

# A Clustering-Based Sampling Approach for Refreshing Search Engine's Database

Qingzhao Tan<sup>†</sup>      Ziming Zhuang<sup>‡</sup>      Prasenjit Mitra<sup>†‡</sup>      C. Lee Giles<sup>†‡</sup>  
qtan@cse.psu.edu    zzhuang@ist.psu.edu    pmitra@ist.psu.edu    giles@ist.psu.edu

<sup>†</sup>Computer Science and Engineering    <sup>‡</sup>Information Sciences and Technology  
The Pennsylvania State University, University Park, PA 16802, USA

## ABSTRACT

Due to resource constraints, search engines usually have difficulties keeping the local database completely synchronized with the Web. To detect as many changes as possible, the crawler used by a search engine should be able to predict the change behavior of webpages so that it can use the limited resource to download those webpages that are most likely to change. Towards this goal, we propose using sampling approach at the level of a cluster. We first group all the local webpages into different clusters such that each cluster contains webpages with similar change patterns. We then sample webpages from each cluster to estimate the change frequency of all the webpages in that cluster, and the cluster containing webpages with higher change frequency will be revisited more often by our crawler. We run extensive experiments on a real Web data set of about 300,000 distinct URLs distributed among 210 websites. The results show that by applying our clustering algorithm, pages with similar change patterns are effectively clustered together. Our proposal significantly outperforms the comparators by improving the average freshness of the local database.

## Keywords

Refresh policy, sampling, clustering, database, Web crawler, search engine.

## 1. INTRODUCTION

Search engines rely on crawlers to harvest webpages and store them in their local databases, and the local copies are later retrieved to answer relevant user queries. Although ideally the search engines may synchronize these local copies with their online counterparts, due to resource constraints it is not feasible for crawlers to constantly monitor and download every single webpage. Therefore a round-robin polling strategy will not guarantee an effective and efficient performance. To be more efficient, crawlers can periodically re-visit based on certain strategies a portion of the webpages that are more likely to have changed since the last visit.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Copyright is held by the owner/author  
*Proceedings of the 10th International Workshop on Web and Databases (WebDB 2007), June 15, 2007, Beijing, China*  
Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

We investigate a typical scenario in which due to resource constraints, a crawler is only allowed to periodically download a fixed number of webpages and update the corresponding local copies when a change has been found. We define this fixed number of pages as the *download resources* and the periodical interval as the *download cycle*. Thus, the crawler's goal is, given the *download resources*, to *maximize* the number of updated webpages downloaded in each *download cycle*. Because a crawler can verify whether a webpage has changed only *after* it downloads the page, the challenge therefore becomes how to predict, as accurately as possible, the probabilities of change for the webpages in the local repository. Those webpages with higher change probabilities will be assigned a higher priority for download.

To address the above challenge, Cho and Ntoulas proposed a sampling-based algorithm that is widely adopted [9]. In their approach, a small number of webpages are first downloaded from each website as samples, then these samples are used to decide which websites contain more changed webpages so that more resources are assigned to those websites. More specifically, their sampling is at the level of a website with a greedy sampling policy, in which a few webpages from each website are chosen as samples and all the webpages in the website with the largest number of changed samples are re-downloaded. We argue that the level of a website may not be a good granularity for sampling because the update patterns of various webpages on the same website could be quite different, while webpages with similar update patterns may distribute across different websites [12]. We propose using clustering techniques to group webpages with similar update patterns together and use the cluster as the downloading granularity for the sampling approach. Our major contributions include:

- We investigate the feature space for clustering webpages according on their likelihood of change, by studying not only the static features (e.g. size of a webpage), but also the dynamic features (e.g. the degree of the observed changes). By partitioning the feature vector into different categories, we also study which set of features are the most effective in change prediction.
- We propose a new sampling approach which is at the level of a cluster. A sufficient set of webpages are chosen from each cluster as samples, based upon which the change patterns of the webpages in the whole cluster can be predicted.
- We compare four different settings, non-adaptive, short-sighted-adaptive, arithmetically-adaptive, and geometrically-adaptive policy, to compute the change frequency of the samples. We conclude that the adaptive

policies’ prediction is more accurate than the non-adaptive one.

