Sampling-based Refresh Policy with Correlation

Ali Taleghani
School of Computer Science
University of Waterloo
Waterloo, Canada
atalegha@cs.uwaterloo.ca

Yasemin Ugur-Ozekinci
School of Computer Science
University of Waterloo
Waterloo, Canada
ugur-ozekinci_yasemin@emc.com

ABSTRACT

For many Web users search engines represent the starting point of their journey. Given a small number of keywords the search engines returns the pages relevant to that query using its local page repository. The problem of repository freshness is a major obstacle for search engines. It is necessary to detect as many changed pages as possible and update them.

In this work, we present a refreshing policy that is based on sampling with correlation. In our approach, we first categorize the downloaded pages into categories that best describe the topics those pages have a reputation on. Then we take small samples from each category and determine how many of the sampled pages have changed. Based on these results, we make a decision on whether the entire category should be updated. We will present our method and show experiments on real data.

1. INTRODUCTION

Over the past decade the size of the World Wide Web has increased to unbelievable dimensions. While many surfers use Web portals such as Yahoo to gain access to this sea of information, most internet users gain access to the Web by using search engines. Given a list of keywords, a search engine returns the pages that are most relevant to the query.

In order to be able to return search results to the user, search engines have to have a large page repository that contains most of the Internet. A *crawler* is used to crawl the Web and download pages to the repository. When the user submits a query, the local repository and indexes are queried and the results returned to the user. One major difficulty with this approach is related to the fast-changing nature of the Web. A page in the repository might have become "out-dated" since the time it was last crawled and downloaded by the crawler. Therefore, the results returned to the user might not reflect the current state of the page.

This problem has motivated much research into the issue of repository staleness [4, 6, 5, 3]. In general, the search engine performs batch downloads at regular intervals at off-peak hours to refresh the repository. Due to network and computational resources, the repository is much larger than the number of pages that can be download at each cycle. For example, a search engine might be able to only download 10% of the total repository size and perform updates only once a month. Also, as the size of the repository increases it becomes more and more difficult to compare local data items with the data items at the source site to detect change and in many cases it might be necessary to "guess" the number of changes.

The authors in [6] propose a new approach to change detection that uses sampling and the authors show that their method is superior to previous methods. Sampling is performed at the site level, and based on the results of the sampling, the whole site is updated if necessary. In this paper, we build on the work in [6] and propose a new sampling-based change detection technique that takes the correlation of pages into consideration. In this new approach, we first divide the downloaded pages into categories that best describe the topic that a page has a "reputation" on. Consequently, we sample a few pages from each category and decide based on the sample whether a category needs to be downloaded and refreshed.

In order to design such a method, we will investigate how pages can be categorized into categories and show how we used a known technique for our approach. Second, the question of an optimal sample size has to be investigated and finally, we have to show that our method is more effective in detecting change than other methods such as the ones described in section 2.1.

This paper is organized as follows: Section 2 discusses the framework used in our method. We will describe the most popular download policies that we compare our method to. This section also outlines the various evaluation metrics that can be used to compare different policies. Section 2 also presents a motivating example that demonstrates the difference between the sampling-based policies, one that uses correlation and one that does not.

Section 3 presents our main work on correlation and sampling. We present the method used for establishing correlation between Web pages. This section shows how, given an

URL, the topics that the page with that URL has a reputation on are determined. Section 4 describes our work on sampling given a set of categories. This section also discusses the optimal sample size.

Section 5 presents the experiments we conducted to test our method and compares it to other methods. As will be evident, our results need further investigation since we were not able to conduct all the necessary experiments.

Section 6 presents related work and section 7 concludes our work and presents some future possible directions.

2. FRAMEWORK

In this work we will assume that the source data items are updated independently from the local copy. While some research on *push models* [10] exist in which the data source informs the local copy of change, we do not consider such models here. The major reason for this being that such models do not apply to the Web in which Web site administrators update their sites without informing the search engine.

We also assume that the size of the local repository is larger than our download resources and that we can only perform periodical batch downloads. For example, we could have a scenario in which we have 100,000 pages in the repository, but can only download 2000 pages each weekend. In the remaining of this section we will provide an overview of possible download policies and evaluation metrics.

2.1 Download Policies

Given the situation that only a subset of pages in the repository can be updated, the crawler has to choose a policy to refresh the pages. There exist many policies and the most popular are explained below.

1. *Round-robin*: In round-robin download policy, pages are downloaded in a round-robin fashion until all pages have been refreshed. For example, if there are 1000 pages in the repository, but we can only download 100 per week, each week 100 pages are downloaded until all 1000 pages have been refreshed.

2. *Change-frequency based*: In this download policy, the decision on which pages should be refreshed first is based on the change history of the data items. For example, if we downloaded a page each month over a year and detected four changes in total, we might conclude that this pages changes four times a year. A more detailed description of this download policy is provided in [7, 4].

3. *Sampling-based without correlation*: In sampling based download policy, a small number of Web pages are sampled from each site and based on that, an estimate is made on how many pages have changed in each site. Then the download resources are allocated to each data source based on the estimates. This policy is described in more detail in [6].

