to users . It could be formulated as

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

where E_i denotes expected number of lost changes for ith page

As we have assumed that each update instance is independent of others,

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

Resource constraint is given by

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

Note that here we are assuming that in T duration, relevance of each updated copy of page for queries remains the same as estimated during *monitoring* period. After a *scheduling interval* is finished,we will update these relevance measures on the basis of relevance measures actually found in the interval preceding it. So unless T is kept very large or page updates are very erratic, our assumption remains very practical. Also we are assuming that with each update of page, information of update preceding it is completely lost. That is, the number of lost updates is an indication of the amount of lost information. While this may not always be true, it gives us a simple way to state the goal to be accomplished.

In certain cases, it is possible that we would be having more information 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 monitoring period that a particular news site mainly declares health updates only once at the start of day and in rest of the time, it remains mainly concerned about political and sports updates, then we can characterize 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 our resource allocation problem can be formulated as

$$min \sum_{i \varepsilon P} E_i$$

where E_i denotes the weighted expected number of lost changes for ith page

$$\text{So } E_i = \sum_{k \varepsilon Q} \omega_k . w_{i,k} . \sum_{j \varepsilon U_i} \rho_{i,j,k} (1 - y_{i,j})$$

with resource constraint as

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

If we can extract even more information by measuring not only probability of change of i^{th} page at update instance $u_{i,j}$ but also average importance of change at this time instance, then it would make resource allocation even more efficient. For example, suppose we found 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. It is easy to accommodate aforementioned extension in the above model, and so we won't concern ourselves with it due to lack of space. **Is this last paragraph ok as it seems quite impractical. We may argue that though above extension seems to assume very much information, it may possible in continuous query case to actually have this much information because we do get a lot of time to observe pages in continuous query case.**

3.3 The resource Allocation Algorithm

Both of the above formulated resources allocation problem 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

probings. Either a probing would be 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 in theoretical Computer Science [10]. Formally it can be stated as shown below:

$min \sum_{i=1}^{N} F_i(x_i)$

with resource constraint

$\sum_{i=1}^{N} x_i = R$

where $x_i's$ are discrete and $F_i's$ 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 $O(N(logR)^2)$. The fastest algorithm is due to Frederickson and Johnson [8] and it has complexity $O(max\{N, Nlog(R/N)\})$. In our case, the output of these algorithms is a set of $y_{i,j}$'s which get a value of 1 to meet our optimization measure. This set in turn gives us the number of *probings* allocated to a page (x_i=$\sum_{j=0}^{p_i}(y_{i,j})$) as well as the ideal time instances at which these allocated *probings* should be done.

4. SCHEDULING OF PROBINGS

At this point we know the number of *probings* allocated to each page and also the ideal time instances at which these *probings* are supposed to be done. In practice, we have a set of *M* parallel processes which continuously perform these probings. Now our task is to schedule the *probings* among *M* parallel probes. While determining any schedule, our aim is to minimize the total delay occurring between the ideal time instances and the actual scheduled time instances.

Let page P_i be allocated x_i number of *probes* in an optimal resource allocation. Also the time instances at which these x_i *probes* should be employed are t_1, t_2, t_3.....t_{x_i}. Let $fetch_i$ be the average fetching time for 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 *probing* could be regarded as a *job* whereas the probing 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(r_j)* and the time for which it needs to be run on a machine is called the *processing time*. So in our case, ideal *probing* time instances t_1, t_2, t_3.....t_{x_i} would be the *release times* and fetching times of pages would act as *processing times(p_j)* for jobs. Our goal is to minimize the delay d_i between ideal *probing* time instance (r_i) and actual time instance s_i of scheduling.