The rest of the paper is organized as follows. In Section 2 we describe some existing work related to our approach. We propose our clustering process and the features used for clustering in Section 3. In Section 4 we investigate the sampling process on the cluster level. In Section 5 we present and discuss the empirical results. We outline our conclusion in Section 6.

## 2. RELATED WORK

Existing studies have shown the dynamics of the Web pages that covers a broad spectrum of change rate varying from a few hours, a few weeks, to a year [4, 6, 16]. In order for search engines to keep up with the dynamic Web, server-side approaches require that the Web servers keep a file with a list of URLs and their respective modification dates [3]. This approach is very efficient but requires modifications to the server-side implementation.

On the other hand, client-side techniques are independent of server-side implementation. First, probabilistic models have been proposed to approximate the observed update history of a webpage and to predict its change in the future [7, 11]. Most of these change-frequency-based refresh policies assume that the webpages are modified by *Poisson processes* [13]. Besides page’s change frequency, several metrics have been proposed to guide the crawler how to choose webpages to re-download, such as *freshness* and *age* [7], “*embarrassment*” metric [22] and user-centric metric [18]. However, a limitation common to all these approaches is that they need to gather enough accurate historical information of each webpage. To address this problem, sampling-based approaches are proposed to detect webpage changes by analyzing the update patterns. Their sampling method samples and downloads data in the unit of a website. Different from their approaches, we analyze whether sampling in the unit of a cluster, which is based on some change related features, can be more effective.

Recently, a classification-based algorithm [2] takes into account both the history and the content of pages to predict their behavior. Their experiment results on real web data indicate that their solution had better performance than the change-frequency-based policy [6]. The key difference between their algorithm and ours is that they assume the crawler knows the change history of a set of webpages. And these webpages are used as training data to learn the change pattern of other new pages. While in our approach, we do not assume the crawler has any existing historical information. Therefore, our approach is more applicable than the classification-based method.

## 3. CLUSTERING WEBPAGES USING FEATURES RELATED TO CHANGE PATTERNS

We cast the problem of finding webpages that have similar change patterns into a clustering problem, with the assumption that webpages have specific features related to their update behaviors. In this section we discuss how to cluster all the webpages in the local database. The first step of the clustering process is to extract two types of intrinsic features (static and dynamic) that are correlated with the webpages’ change patterns.

### 3.1 Learning Static Features

Recent studies have found that some characteristics of webpages are strongly correlated with their change frequencies. For example, Douglis et al. [10] noted that actively-changing webpages are often larger in size and have more images. Fetterly et al. [12] observed that the top-level domains of the webpages are correlated with their change patterns. Based on the previous findings and our observations, the following static features are used in our clustering process.

**Content features** There are 17 features in total. First, we construct a word-level vector to represent the content of the webpage, which is used as one feature vector for clustering. To avoid its dominance over other features, we first cluster all the webpages based on the *tf · idf* [20] vector of all the words. The IDs of the 10 largest clusters are chosen as the labels of 10 dimensions in the feature vector. Then for each webpage  $p$ , the values of these 10 dimensions are computed as follows. Suppose the webpage belongs to the cluster  $C_p$ , the feature’s cluster label is  $C_f$ . Using the distance of the centroids of each cluster to represent the distance of two clusters, the value of the feature  $f_{C_f}$  is computed as follows.  $f_{C_f} = 0$  for the webpages in  $C_p = C_f$ ,  $f_{C_f} = 1$  for the webpages in  $C_p$  which has the shortest distance to  $C_f$ ,  $f_{C_f} = 2$  for the webpages in  $C_p$  which has the second shortest distance to  $C_f$ , and so forth. To speed up the computation, we use the 1000 most frequently appearing words in all the webpages.

In addition to the 10 content-related features, we consider seven other features: number of images, number of tables, number of non-markup words, file size in bytes (excluding the HTML tags), and file types: whether the file is an HTML file, a TXT file, or others (i.e. each file type is represented as one dimension in the feature vector and they are all in binary form, e.g. for an HTML file the corresponding dimension has a value of 1; otherwise, 0).