4. *Sampling-based with correlation*: The difference between this policy and the one described above is that sampling with correlation uses the assumption that related pages change together. As a result, related pages are first grouped together and then samples are taken from each group. A group of correlated pages is updated if the sample indicates a large enough change.

The round-robin policy is the one used by most systems because of its simplicity [2, 9]. The main advantage of this method is that it guarantees that all data items are downloaded at certain intervals. It does, however, not have the best efficiency compared to the other methods. The frequency-based policy has the advantage of performing best if it has the correct change frequency [4], however, it is very difficult to estimate the change of data items accurately. In addition, keeping past history data adds a large overhead to the system.

In general, both of the sampling based policies described do not have the disadvantages of the frequency-based policy and also perform better than round-robin (it has at least been shown for sampling-based without correlation and we intend to show the same result for sampling-based with correlation). Both of these policies do not need to keep data from previous crawls which reduces the overhead.

When discussing sampling-based policies one has to decide whether additional pages should be downloaded in a *proportional* or *greedy* manner. If the policy determines that site in A 70% of the pages and in site B 30% have changed, then a decision has to be made on how the remaining resources should be divided up between site A and B. Under the proportional policy the remaining resources are allocated to a site proportionally to its number of changed samples. In this case, we would allocate 70% of the remaining resources to site A and 30% to B. Under the greedy policy we start with the site with the most changes and download all pages in that site. If there are resources remaining, we will allocated them to other sites. In our example, we would allocate all remaining resources to site A first and if any other resources are left we would allocate them to B.

The authors in [6] argues that for sampling-based policy without correlation, no correlation between the data items in a site is necessary. The paper argues that since the sampling is random we can expect that if the sample shows a change in the data source, then with high probability there is a real change in the data source.

In the current work, we argue that this assumption does not provide the best performance and a sampling-based policy that takes correlation into consideration performs better. The intuition behind this is that most Web sites have very heterogenous data. A Web site usually consists of different sections with each section having a different change frequency. There are sections on a site that never change and there are sections that change more often than others. For example, in a news Web site, the front page, the world politics and sports pages might change more frequently than a travel or science section that might only be updated weekly. In general, the entire site is not updated together.

We, however, expect that correlated pages that belong to different sites change together. While this phenomenon might not apply to all pages, in general, much information on the Web is duplicated, and change in one source propagates to other sources. As a result, we can expect that if page A and B are correlated, that a change in page A will result in a change in page B. Consequently, it might be possible to make decisions regarding the state of B by only considering A.

2.2 Evaluation Metrics

To be able to compare different download policies, it is important to use an evaluation metric that is meaningful and can be used for all policies. There exist several ways to compare performance of a refresh policy and we will describe three of these below. In this paper, however, we will use the *ChangeRatio* metric as the evaluation metric as it is easy to measure on real data and it is also proportional to the other two metrics [6].

2.2.1 *ChangeRatio*

The metric *ChangeRatio* is based on the number of changed data items that were downloaded in a cycle. In particular, *ChangeRatio* is equal to the number of downloaded pages that had changed over the total number of downloaded pages:

$$ChangeRatio = \frac{Downloaded\ and\ changed\ pages}{Total\ number\ of\ downloaded\ pages} \quad (1)$$

For example, if we downloaded 100,000 pages and detected that 60,000 had changed, then by using equation 1 we would obtain a *ChangeRatio* of 0.6. In general, we are interested to maximize the *ChangeRatio* .

2.2.2 *Freshness*

The *freshness* metric was first proposed in [4]. Informally, item o_i is "fresh" at time t if o_i is up-to-date and not otherwise. Equation 2 defines *freshness* formally.

$$F(o_i;t) = \begin{cases} 1 & \text{if } o_i \text{ is up-to-date at time } t \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

In equation 2, "up-to-date" means that the locally stored data item is identical to the data item at the source. The *freshness* of the entire local copy at time t can be defined as

$$F(U;t) = \frac{1}{|U|} \sum_{o_i \in U} F(o_i;t) \quad (3)$$

where U represents the set of all local copies. For example, if the local repository contains 1000 pages and 750 of them are up-to-date at time t then the *freshness* of our repository is 0.75. As with *ChangeRatio* , the goal is to maximize *freshness* .

2.2.3 *Age*

The *age* metric was first described in [4] and is related to the time since a data item has been modified. Formally, it is defined as follows

$$A(o_i;t) = \begin{cases} 0 & \text{if } o_i \text{ is up-to-date at time } t \\ t - \text{modification time of } o_i & \text{otherwise.} \end{cases} \quad (4)$$

The age of the entire local copy is defined similarly to that of *freshness*

$$A(U;t) = \frac{1}{|U|} \sum_{o_i \in U} A(o_i;t) \quad (5)$$

As an example, consider a data item in our repository that changed 8 hours ago in its source and we have not downloaded it yet. As a result, the age of this data item is 8 hours. It is important to realize that *freshness* and age are in general difficult to measure as it is necessary to instantaneously compare all data items to the source items, which is in practice very difficult due to the large size of the repository. Often, when these two metrics are used, some stochastic models are assumed for data changes.

2.3 Example

In this section, we will provide a small motivating example that compares our approach to that of [6]. For this example, we will use the change ratio metric in order to compare the two methods and also use the greedy policy described above.