In our case all jobs are equally important as there is no weight assigned with each *probing*. So our problem could be formulated in scheduling notation as $R|M|r_j \geq 0|\sum_j Cm_j$ meaning that *R* jobs of non-trivial release times are available for scheduling at *M* machines with goal as minimization of average completion time. Here Cm_j denotes completion time for job j. It might not be clear how minimizing average completion time would lead to minimization of average delay time. It is because of the following equality :

Total Completion Time

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

As $\sum_{i=1}^{R}(r_i)$ 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 simpler problem $R|1|r_j \geq 0|\sum_j Cm_j$ don't have any polynomial time algorithm and has been proved to be *NP-Complete* [11]. So we have to look for *approximation algorithms*. As our problem is *offline*, there is 1.58-approximation algorithm [11] that could be used for it. We have used this algorithm in conducting 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 proposed an optimal *resources allocation policy* for probing changes in pages relevant to *continuous* queries. Here we evaluate our policy by comparing it with some classical policies using synthetic a data set. These policies are *Uniform* and *Proportional* policy [4] in which resources*(probings)* are allocated uniformly across all pages or proportional to change-frequencies of pages respectively. As suggested in [16] , 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 *probings(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. 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 for queries submitted. It is also set to 500.

3. *C* : number of *probings* available. It is varied from 1000 to 50000 in our experiments.

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=2) to uniform (when θ=0). This distribution is henceforth referred as *change frequency distribution*. Unless otherwise specified, θ is set to 2 in experiments.

5. *Update probability distribution* : As said above, we have used a universal sequence of update instances(U) in our experiments. In this universal sequence, update instances are uniformly distributed throughout the duration *T* every δ time units. 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 any a priori assumptions about page change behaviour and helps in evaluating our policies for all possible 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 [15] show that popularity of pages vary in *zipf* fashion as shown in Fig 2. Drawing an analogy, we make relevance of page j for a query

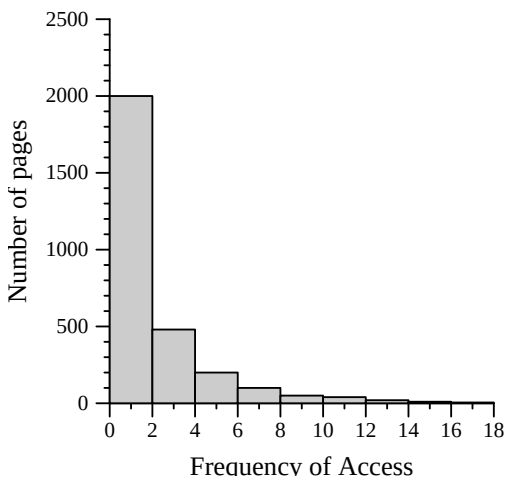


Figure 2: Observed Popularity Distribution

i ($w_{i,j}$) to be distributed according to *zipf* distribution in our experiments. This distribution will be referred as *query relevance distribution*. Also the more dynamic a page, the more more popular it is too, as shown in [17]. So we make the important pages be more dynamic in our experiments by assigning *maximum* value of *page relevance distribution* 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 shown in section 3. The distribution according to which W_i varies is referred to as *page weight distribution*.

8. *probing ratio*: denotes the ratio of the total number of *probings* to, ($\sum_{i \epsilon P} \lambda_i$), i.e., the number of actual changes expected in time T time.0

Performance Metric: Returned Information ratio: Part of the *information* that is returned by a policy in a fixed number of probes is called as *Returned Information Ratio*. Using section 3, it turns out to be

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

where $y_{i,j}$ is defined in the same way as in section 3. 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 compare three aforementioned resource allocation polices and also observe the effects of *update probability distribution* and *page weight distribution* on their performance.

5.2.1 When both page weights and update probabilities are uniformly distributed

We make both these distributions uniform and set *Zipf* parameter of *change frequency distribution* to 2 as shown in Fig. 3.

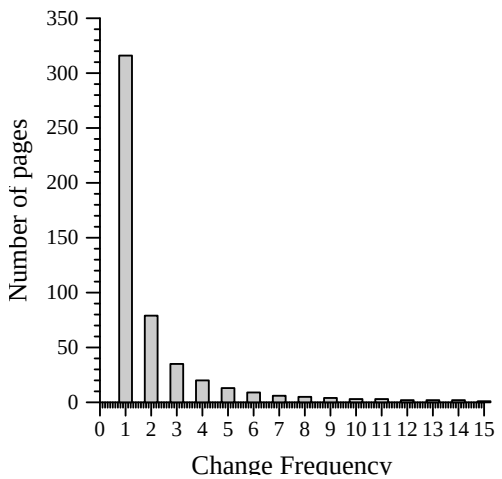


Figure 3: Change frequency distribution

Uniform *page weight distribution* means that all pages are having 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 2 important observations.