**URL features** There are 17 features in total. First, we process the words in the URL as what we do for the words on the webpage in order to keep the final feature vector at a reasonable size. Second, we compute the depth of the webpage in its domain, i.e. the number of “/” in the URL. We also consider the name of the top-level domain in the URL, and reserve six dimensions in the feature vector for the domains .com, .edu, .gov, .org, .net., and .mil. Each of these six dimensions is a binary feature indicating whether the webpage belongs to the corresponding domain or not.

**Linkage features** There are four features in total. We use PageRank [17] of the webpage as an indicator for the *popularity* of a webpage. The assumption is that the more popular a webpage is, the more up-to-date it needs to be kept. PageRank is a probability distribution to indicate the likelihood that a person randomly traversing the Web will eventually arrive at any particular page. The PageRank of a webpage,  $p_i$ , is calculated as follows:

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in IL(p_i)} \frac{PR(p_j)}{OL(p_j)}. \quad (1)$$

where  $p_1, p_2, \dots, p_N$  are the webpages,  $IL(p_i)$  is the set of pages that link to  $p_i$ ,  $OL(p_j)$  is the number of out-going links on page  $p_j$ ,  $N$  is the total number of webpages in the local repository, and  $d$  is a damping factor. PageRank has been proven in previous studies [8] as an excellent ordering metric when crawling for pages with many in-links. We also consider three other features related to linkage: number of

incoming links within the local repository, number of outgoing links, and number of email addresses on the webpage.

### 3.2 Learning Dynamic Features

Ali and Williams have discussed in [1] that the significance of the past changes in document content is an effective measure for estimating whether the document will change again and should be re-crawled. We extend this idea to some of the aforementioned static features as follows. Note that it is impossible to calculate these dynamic features until after the second download cycle, due to the fact that they are based on the comparison between two consecutive snapshots of a webpage.

**Dynamic Content Features** We use the following five features: Change in the webpage content, which is computed by the cosine similarity between the webpages’ content in two consecutive download cycles; change in the number of images; change in the number of tables; change in the file size (in bytes, without the HTML tags); and change in the number of non-markup words on the webpage.

**Dynamic Linkage Features** We use the following four features: Change in PageRank, change in the number of incoming links within the local repository; change in the number of outgoing links, and change in the number of email addresses on the webpage.

### 3.3 Clustering Webpages

With each webpage represented by a feature vector, we apply the Repeated Bisection Clustering (RBC) algorithm [14] to construct hierarchical clusters. With the RBC algorithm, a  $k$ -way clustering solution is obtained via a sequence of cluster bisections.

An important question that needs to be answered before applying the clustering algorithm is how many clusters there are in the data set. Since clustering analysis is an unsupervised learning technique, we do not have prior knowledge about the number of clusters. However, we can estimate this number by using the method of *v-fold cross-validation*. The general idea of the *v-fold cross-validation* is to divide the overall data set into a number of  $v$  folds. The clustering process is then successively applied to the data belonging to the  $v - 1$  folds (training samples) to get  $k$  clusters. For the  $v$ th fold (test sample), the centroids of these  $k$  clusters are applied to them and the data in the  $v$ th fold are put into  $k$  different clusters. A single measure is used to evaluate the quality of the clustering results in the  $v$ th fold. This measure is based on the goal of clustering, which is to minimize the similarity of data within the same cluster while maximize the similarity of data between different clusters.