The scenario is as follows: There are two Web sites A and B. Each site has 100 pages for a total of 200 HTML pages. Assume that after considering correlation between these pages, we have determined five categories C_1, C_2, C_3, C_4 and C_5. Each category has 40 pages from sites A and B. Each category has 20 pages from site A and 20 pages from site B(In this scenario, we do not consider the possibility that pages belong to several categories, but in reality, this is most probably the case).

Lets assume that 70% of pages in A and 30% of pages in B have changed since the last crawl of these two sites. Also, in this scenario we can only download 100 pages in each crawl cycle. Table 1 shows the number of pages from each site that change in each of the five categories. For example, in category C_2 75% of pages have changed. In this case, 20 of the changed pages are from site A and 10 from site B.

Changes	C_1	C_2	C_3	C_4	C_5
Site A	20	20	20	10	0
Site B	20	10	0	0	0

Table 1: Changes in each of five categories

2.3.1 *Sampling without Correlation*

Using the method described in [6], we sample 10 pages from each site and discover that 0.7 of pages in A have changed and 0.3 in B. As a result, all remaining resources are devoted to site A and an additional 80 pages are downloaded from site A. The change ratio therefore becomes (90 * 0.7 + 10 * 0.3) / 100 = 0.66.

2.3.2 Sampling with Correlation

Using our new method, we first sample four pages per category for a total of 20 pages for sampling. After this step, we conclude that all pages in C_1, 75% of pages in C_2, 50% of pages in C_3 and 25% of pages in C_4 have changed. We have now 80 pages left that can be used to download additional pages. Each category has 36 pages that have not been downloaded yet. Applying the greedy policy again, we commit 36 pages to C_1 and C_2 each and another 8 pages to C_3.

The change ratio we obtain is the following: From the initial sampling we obtain (4*1 + 4*0.75 + 4*0.5 + 4*0.25) = 10 changed pages and from the crawl we obtain (36*1 + 36*0.75 + 8*0.5) = 67 changed pages. Therefore, the change ratio for this method is 77 / 100 = 0.77, which is significantly higher than the method above.

While this example presents a small and simplistic scenario, we believe that it shows the potential of using correlation with sampling.

3. SAMPLING WITH CORRELATION

In this section we discuss the sampling-based download policy that takes Web page correlation into consideration. We will first elaborate how Web page correlation is established and then describe the sampling process.

3.1 Establishing Web page Correlation

The first step in our refreshing policy is to establish the correlation between the data items in the repository. For this discussion, we assume that all data items are HTML pages. Given Web pages $w_1 ... w_k$, we would like to distribute these pages to the categories $C_1 ... C_n$ if page w_i has a reputation on a topic described by category C_j. In particular, we would like this categorization to occur at download time of the page or shortly after.

3.1.1 Category assignment at download time

When pages on the Web are initially crawled, millions of pages are downloaded and it is necessary to assign each page to one or more categories that best describe that page. For example, if we download a page on the functionality of hard drives, we would like to assign it to the *computer hardware*, *hard drives* and possibly *computer* categories.

In order to achieve this task, it is necessary to know the *topics* a page has a reputation on given the URL of that page. This problem has been investigated extensively [1, 8, 12] and some solutions have been proposed. Most of the recent works go beyond simply looking for frequent terms on a page and returning those as the topics. After the success of PageRank [2], most researchers have realized the importance that lies within incoming links (*in-links*) and as a result, started to exploit them in categorization as well. The idea behind this is that a page might not necessarily mention the terms it concentrates on as part of its HTML content. Hyperlinks on the other hand contain more reliable semantic information regarding a page.

The direction that we chose for our work is the one described in [12] as part of the *TOPIC* project. In this work, given the URL of a Web page p, the system returns the top k topics that p has a reputation on. The basic idea of *TOPIC* is as follows: Instead of just considering page p, the algorithm also considers the neighborhood of p. A page q is in the neighborhood of p, if either q links to p or if p links to q.

The work in [12] introduces two methods for calculating the reputation rank of a page and we will describe one here. The interested user can refer to [12] for a detailed description of both methods. To calculate the reputation rank of a page, the reputation of all pages pointing to it are considered and weighted differently. The reputation of a page on a topic is proportional to the sum of the reputation weights of pages pointing to it on the same topic. This means that links emanating from pages with high reputations are weighted more. This method is a generalization of the PageRank method. Algorithm 1 describes how the reputation rank is calculated.

Algorithm 1 Algorithm for TOPIC classification

```
 1: for For every page $p$ and term $t$ do
 2:     if $t$ appears in $p$ then
 3:         Initialize $R(p,t) = 1/N_t$
 4:     else
 5:         $R(p,t) = 0$
 6:     end if
 7: end for
 8: while $R$ has not converged do
 9:     Set $R'(p,t) = 0$
10:     for Every link $q \rightarrow p$, do
11:         $R'(p,t) = R'(p,t) + R(q,t)/O(q)$
12:     end for
13:     $R(p,t) = (1-d)R'(p,t)$ for every page $p$ and term $t$
14:     $R(p,t) = R(p,t) + d/N_t$ if term $t$ appears in page $p$
15: end while
```

In this algorithm, d is the probability with which a random surfer would jump into a page uniformly chosen at random; N_t denotes the total number of pages on the Web that contain term t; $O(q)$ denotes the number of outgoing links from page q; $R(p,t)$ denotes the probability that surfer visits page p for topic t and $q \rightarrow p$ denotes a link from page q to page p. In this calculation, the ranks are in the form of a sparse matrix R. In this matrix, rows represent Web pages and columns denote each term that appears in some document. During the computation, R is first initialized and then updated until convergence.