1. Proportional policy performs better than its Uniform counterpart. This is very surprising as all earlier studies showed the reverse to be true [4] [16]. The reason becomes pretty clear when we look into the nature of the crawling vs. the probing problem. In our case, we are answering *continuous* queries and naturally our aim is to detect as many changes as possible. So when all other parameters are uniform(page-weight and update probability distribution), one would certainly expect more benefit by *probing* those pages which are having high change frequency(λ_i) because ultimately only these pages are having maximum 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*

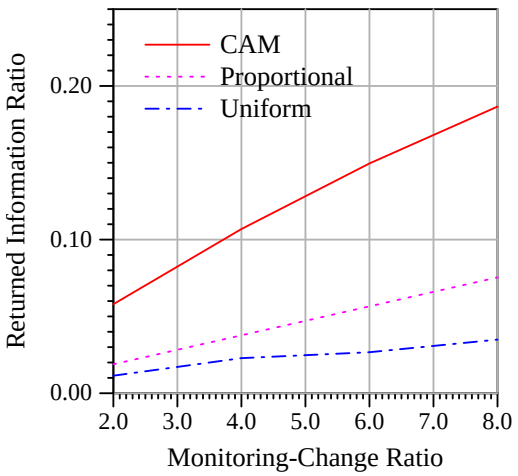


Figure 4: Under uniform weight and update probability distribution

of page which is found to be of *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 at the end of the paper.

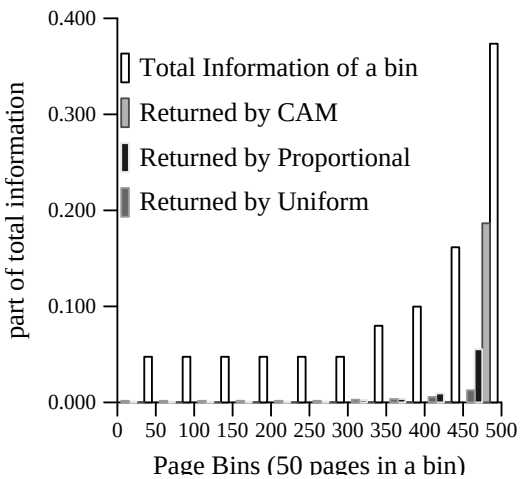


Figure 5: Characteristics of resource allocation policies

2. Optimal policy also allocates more *probings* to more dynamic pages but it does it even more aggressively than Proportional. Fig. 5 shows that Optimal allocates all its *probings* 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 *probed* are those which have high probabilities of actually showing changes. Proportional too does this but it allocates *probings* in proportional manner only while Optimal does it in more biased way and so it gets even better performance. As it is evident from the graph that optimal policy performs 300% better than proportional policy and around 600% better than uniform policy!

Obviously now if we start decreasing the skewness, i.e., the *zipf* parameter, of *change frequency distribution* too, then policies start coming closer and in the extreme case, they all become the same when frequencies are made to be distributed in a completely uniform manner (*Zipf* parameter = 0).

5.2.2 *When page update probabilities are skewed*

Here, we skew the *update probability distribution* with *zipf* parameter being set to 1. So pages are still of equal importance but for each page, the update instances are no more equi-probable, in experiencing a change. Fig. 6 shows the performance graph. for this data set. Again, optimal performs best leaving other allocation policies far behind. It is 12 times better than uniform policy. But in this case, the pages which are *probed* under optimal allocation 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. 6.