In general, we try a range of numbers,  $ks$ , in clustering. For each  $k$ , we apply the *v-fold cross-validation* method and observe a score function used in the clustering process. Specifically, we focus on the criterion function used by most vector-space based clustering algorithm. Let the  $k$  clusters denote as  $S_r$ ,  $r \in [1, k]$ , and their centroids denote as  $C_r$ . If we use the cosine function to measure the similarity between a webpage  $p_i$  and a centroid  $C_r$ , the criterion function becomes the following:

$$\tau = \sum_{r=1}^k \sum_{p_i \in S_r} \cos(p_i, C_r). \quad (2)$$

where  $\cos(p_i, C_r)$  is obtained as

$$\cos(p_i, C_r) = \frac{p_i^T C_r}{\|p_i\| \|C_r\|}. \quad (3)$$

To maximize (2) is to maximize the similarity between each webpage and the centroid of the cluster that is assigned to. Generally speaking, the larger  $k$  is, the higher  $\tau$  is. However, a very large  $k$  may impact the efficiency of the clustering algorithm. Thus we need to try different values of  $k$  and select the one that can balance both. The results of *v-fold cross-validation* are best reviewed in a simple line graph. The line showing the score function should first quickly increases as the number of clusters increases, but then levels off. As such, the optimal number of clusters can be found at the point switching from increasing to levelling off.

## 4. SAMPLING IN CLUSTER LEVEL FOR UPDATE DETECTION

We present in this section a sampling-based update detection algorithm that works at the cluster level, under the assumption that each cluster contains webpages with similar change patterns. However, as we have no prior knowledge about which cluster is more likely to change, we adopt a sampling approach to determine which cluster contains webpages with the highest change probability. The mechanism of this sampling approach is as follows. The crawler first chooses samples from each cluster instead of each website, and then checks whether the samples are up-to-date. Finally the crawler selects the clusters with high change probabilities to re-visit.

### 4.1 Sampling webpages

We first address two challenging questions in sampling webpages: which webpages in the cluster, and how many of them should be selected as samples.

To select *representative* webpages for the whole cluster as samples, those that are near the cluster’s *centroid* are the most suitable candidates. The distance between a webpage and the centroid can generally be represented using the cosine function given by (3). This measure ranges in  $[0, 1]$  and becomes larger when the webpage is more similar to the centroid.

A *sufficient* sample for a cluster is defined as the sample which produces a valid mean score for the cluster. This valid mean score should represent the population mean  $\mu$  within  $\mu \pm \delta$  at a confidence level (e.g. 80%) in a standard  $t$ -test. Here  $\mu$  is the average change frequency of the samples (explained later). The sample’s valid mean score is used to represent the mean score of the whole cluster. It’s possible that a sample of reasonable size is unable to produce a valid mean score. Therefore, we set an upper bound for the sample size in each cluster to avoid analyzing too many samples. In our setting, we set the upper bound of the sample size as the total number of download resources over the number of clusters.

### 4.2 Predicting change patterns

The sampled webpages are used to predict the change patterns of the cluster. For each sampled webpage we estimate the change frequency based on its update history. Then for each cluster, we compute its *download probability*  $\varphi$  as the average change frequency of all sampled pages in that cluster. The crawler can then download webpages from clusters

in the descending order of their download probabilities, until the download resources are exhausted.

One can study a sampled page’s change frequency based on its history [21]. The basic idea is that pages with many recent changes are more likely to change again. We also model the update of the webpages using a *Poisson process* [13] which has been shown effective in experiments with real webpages [5, 7]. The *Poisson process* is often used to model a sequence of events that happen randomly and independently at a fixed rate over time. In this process, the time to the next event is exponentially distributed. For our scenarios, it is reasonable to assume that the update of a webpage  $p$  follows a Poisson process with its own change rate  $\lambda_p$ . This means a webpage changes on its own, instead of depending on other pages. This assumption may not be strictly held but it has been proved to be the first workable approximation for real applications. Note that the change rate may differ from page to page. For each webpage  $p$ , let  $T$  denote the time when the next event occurs as a *Poisson process* with change rate  $\lambda_p$ . Then, we obtain the probability that  $p$  changes in the interval  $(o, t]$  by integrating the probability density function:

$$Pr\{T \leq t\} = \int_0^t f_p(t)dt = \int_0^t \lambda_p e^{-\lambda_p t} dt = 1 - e^{-\lambda_p t}.$$