Our algorithm for classifying the pages uses the algorithm above. The idea behind our algorithm is that for each page downloaded we obtain the top k topics that a page has a reputation on. If those topics already exist, we classify that page under those categories, otherwise, we create any nonexistent categories and classify the page under them.

Algorithm 2 presents our algorithm for crawling and categorizing Web pages. In this algorithm, Q is the queue for holding the URLs to crawl, $topic(u)$ represents the invocation of a method to obtain the topics the page at URL u has a reputation on and T represents the collection of topics u has a reputation on. The output of the algorithm are categories $C_1 ... C_n$ each holding a collection of crawled Web pages. It is clear from the algorithm that a page could belong to several categories.

Algorithm 2 Algorithm for crawling the Web and categorizing downloaded items

```
1:  Start with a staring URL $d$
2:  Insert $d$ into $Q$
3:  while $Q$ not empty do
4:      $u \leftarrow Q.dequeue$
5:      download the page with URL $u$
6:      Extract all links $l$ from $u$ and enqueue on $Q$
7:      $T = topic(u)$
8:      for Each topic $t$ in $T$ do
9:          if $t$ not already in the system then
10:             Add $t$ to system
11:             Classify $u$ to $t$
12:         end if
13:     end for
14: end while
```

One issue that arises from this algorithm is the number of topics a page p should be classified under (i.e. the number of topics that *topic*(u) returns). In the ideal case, each page should be assigned to exactly one category that best describes the page. In order to achieve this, however, the categorization process must return the one *exact* topic a page has a reputation on. None of the classifying approaches, however, claim to be able to be this precise. It is therefore necessary to classify a page to several categories in order to increase the probability that it is assigned to the best topics that describe it.

Once this approach is chosen, the optimal number of topics p should be assigned to must be determined. A theoretical analysis of this problem is beyond this work and is left as future work. Section 5 provides some experimental insight into this issue.

It is important to note at this point that any other categorization method could be used for our approach. We just chose one to be able to demonstrate the advantages of taking correlation of pages into consideration when performing sampling. It would be interesting to investigate how different categorization techniques could affect our sampling-based approach.

4. SAMPLING

Once the correlation between data items are extracted, the second step in our Web page refreshing policy is to perform sampling in order to estimate the number of changed pages in each group of correlated pages, and download the Web pages according to the estimated change ratios.

The main idea behind the sampling is the following: The number of Web pages kept by search engine is too large to perform a download in each Web repository refresh cycle, therefore, using the portion of the pages, we try to estimate the ratio of the changed pages to total number of pages in each category (henceforth, change fraction), and draw a conclusion based on the fraction of the changed pages.

For the rest of the discussion, let's assume our page repository $\Re$ contains the following Web page categories, formed using a correlation criteria ρ, as described in the previous section:

$$\Re_\rho = \{C_1, C_2, .., C_n\} \tag{6}$$

The categories of the Web pages are not necessarily mutually exclusive (This has a impact on the sampling size and result, which will be discussed later in subsection 7.1). That is, a page p may belong to different categories C (e.g. The same Web page be classified bother under "sports" and "basketball").

In order to achieve our purpose, we are going to consider uniform random sampling, where the probability of any Web page in category C being in the sample S is $\frac{1}{\|C\|}$, where $\|C\|$ denotes the cardinality of C. In other words, any variable in the uniform random sample is identically distributed and independent. Once we perform random sampling, we will estimate the number of real value of changed page fraction p of each category, based on the fraction of the changed pages in the sample to the total number of pages in the sample (a.k.a sample size).

In the following subsections, we will first describe what we think the optimal sample size should be, and then we will discuss how to perform uniform random sampling, and estimate changed pages fraction. We are also going to talk about other factors that have an impact on the optimal sampling size, and how this impact can be augmented into the optimal sample size formula. At the end, the deficiency of the uniform sampling will be briefly discussed.

4.1 Optimal Sample Size

We are limited with the download resources of search engine used for a page-refresh, when deciding on the sampling size. As the number of download resources D (i.e. number of pages that a search engine can download in a refresh cycle) increase, more resources can be used for sampling.

Another factor affecting sample size is the number of pages in each category. We have to sample more for the categories that have more pages in order to obtain a meaningful estimation of the ratio of changed pages in the category C. For simplicity, when calculating sample size, we will assume each category has approximately same number pages, N. Handling categories with different sizes will be discussed in subsection 7.1.

The download policy has also an impact on the sample size. As the greedy download policy is shown to give better *ChangeRatio* on average than the proportional policy in the [6], we will consider only the greedy download policy in this section.