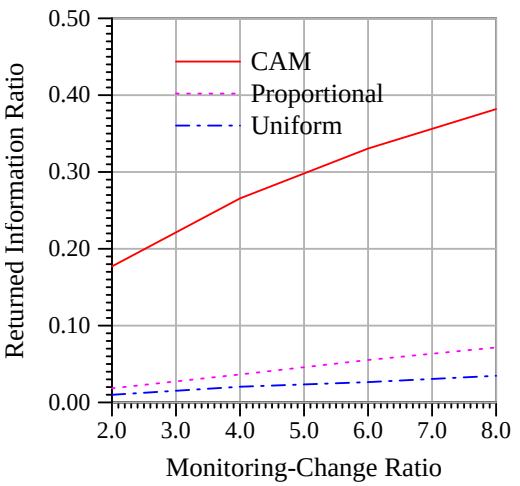


Figure 6: Under skewed update probability and uniform page weight distribution

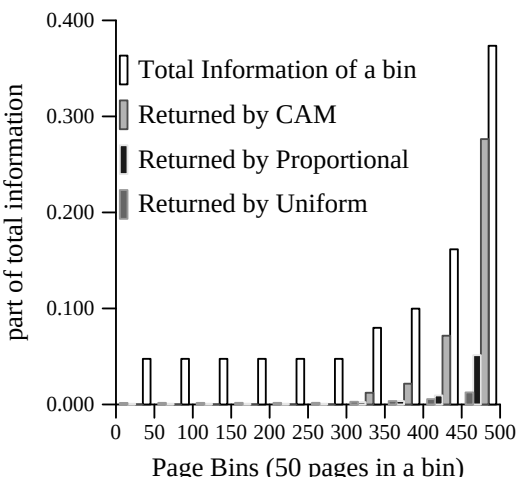


Figure 7: Characteristics of resource allocation policies

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 Fig. 8. Also, now it allocates *probings* to those pages which have high importance and a higher probability of getting changed.

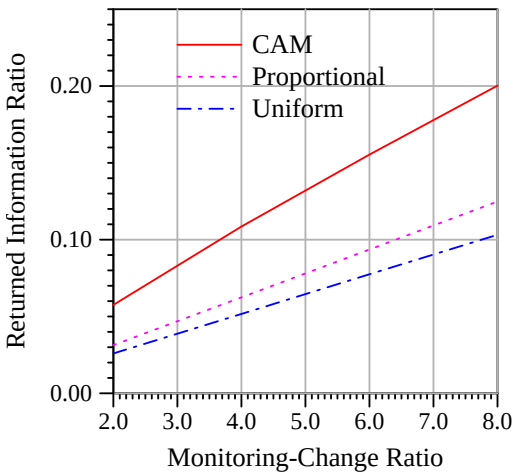


Figure 8: Under skewed page weight and uniform update probability distribution

5.3 Effect of varying the skewness of the update probability distribution

In this experiment, we compare our resource allocation policy with *proportional* and *uniform* policies under varying skewness of *update probability distribution*. This will also help us investigate the drawbacks of these classical approaches in *probing* web pages for answering *continuous* queries.

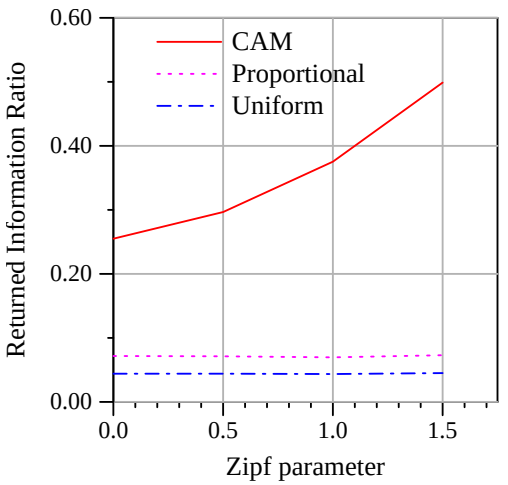


Figure 9: Under varying skewness of update probability distribution

Fig. 9 compares performance of different resource allocation policies when *probing ratio* is kept at 9 and *page weight distribution* is uniform. It is clear from the curves that for this data set, optimal resource allocation policy always performs better than other resource allocation policies. To emphasize the difference, we varied the *update probability distribution* keeping other parameters same as before. As evident from Fig. 9 that optimal 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.