We set the parameter *download probability*  $\varphi$  to be  $Pr\{T \leq t\}$  where  $t = 1$ , which means one download cycle. Therefore,

$$\varphi = Pr\{T \leq 1\} = 1 - e^{-\lambda_p}. \quad (4)$$

Clearly,  $\varphi$  depends on the parameter  $\lambda_p$ . We compute  $\lambda_p$  based on the change history of the webpage  $p$  within  $n$  download cycles:

$$\lambda_p = \frac{\sum_{i=1}^n w_i \cdot \mathbf{I}(L_i[p])}{n}, \sum_1^n w_i = 1, \quad (5)$$

where  $\mathbf{I}(p_i)$  is an indicator function defined as follows:

$$\mathbf{I}(L_i[p]) = \begin{cases} 1 & \text{if } L_i[p] \neq L_{i-1}[p], \\ 0 & \text{otherwise.} \end{cases}$$

And  $w_i$  is the weight assigned to changes occurred in different download cycles so that the distribution of change events is also taken into account.

Typically,  $w_a \leq w_b$  is satisfied when  $a < b$ , which indicates that changes occurred in the more recent download cycles are more important. We propose four settings for  $w_i$  that we will experiment with later in Section 5:

1. Non-adaptive –  $w_1 = w_2 = \dots = w_n = \frac{1}{n}$ . In this case, all change events have equal importance.
2. Shortsighted adaptive –  $w_1 = w_2 = \dots = w_{n-1} = 0, w_n = 1$ . In this case, the crawler only concerns about the current change status of the webpage.
3. Arithmetically adaptive –  $w_i = \frac{i}{\sum_{i=1}^n i}$ . In this case, bias is given to only the latest change event.
4. Geometrically adaptive –  $w_i = \frac{2^{i-1}}{\sum_{i=1}^n 2^{i-1}}$ . In this case, bias is given to the more recent change events.

## 5. EXPERIMENTAL EVALUATION

### 5.1 Data Collection and Evaluation Metric

We carried out extensive experiments on a large dataset to evaluate the sampling-based update detection algorithm and the various parameter settings we proposed. A collection of real webpages from the WebArchive project<sup>1</sup> were first obtained, and we implemented a special spider to crawl the Internet Archive<sup>2</sup> to obtain the historical snapshots of these webpages. The Internet Archive has archived more than 55 billion webpages since 1996. Excluding the webpages that were not available in the Internet Archive, we eventually constructed a dataset containing approximately 300,000 distinct URLs that belong to more than 210 websites. For each URL, we downloaded from Internet Archive their historical snapshots dated between Oct. 2002 and Oct. 2003. We have the largest number of websites in the .com domain and the largest number of webpages in the .edu domain (see Table 1 for the distribution by different top-level domains). Overall, our dataset is diverse enough to evaluate the proposed algorithms.

domain	total	sites	avg urls / site
.edu	107204 (37.7%)	68	1576.5
.com	100725 (35.4%)	92	1094.8
.gov	38696 (13.6%)	23	1682.4
.org	22391 (7.9%)	16	1399.4
.net	12972 (4.6%)	12	1081.0
.mil	1998 (0.7%)	1	1998.0
Sum	284692 (including 706 misc. URLs)		

**Table 1: Distribution of top-level domains in the collected data; .com and .edu together account for more than 70% of the webpages.**

We used the average *ChangeRatio* as the evaluation metric in the experiments. *ChangeRatio* is defined as the fraction of downloaded and changed webpages  $D_i^c$  over the total number of downloaded webpages  $D_i$  in the  $i$ th download cycle [10]. In our experiments we measure the per-download-cycle *ChangeRatio*  $C_i$  as well as the *average ChangeRatio*  $\bar{C}$  which is the mean  $C_i$  over all download cycles.