Let's assume that we can download a maximum of D pages in each page-refresh cycle, out of a total of N average number of pages in each category. We will use the following formula given in [6] to get the optimal sample size for experiments that attempt to gauge the affect of other factors than the sample size (e.g. comparison between page refresh policies, optimal number of categories):

$$s \approx \sqrt{\frac{Nrf(p_t)}{6(\bar{p_r} - \bar{p})}}, \tag{7}$$

p_t refers to a change fraction threshold value, such that the number of categories that has a higher value than the threshold, will be all downloaded. $f(p_t)$ represents the function that returns the number of categories that have the changed page fraction value of p_t, $\bar{p_r}$ and $\bar{p}$ represent the average change fraction across all the categories, and the average change fraction across the categories that exceed the threshold change fraction, respectively.

The intuition behind including $f(p_t)$ in the optimal sample size formula is that as the number of categories that have the threshold change fraction value increase, we are more likely to make errors as the threshold is used to decide whether the category should be downloaded in the refresh cycle. A very small enhancement in the estimation in the categories, which can happen with a bigger sample size, will enable us to decide whether a category is lower or higher than the threshold.

Also, the higher the value of $\bar{p_r} - \bar{p}$ the more confidence we have in the estimation result. In other words, the sample size is reversely proportional to $\bar{p_r} - \bar{p}$.

As we do not know the p or $f(p)$ values, we will use $\sqrt{Nr}$ as a rough estimate to the sample size, similar to [6]. In section 5, we will investigate this issue experimentally.

4.2 Uniform Random sampling

In order to perform random sampling, we number each page in each category with an integer sequentially from 1 to n, and then, randomly pick a number between 1,n. The algorithm is given in Algorithm 3, where S_C denotes the sample set of the pages in category C.

Algorithm 3 Algorithm for sampling categorized Web repository

```
    for every category $C_i$ in $\Re$ do
        $sample - size = k = \sqrt{Nr}$
 3:     for $j = 1tok$) do
            $r$ = Pick a random number between 1,$n$
            $u$ = Page with a index $r$ in $C_i$
 6:         if $u$ is not already in the $S_{C_i}$ then
                $S_{C_i} = S_{C_i} \bigcup u$
                j= j+1;
 9:         else
                j = j-1;
            end if
12:     end for
        $p = numberofchangedsampledpages/sample - size$
        Insert ($cr$, $C_i$) tuple into a sorted data structure on
            the first attribute
15: end for
```

The sampling part of the algorithm is also called classical sampling with membership check [Devroye86]. Once a sample S_i is constructed for each category C, we calculate the change fraction p in C_i based on the number of pages changed in the given sample. A "changed" page refers to the page whose current downloaded version is different from the version in the original Web page repository $\Re$.

Based on the changed fraction p_i of each category C_i, the greedy policy will download the categories with the highest p_i until there is no download resource left.

The discussion so far assumes that each page has the same weight in the estimated changed fraction p. However, some pages in the correlation group may have more weight than other pages in the estimated result. For example, the World news pages of the CNN is more indicative of any changes in the World news page group, than a local town's news Web site. In this case, we have to perform weighted sampling where the "importance" of a page in the sample has to be considered. Finding authoritative pages and performing weighted sampling is considered as a future work.

5. EXPERIMENTS

In this section we will present some experimental results from testing our method and propose some tests that could be run in the future. Due to time restrictions, we were not able to fully implement the system described above. In particular, the algorithm for classifying Web pages had to be simplified to be able to finish this work within the required time frame.

The major difficulty to implement a full system was to be able to handle *all* possible terms that could appear in Web pages. Also, since the authors of *TOPIC* were not able to provide us with an already working implementation, we were forced to take a simpler approach in which we only considered a subset of terms that could appear in pages. In particular, we concentrated on 20 terms related to the *Internet*. It is important to note that this simplification does not make the system less complex, but rather reduces the number of terms the system has to "know" and as a result, the number of pages that have to be downloaded.

In order to run meaningful experiments for crawling and repository freshness, it is important to do three things: one, we must be able to download pages at an interval that is meaningful and represents a time frame in which data items change. Secondly, the experiments must be run over several download cycles to obtain meaningful results. Finally, the size of the repository must be large enough so that the results can be applied to the Web.

Unfortunately, we were not able to meet the requirements above. First, due to resource restrictions, we were only able to construct a repository with 1000 pages and 20 categories. Each category contained 50 pages (in order to ensure that all our categories have a meaningful size, we downloaded more than 1000 pages). Secondly, we were able to two download cycles of our experiment that were two weeks apart. The first cycle was to build the repository and the second to check for freshness and download changed pages. In our experiments we assumed that at each cycle we can download 200 pages.

As a result of the restrictions described above, many experiments do not show the desired results. It is, however, not possible to determine whether these results are due to

Table 2: The time to create categories with changing number of in-links.

	10	20	50	100	300
Time (s)	450	455	463	472	479

problems with our method or the fact that more through experiments had to be run.

All our experiments were run on an Intel P4 2.4 GHz computer with 512MB of DDR memory. The 1000 Web pages were stored on a hard drive with 9ms disk access. In order to obtain all pages that link to a particular page (needed for the *topic* algorithm) the Google API was used. Since this API is implemented in Java, all our experiments are run using Java. The source code for this work can be obtained at http://www.cs.uwaterloo.ca/∼ataleghani.