This is because of the fact that in optimal resource allocation policy, a *probing* is made at those update instances which have high probability of returning relevant information and so as *update probability distribution* is made more and more skewed, it copes up with the skewness of data by selecting the most beneficial instances for *probing* and performs even better than before. But this is not the case with *uniform* and *proportional* policies as they do not look at the granularity level of update instances and decide *probings* only 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 Parameters that produce Better Results for Continuous Queries

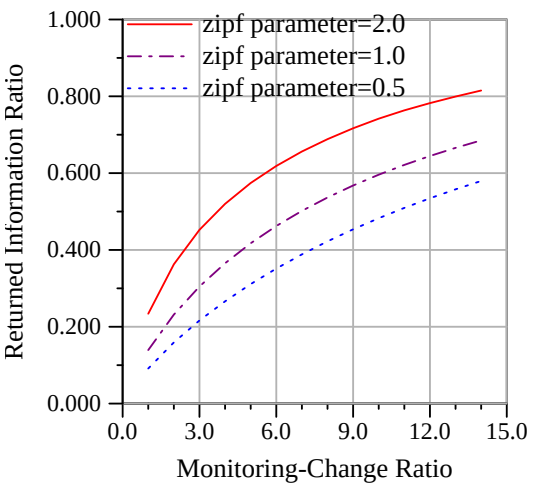


Figure 10: With varying skewness of update probability distribution

In the previous experiment, we observed that even when we have 9 times more *probings* available than expected number of changes in T, the loss of information remains significant. We feel that this is because of the distributed and uncertain nature of page change behaviour which make number of *probings* 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 *probings* (until they become comparable to number of update instances) won't 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* would affect the performance for *continuous* queries. We find that *page weight* distribution also affects the performance in a significant

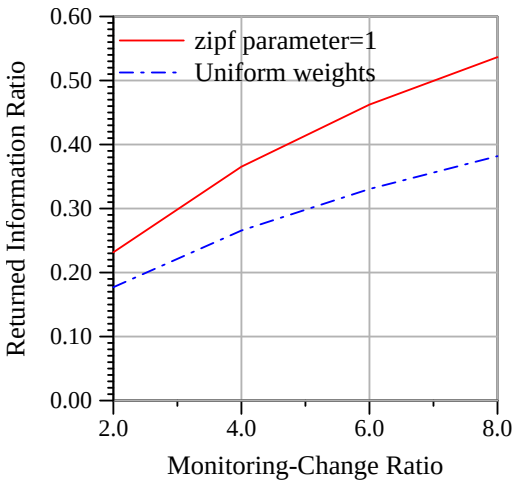


Figure 11: With varying skewness of page weight distribution

way. It is justified too because if we can somehow figure out during *monitoring 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 *probings* also. Fig. 11 shows the effect of *page-weight* distribution on the performance of allocation techniques.

Continuous queries can be responded to even more efficiently by extracting meta-information about the change behaviour of web pages. We need to have more knowledge of page characteristics to get a good performance for answering *continuous* queries .

5.5 Effect of Probing Ratio on Continuous Queries

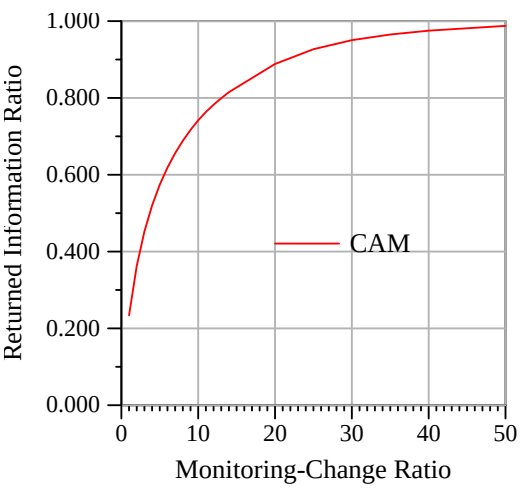


Figure 12: Performance curve for optimal policy

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 *probing ratio* is 20. Without using our probing technique, retrieving 90% of the information would require *probing* of at least 420 instances while our scheme reduces this to only 20 *probings* (5% of blind probings). So it is indeed helping us in monitoring the pages but it is the uncertain nature of page changes which is actually impairing the performance.

5.6 Reallocating resources

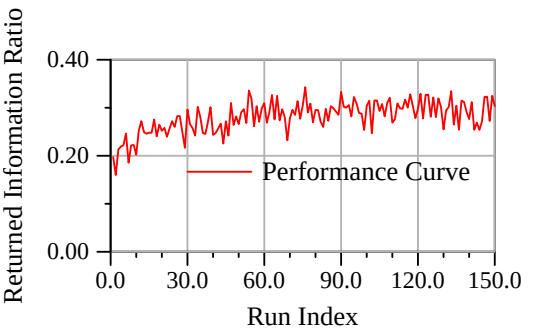


Figure 13: Effect of Resource Reallocation after every run

As we said while describing our technique that after every run of length T, we update page change behaviour and accordingly modify resource allocation for next run. But this may become very expensive especially when T is small. We next study the effect of the the *resource allocation delay*.