### 5.2 Results and Discussion

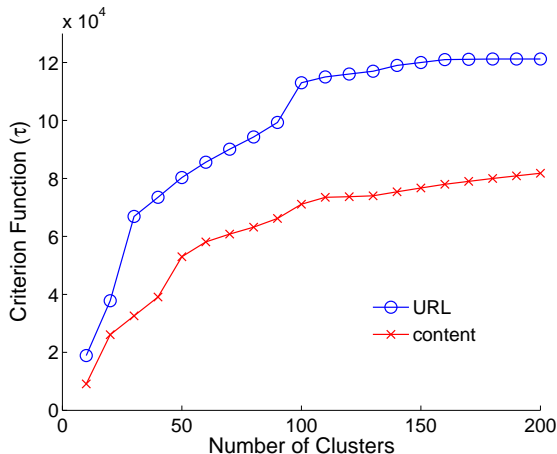
We now present our findings in applying the sampling-based algorithm on the aforementioned data collection. Note that when a webpage is visited, its historical snapshot with the same time-stamp from the Internet Archive is compared with the corresponding copy in the local repository, to check whether it has changed or not; if there is no exact match in the archive, the version with the closest time-stamp is checked instead.

**Estimating Number of Clusters** We employ the *10-fold cross-validation* method to cluster the content of webpages and URLs, and incrementally set the number of clusters from 10 to 200 to learn which is the best choice for the clustering process. Figure 1 shows the values of  $\tau$  under different numbers of clusters,  $k$ . We can see from this figure that the value of  $\tau$  goes up as  $k$  increases. This is because with larger  $k$ , fewer webpages are assigned to one cluster and the clusters are more concentrated. Hence, the cosine similarity between the webpage and the centroid of cluster

<sup>1</sup><http://webarchive.cs.ucla.edu/>

<sup>2</sup><http://www.archive.org/>

gets larger. However,  $\tau$  is not increasing steadily. Its gradient is sharp when  $k$  is smaller. For URL clustering, there is a turning point for  $\tau$  when  $k$  reaches 100: as  $k$  passes 100,  $\tau$  increases much slower than before. This indicates that once  $k > 100$ , increasing  $k$  does not have significant impact on the clustering output. The similar trend also appears in content-based clustering. Therefore, we use  $k = 100$  for all the clustering processes in the following experiments.

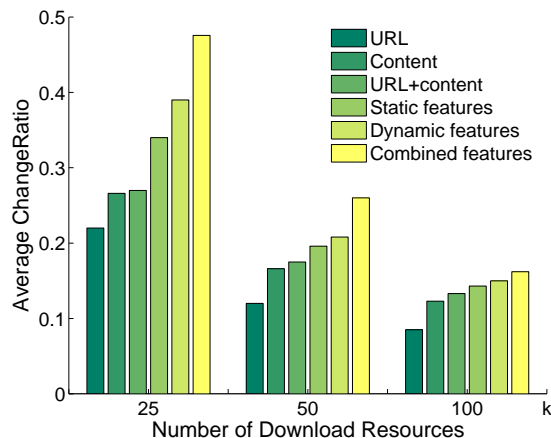


**Figure 1: Cluster quality for URL and content clustering over different numbers of clusters.  $k = 100$  is the optimal setting because it can balance the quality of the clusters and the overhead for clustering.**

**Evaluating Feature Effectiveness** We partition all the features into different categories and apply the clustering process to each category. These categories include: (1) URL: All words in the webpages’ URLs; (2) Content: All words on the webpages; (3) URL+content; (4) Static: All the static features listed in Section 3.1 except the 10 features related to URL words and the 10 features related to webpage content. There are 28 features in total; (5) Dynamic: All the dynamic features listed in Section 3.2. There are 9 features in total; (6) Combined: All the features we have discussed so far. There are 47 features in total.