5.1 Categorization

The main two factors that affect the speed of the categorization are the number of "topics" that pages can belong to and the number of in-links that are considered for each page. In our system, we were already forced to reduce the number of terms available for categorization, as a result, we performed experiments on the number of in-links considered for each page.

Table 2 shows the result of categorizing 1000 Web pages using 50 categories (topics). In order to not confuse download time with categorization time, the time does not include the download time for a page. This experiment was conducted on 1000 URLs and the result was a categorization for each of these pages.

Table 2 shows that there is little difference between the number of in-links considered. These results can be explained as follows. First, the computational overhead to consider additional links in-links is negligible because the results from Google (which we use to obtain in-links) are returned within very few seconds, no matter the number of results. Second, we observed that as the number of links increase, there are not many pages that have that many in-links. For example, we observed that only 18% of our 1000 pages had more than 50 in-links so that as we increase the number of in-links, the performance does not change.

It is important to note that the absolute time values of the results of the experiment outlined in Table 2 are not crucial for our results. In our implementation, we had to make use of an external method (Google) to obtain all in-links to a particular page. This added a large overhead to the system. In reality, however, a crawler would take advantage of internal *link indexes* that are maintained by the search engine. As a result, the time to categorize pages would decrease dramatically.

5.2 Comparison of download policies

In this section, we compare our method with round-robin and sampling without correlation. We do not compare our method with the frequency-based policy because we did not perform enough download cycles to collect relevant change history.

Table 3: Comparison of sampling-based policy with correlation and round-robin. The result for sampling-based with no correlation was from another experiment.

Policy	*ChangeRatio*
Round-Robin	0.45
Sampling with Correlation	0.6
Sampling without Correlation	0.75

Table 3 shows the results of our experiments. In this experiment, the *ChangeRatio* was calculated after the second cycle of downloads. The *ChangeRatio* for round-robin and sampling-based policy with correlation could be determined from our experiments. That of sampling without correlation though could not be determined from the experiments.

The difficulty in this experiment was that from our 1000 page repository, we had obtained 20 categories, but these pages belonged to more than 20 different sites (in fact, they belonged to more than 300 different sites). As a result, an accurate comparison between sampling without and with correlation was not possible. The value reported for sampling without correlation is the largest value that was obtained from experiments in []. We realize that this comparison is not accurate and further experiments are necessary. In particular, an experiment is necessary in which the pages that are distributed between categories are from the same number of different sites as there are categories. This experiment is left for future work.

Even though the results do not provide us with a definite comparison between the two sampling based approaches, they do indicate that, at least after running one cycle, our approach performs better than round-robin. It remains to investigate more results after running more thorough experiments.

5.3 Optimal sample size

In a sampling-based policy the size of the sample affects the performance significantly. A small sample could skew the results in a way that the estimated change in the sample is different than the real change in the repository. A too large sample has the disadvantage that too much of the download resources are devoted to sampling. Figure 1 illustrates this issue in more detail. This graph compares the optimal download sample for the sampling based policy with correlation.

As can be seen in Figure 1, from a sample size of [0,50] the *ChangeRatio* is very low. The reason behind this is that no useful conclusions can be deduced from the sample size and the categories are updated more or less randomly. As the sample size increases, more and more resources are devoted to sampling and the advantage of our approach increases. From the graph, the optimal sample size is between 70 and 90. As the number of samples increases, the performance decreases again. The reason behind this is that resources are wasted for sampling that could be used for downloading pages that have changed in categories. These results support the theoretical results of section 4.

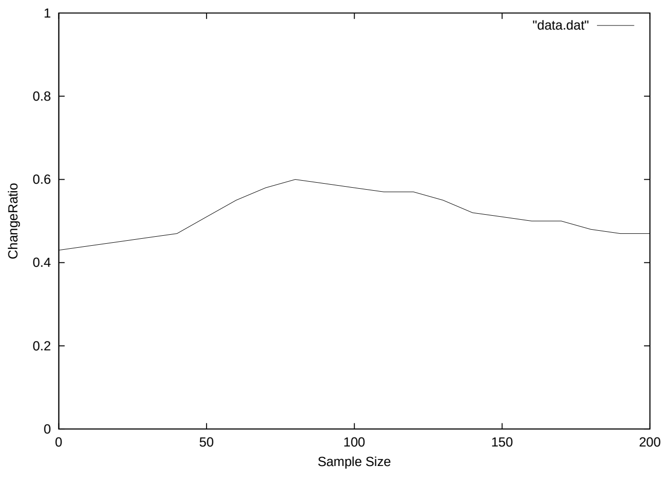

Figure 1: Relationship between sample size and *ChangeRatio*

5.4 Optimal number of categories

As described above, one question that arises during categorization is the number of categories a particular page should be assigned to. For example, a Web page describing the differences between various computer microprocessors could be assigned to "computer hardware" and "microprocessors". But it could also be additionally assigned to "computer" and "electronics". As a result, we have to determine the optimal number of categories a page should be assigned to.

Our method for obtaining the topics a page has a reputation on returns in general more than 10 topics for each page. In this section we describe the experiment we conducted to obtain the optimal number of categories. Figure 2 shows the results of changing the number of categories a page is assigned to.