We making many *probing runs* of period T assuming that a brief *monitoring period* preceded it in which page change was fully captured and then we would show how performance varies with each run and with varying delays in repeating resource allocation. We start with *update probability distribution* of *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 that instance if it falls head.

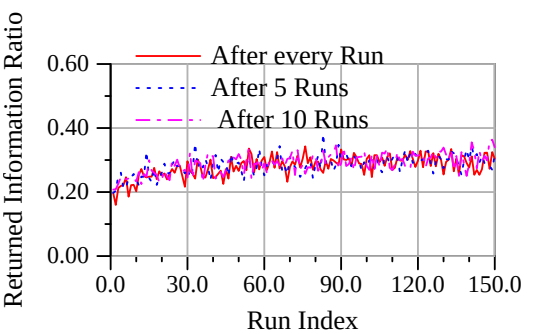


Figure 14: Effect of Varying the Resource Reallocation Delay

Before next run, we modify the *update probability distribution* based on this last run made by modifying update probabilities($\rho_{i,j}$) of those update instances which get *probed* in this run. We do this modification in very naive fashion by estimating average rate of occurring of updates at this update instance. So if a page got updated on 5 occasions in last 10 probings to this page at a certain update instance, we assign 0.5 probability of expecting a change at this update instance. Then we reallocate resources accommodating this new *update probability distribution*. As Fig. 13 shows that

performance of our allocation policy do increase in initial runs and then it becomes steady. This is what one would expect because after a large number of runs, 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. We find that resource allocation is not required to be done after every run and can be delayed without incurring any significant loss provided the *monitoring phase* does capture the page change behaviour nicely and also only if page change behaviour is not very erratic.
Not putting the other case when monitoring is fallacious because of no good experiment in support of it

5.7 Performance of Scheduling algorithm

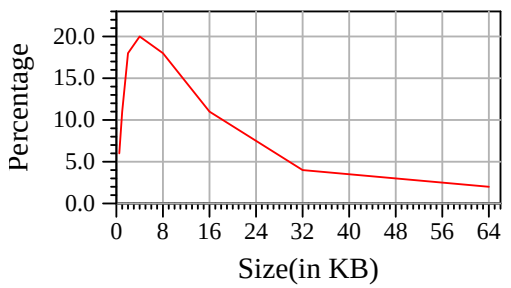


Figure 15: Size of web documents

In this experiment, we test our scheduling algorithm and show its performance. *Change frequency distribution zipf* parameter is set to 2 and *update probability distribution zipf* parameter to 1. Sizes of the documents are generated as shown in Fig. 15 as found in [9]. Also more popular pages are set to smaller sizes in accordance with [6]. We define average *probing capacity* as available bandwidth divided by average size of documents. Note that this is average analysis and it is possible to make more probings than average probing capacity. As shown in Fig. 16, our *scheduling* algorithm performs very good and is almost *lossless* when number of *probings* is less than average *probing capacity*. Even when number of *probings* required to be scheduled exceeds average *probing capacity*, the loss of information incurred in *scheduling* phase remains quite negligible in comparison of *resource allocation* phase. The two kinds of losses incurred are:

1. As the number of *probes* to be scheduled becomes more than the average *probing capacity*, some *probes* remain undone and so some loss of information is incurred.

2. Also as number of probes becomes more and more, the chances of cases where probes scheduled for a instance exceeds the *probing capacity* at that instance also becomes high. So these *probings* get delayed for time being and hence loss of information is incurred.

6. PROOF

Assumption : Update distribution is uniform.
Aim : Proportional policy always performs better than Uniform policy under above assumption.
We would be comparing weighted uniform and weighted proportional policies.