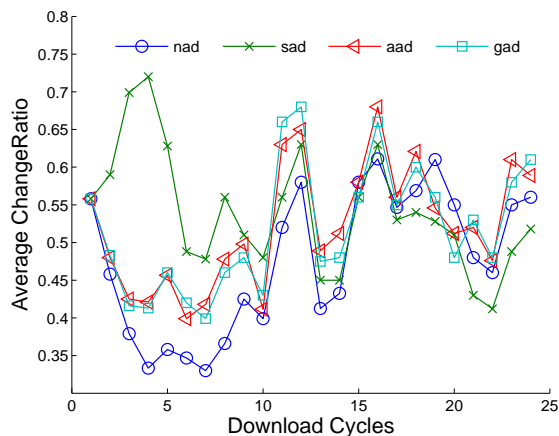
We show the comparison results in Figure 2. For this set of experiments, the download cycle is set to be one month and the download resources  $R$  is set incrementally from 25K, 50K, to 100K. For the sampling process, we set the change frequency to be the non-adaptive one and confidence level at 80%. The figure gives us the following illustrations: First, when the download resource is scarce, the choice of feature sets for clustering has a huge impact on the average ChangeRatio of the clusters. While the combined feature set performs the best, using only the URL words for clustering has the lowest ChangeRatio because it ignores other important features. Second, when the download resource is abundant, the performance of all feature settings become similar. Using words in both URL and content for clustering has similar performance as using words in content only. This is because, compared with content, URL is relative short and much less informative. The performance of using all features for clustering is better than using any of the subsets, as such we use all features discussed for clustering in the following

experiments.



**Figure 2: Comparison of different clustering feature sets. Dynamic features can predict change patterns better than static features.**

**Selecting  $w_i$  for Change History** We experimented with the four different settings for  $w_i$  discussed in Section 4. They are non-adaptive (nad), shortsighted-adaptive (sad), arithmetically-adaptive (aad), and geometrically-adaptive (gad). For this set of experiments, the download cycle is set to be one month and the download resources  $R$  is set to be 25K. Figure 3 shows the results from this experiment. At the beginning, the policy with shortsighted-adaptive  $w_i$ , which focuses on only the current change status, outperforms all the other because others (except the non-adaptive policy) have no knowledge about the change history. On the contrary, using the non-adaptive  $w_i$  has the lowest *ChangeRatio* at the beginning while gradually outperforms the shortsighted-adaptive one. The other two adaptive policies have very similar performances, and eventually their *ChangeRatios* are the highest among all.



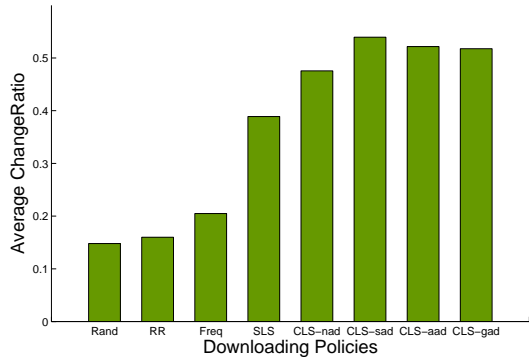
**Figure 3: Comparison of adaptive and non-adaptive  $w_i$ . Using adaptive  $w_i$  can improve the *ChangeRatio* in the long term.**

**Comparing to Existing Methods** We compare the

effectiveness of our cluster-level sampling approach against that of four existing algorithms:

- Random Node Sampling (Rand): The crawler uniformly re-download at random webpages in each download cycle [15].
- Round Robin (RR): The crawler downloads webpages in a round-robin fashion in each download cycle.
- Change-Frequency-Based policy (Freq): The crawler selects webpages to re-download based on their past change frequencies.
- Greedy Site-Level Sampling (SLS): It is a site-level sampling approach proposed in [9]. We set the sample size for each site as the optimal one,  $\sqrt{Nr}$ , which is proposed in [9]. Here  $N$  is the average number of pages in all sites, and  $r$  is the ratio of download resources to the total number of webpages in the local repository.

For our cluster level sampling algorithm (CLS), we show the average *ChangeRatio* of four different settings for  $w_i$  and set the confidence level to 0.8. Figure 4 gives the *ChangeRatio* for all download policies. Clearly, the performance of our cluster-level sampling is better than that of the site-level sampling. This is because in CLS, webpages are grouped based on the features most relevant to their update patterns, while in SLS webpages are grouped based only on their top-level domains. Thus a crawler implementing the CLS scheme can easily capture webpages with similar change patterns. In this set of experiments, using *sad* has higher average *ChangeRatio* than *nad*, *aad*, and *gad*. However, as we observe in Figure 3, the two adaptive strategies *aad* and *gad* will outperform *sad* in the long term.