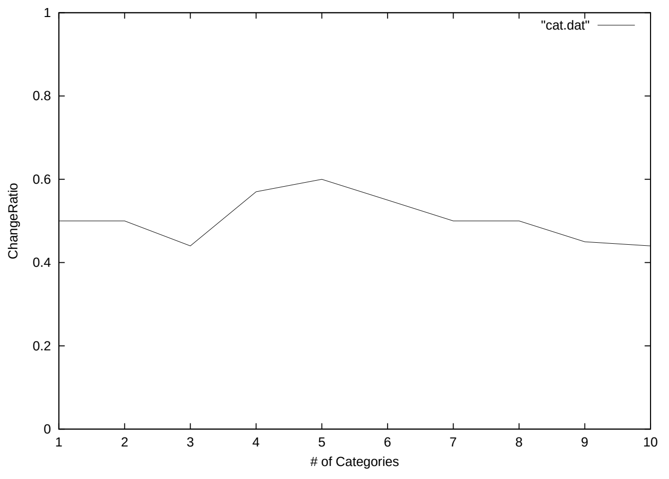

Figure 2: Relationship between number of categories and *ChangeRatio*

As Table 2 shows, the performance of the sampling-based approach with correlation changes with different number of categories. According to the experiment, the optimal number of categories a page should be assigned to is 6. This issue is further described in section 7.1.

6. RELATED WORK

There is a considerable amount of work in the Web crawling area. The work related to this paper falls under the subject of Web repository refreshing techniques within Web crawling. The problem can also be formulated as the problem of keeping any local copy of data (or replica) up-to-date, where the original data resource is updated without the knowledge of the replica. The same techniques can be used for Web page refreshing, however there are additional challenges in Web, such as huge number of Web pages over the Web.

The authors in [4], and [5] study how to estimate how often the original data item changes and refresh a repository based on the past change history of the item. Using the estimation on how often the Web pages change, these papers develop techniques to improve the freshness of the local repository. The problem with this type of approaches is that they require the collection of a long chain of history of change for every single Web page. Otherwise, they may lead to incorrect estimates, and as a result, poor quality of the local repository (in terms of freshness). Also, the statistical function used in these approaches, such as Poisson distribution, assumes that all the changes are independent, and the average rate of change does not change over the period of time. Our approach, however, does not require any kind of history regarding to the changes. Therefore, it can be used immediately after initial Web crawling. Also, one of the main contribution of our approach is the observation that update events are correlated, and we make no assumption regarding the average rate of change, which seems to be more applicable to the Web.

The authors in [6] studies how to solve the biggest drawback of the change-frequency based approaches, i.e. the requirement of gathering a long history of change frequency of each page. When there is not enough history chain to be used for change-frequency based approaches, this paper proposes to perform sampling in order to estimate the change frequency of the Web pages, and download the pages accordingly (Once enough history log is gathered, the system can switch to the change-frequency techniques). In this sense, this approach seems very similar to ours, however, there is a significant difference: [6] performs sampling on each Web site, and uses this information as a indicator about the rest of the Web pages in that Web site, whereas our approach performs sampling for each group of pages categorized based on their correlation to each other, and use this information as a indicator about the rest of the Web pages in the same category.

The work in [11] tries to maximize the quality of user experience who query the search engine, rather than the quality of every single page in the repository. This work studies how to estimate the improvement in the repository if a particular Web page were to be downloaded, without actually downloading it. This estimation requires the Web search engine to keep track of user query logs. This approach is also biased towards the popular Web pages. If the information that a user is looking for is not located in these popular pages, the user experience will degrade drastically. In our approach, we neither require Web search engine to use their precious resources to keep historical information, nor do we favor one page over another.

7. CONCLUSION

As more and more Web users take advantage of search engines to gain access to the information on the Web, more effort has to be put in the process of returning the most relevant pages for a given query. One major difficulty is the fast changing rate of the Web. Pages that were crawled and saved in the repository might become out-dated very fast and therefore, the results returned to the user are out-of-date.

In this paper, we described a new technique of increasing repository freshness. Our work was built on the ideas of the work in [6] which suggested sampling. In our work, however, we consider the correlation of Web pages as an important factor in categorizing and sampling. Our approach first categorizes Web pages into a set of categories that best describe them and consequently takes a certain number of samples from each category. Based on the *ChangeRatio* of the samples, a decision is made as to whether update the whole category.

In this work we described the method we used for categorizing Web pages. As mentioned above, any other form of categorization could have been used and it remains to investigate how different categorization techniques perform with our sampling approach.

We also described the sampling method used in our refresh policy. The most important factor that has to be determined during sampling is the optimal sample size. As the sample size increases, our estimate of change improves, but at the same time our download resources decrease. We provided some theoretical analysis on this issue with was based on the work in [6], but it remains to further investigate this issue.

Finally, we described some experiments that we conducted to compare our method with others. As described in section 5, we were not able to conduct all the necessary experiments and as a result, our result might not be indicative of the real performance of our method. It remains to run more thorough experiments in the future to deduce the full potential of the sampling-based refresh policy with correlation.

7.1 Future Work

In the following sections we provide some future direction of our work.