Number of crawls allocated to i^{th} page in proportional policy is

$$\frac{W_i * \lambda_i}{\sum_i W_i * \lambda_i}$$

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

$$\frac{W_i^2 * \lambda_i * \rho_i}{\sum_i W_i * \lambda_i}$$

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

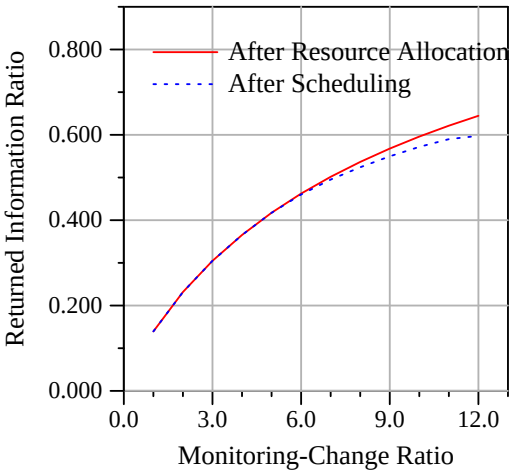
$$\frac{W_i^2 * \rho_i}{\sum_i W_i}$$

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.



Average probing capacity = 7.6*No. of Expected Changes

Figure 16: Performance Curve for different phases of scheme

7. CONCLUSIONS AND RELATED WORK

As mentioned earlier, in our problem formulation we are not making any assumptions about the change behaviour of pages. Instead we collect and build the above statistics about a page during a *monitoring* 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 probings done. This makes our solution robust and adaptable in any web scenario.

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. A series of papers, [4] and [3] addresses a number of issues relating to the design of effective crawlers. In [5][16], authors approached the problem formally and devised an optimal crawling technique. 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 [**?**] 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 basic assumptions and present an *optimal monitoring* technique for answering *continuous* queries [12] independent of any assumption about page change behaviour. We also show that traditional crawling technique don't work well for answering of *continuous queries* because of the change in nature of problem as well as goal to be achieved. We formally prove that *proportional allocation* works better than *uniform policy* which differentiates the *continuous* and *discrete* query case.

There is also an altogether different approach possible for answering *continuous queries* where information is *pushed* from web pages instead of *pulling* it as done in our scheme [14]. Here users submit their query to the query system and then query-system registers itself to all the web pages it finds relevant to the submitted queries. Now whenever these web pages get updated, they themselves propagate their changes to the query-system which in turn report back this to the user only if it finds that changes are relevant for user query. An advantage of this approach is that here query system is no more required to monitor web pages and so there is no wastage of resources too but an obvious disadvantage of this would be dependence on web sites which may or may not propagate information at correct time, particularly when there are large number of query systems that need to be reported. Also some web sites might not even allow registration of query systems.

To our knowledge, no earlier work has focused on the aspect of *probing* the relevant web pages to respond to a set of *continuous queries*. (1) We present optimum probing techniques and also show how traditional crawling approaches won't suffice for answering *continuous queries* and need to be improved. We do not make any assumptions about the way in which pages change. Most of the earlier crawling strategies assume a *Poisson* update process. Instead of this, what we do is to record information about the nature of change of pages and keep on evolving it every time a page is *probed*. This makes our study more practical and robust as it is not designed with any basic assumptions. (2) Our optimization metric is defined to minimize the information loss compared to an ideal *probing* algorithm which probes upon every change of a page. This metric also differentiate crawling from *probing*. (3) We formally prove that *proportional allocation policy* in which the pages which are having high frequency of change are allocated more *probings*, works better than *uniform policy* which allocates equal number of *probings* to each page independent of its change frequency, in *continuous* query case which on the first sight seems to be in contradiction with a long held result that *uniform* is a better allocation technique than its *proportional* counterpart [4]. We justify this surprising behaviour and also give an intuition behind it. This shows that nature of problem of *monitoring* of web pages for answering *continuous* queries is strikingly different from the problem of devising optimal crawling techniques addressed in earlier studies.

8. 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] C.Chekuri, R.Motwani, B.Natarajan, and C.Stein. Approximation techniques for average completion time scheduling. *In proceedings of 8th ACM-SIAM Symposium of Discrete Algorithms*, pages 609–618, 1997.

[3] 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.

[4] 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.

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

[6] 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.

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

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

[15] 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.

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

[17] 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.