**Figure 4: Comparison with four published update-detection methods. Our cluster-level sampling algorithm significantly outperforms the site-level sampling method.**

## 6. CONCLUSION

In this paper, we studied how to make the search engine’s local database as up-to-date as possible with a fixed amount of available download resources. Motivated by the observation that webpages sharing similar change patterns tend to have certain similar features, we proposed a sampling-based synchronization algorithm which works at the cluster level. The algorithm first clusters the webpages in the local repository based on the features closely correlated with their

update patterns, then samples the clusters that are more likely to change, and eventually uses the changed samples in each cluster as the *seeds* to discover more changed webpages. Experimental results show that our approach outperforms various existing sampling-based update detection algorithms.

There are several promising directions for future research. First, we plan to investigate different clustering algorithms and find out which is the most suitable for our application. Second, in addition to the *v-fold cross validation*, we can apply some newly proposed methods, e.g. *X-means* [19], to estimate the appropriate number of clusters before the clustering phase.

## 7. REFERENCES

- [1] H. Ali and H. E. Williams. What’s changed? measuring document change in web crawling for search engines. In *SPIRE*, pages 28–42, 2003.
- [2] L. Barbosa, A. C. Salgado, F. de Carvalho, J. Robin, and J. Freire. Looking at both the present and the past to efficiently update replicas of web content. In *WIDM ’05*, pages 75–80, 2005.
- [3] O. Brandman, J. Cho, H. Garcia-Molina, and N. Shivakumar. Crawler-friendly web servers. In *PAWS ’00*, 2000.
- [4] B. E. Brewington and G. Cybenko. How dynamic is the Web? In *WWW ’00*, 2000.
- [5] B. E. Brewington and G. Cybenko. Keeping up with the changing web. *Computer*, 33(5):52–58, 2000.
- [6] J. Cho and H. Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *VLDB ’00*, pages 200–209, 2000.
- [7] J. Cho and H. Garcia-Molina. Effective page refresh policies for web crawlers. *ACM TODS*, 28(4):390–426, 2003.
- [8] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through url ordering. *Computer Networks*, 30:161, 1998.
- [9] J. Cho and A. Ntoulas. Effective change detection using sampling. In *VLDB ’02*, 2002.
- [10] F. Douglass, A. Feldmann, B. Krishnamurthy, and J. C. Mogul. Rate of change and other metrics: a live study of the world wide web. In *USENIX Symposium on Internet Tech. and Syst.*, 1997.
- [11] J. Edwards, K. McCurley, and J. Tomlin. An adaptive model for optimizing performance of an incremental web crawler. In *WWW ’01*, pages 106–113, 2001.
- [12] D. Fetterly, M. Manasse, M. Najork, and J. L. Wiener. A large-scale study of the evolution of web pages. In *WWW ’04*, Budapest, Hungary, 2004.
- [13] G. Grimmett and D. Stirzaker. *Probability and Random Processes*, 2nd ed. Oxford, England: Oxford University Press, 1992.
- [14] G. Karypis and E.-H. S. Han. Fast supervised dimensionality reduction algorithm with applications to document categorization & retrieval. In *CIKM ’00*, pages 12–19, 2000.
- [15] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *KDD ’06*, pages 631–636, Philadelphia, USA, 2006.
- [16] A. Ntoulas, J. Cho, and C. Olston. What’s new on the web?: the evolution of the web from a search engine perspective. In *WWW ’04*, pages 1–12, 2004.
- [17] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.
- [18] S. Pandey and C. Olston. User-centric web crawling. In *WWW ’05*, pages 401–411, 2005.
- [19] D. Pelleg and A. W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *ICML ’00*, pages 727–734, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
- [20] G. Salton. Developments in automatic text retrieval. *Science*, 253:974–979, 1991.
- [21] Q. Tan, Z. Zhuang, P. Mitra, and C. L. Giles. Efficiently detecting webpage updates using samples. In *ICWE ’07*, to appear, 2007.
- [22] J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen. Optimal crawling strategies for web search engines. In *WWW ’02*, pages 136–147, 2002.