7.1.1 Pitfalls of Uniform Random Sampling

A significant error might be introduced in uniform sampling if the change ratio is skewed among the pages in the same group, and the sample size is small. The skew occurs when there is a large variance between the change frequency of the pages in the same group. For example, out of 100 pages in C_i, 5 pages might change daily, but the rest might change every month. If our sample size is small, e.g. 2, and one of the sampled pages is one of these 5 pages then our sampling will estimate a change ratio of 0.5. But in reality, the change ratio of this group is 0.05 (assuming that 95 pages did not change).

This issue brings up the possibility to combine our method with the frequency-based policy. If we obtain the history of change for a category, we might be able to eliminate this problem by having an estimate on the change of a category.

7.1.2 Improving Optimal Sample Size

In the section4.1, we assumed that each category has approximately the same size, and we used the optimal sample size theorem from [6]. However, this assumption may not be true in practice. We need to factor into our formula possible differences in category sizes. To achieve this, we multiply the optimal sample size s for each category with the ratio of category size to the total number of pages in $\Re$.

$$s_{C_i} \approx s * \frac{n_c \|C_i\|}{\|\Re\|}, \tag{8}$$

where n_c represents the number of categories in page repository $\Re$, $\|C_i\|$ represents the number of pages in the category C_i, and $\|\Re\|$ represents the total number of pages in the given page repository.

This way, we can fairly share the total number of samples among the different sized Web page categories, in an attempt to reduce the number of errors in the estimated changed fraction rate.

Another aspect we need to consider when calculating optimal sample size is that categories are not mutually exclusive. In other words, multiple categories can include the same Web page. We expect the refresh policy to degrade as the number of pages shared among different categories increases because our estimate for the optimal sample size does not consider the number of pages shared between the categories. One improvement to alleviate this issue is to use $\frac{n_c \|C_i\|}{Total size of C}$ in the sample size, where we employ the ratio of size of category to the total size of all the categories. The reason we take the ratio of the size of a category over the total size of all categories is that the total number of pages in the repository is not equal to the total size of pages in all categories, as some pages are counted more than once.

Also, the sampling algorithm can be improved by not sampling the same page if it is already sampled in another category, and using the sample resource for another page. More precisely, in line 6 of Algorithm 3, we will add another if statement that checks whether the the selected sample page for the current category S_{C_i} is already located in the the samples created so far in the algorithm for the other categories. This addition is shown in Algorithm 4.

As a result, even though we are not going to use our sampling resource for this page, the page is going to contribute to the estimation of the change fraction at the end, and improve the estimation.

Algorithm 4 Improvement to Algorithm 3

```
if $u$ is already in any other $S_{C_j}$ created so far then
        Adjust the number of changed sampled pages
        Increase the sample-size by one
end if
```

7.1.3 Estimated vs. real change fraction

The deviation between the estimated change fraction and real change fraction has a direct impact on the quality of page freshness. If the deviation is big we might be downloading the C_i pages, while, in reality we should be downloading another category first. The deviation is affected by the sample size and the variation among the change frequency of pages in the group. The latter was discussed in section 7.1.1.

If we can define the deviation between the estimated change ratio, and real change ratio, we can make much smarter choices about which C_i to download. This can be achieved through calculating a confidence interval, for a given confidence and level:

$$l \leq p \leq h, \tag{9}$$

with the confidence level of ε.

In this equation, $[l, h]$ represents the confidence interval and ε represent the probability of p being within the $[l, h]$. For example, if we say our estimate change fraction rate is 0.5 ± 0.2 with %95 confidence level, it means that the real changed fraction is within $[0.3, 0.7]$ boundary with the 0.95 probability. This definition would enable us to compare two estimated fraction rate more accurately.

8. REFERENCES

[1] K. Bharat and M. R. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of SIGIR-98*, pages 104–111, Melbourne, AU, 1998.

[2] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[3] C. Castillo, M. Marin, A. Rodriguez, and R. Baeza-Yates. Scheduling algorithms for web crawling. In *Proceedings of Latin American Web Conference (WebMedia/LA-WEB)*, Riberao Preto, Brazil, 2004. IEEE CS Press. To appear.

[4] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 117–128, 2000.

[5] J. Cho and H. Garcia-Molina. Estimating frequency of change. *ACM Trans. Inter. Tech.*, 3(3):256–290, 2003.

[6] J. Cho and A. Ntoulas. Effective change detection using sampling. In *VLDB*, pages 514–525, 2002.

[7] E. G. Coffman, Z. Liu, and R. R. Weber. Optimal robot scheduling for web search engines. Technical Report RR-3317, 1997.

[8] J. Dean and M. R. Henzinger. Finding related pages in the World Wide Web. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1467–1479, 1999.

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

[10] C. Olston and J. Widom. Best-effort cache synchronization with source cooperation. In *SIGMOD Conference*, 2002.

[11] S. Pandey and C. Olston. User-centric web crawling. In *WWW '05: To appear in the fourteenth international conference on World Wide Web*, 2005.

[12] D. Rafiei and A. O. Mendelzon. What is this Page Known for? Computing Web Page Reputations. *Computer Networks*, 33(1-6):823–835, 2000.