Effective Change Detection using Sampling

Junghoo Cho Alexandros Ntoulas

UCLA Computer Science Department
Los Angeles, CA 90095
{cho, ntoulas}@cs.ucla.edu

Abstract

For a large-scale data-intensive environment, such as the World-Wide Web or data warehousing, we often make local copies of remote data sources. Due to limited network and computational resources, however, it is often difficult to monitor the sources constantly to check for changes and to download changed data items to the copies. In this scenario, our goal is to detect as many changes as we can using the fixed download resources that we have. In this paper we propose three sampling-based download policies that can identify more changed data items effectively. In our sampling-based approach, we first *sample* a small number of data items from each data source and download more data items from the sources with more changed samples. We analyze the effectiveness of the sampling-based policies and compare our proposed policies to existing ones, including the state-of-the-art frequency-based policy in [7, 10]. Our experiments on synthetic and real-world data will show the relative merits of various policies and the great potential of our sampling-based policy. In certain cases, our sampling-based policy could download twice as many changed items as the best existing policy.

1 Introduction

Many applications often make local copies of remote data sources. For instance, a data warehouse may copy remote sales and transaction records for local analysis. Similarly, a Web search engine copies a subset of the Web and indexes them to help users access Web pages. In many cases, the remote sources are updated independently of the local copies, so we must periodically *poll* and *download* data from the sources to detect changes and incorporate them to the copies.

Change detection and download is often performed in batch at a regular interval, typically during off-peak hours, to avoide interference with the main tasks that the sources and/or clients perform. As the size of the data grows, however, detecting changes and incorporating them to the copies become increasingly difficult. Due to limited network and computational resources, we may not be able to check every data item in the data sources within the limited time window, so we may miss certain changes at the sources.

In this paper, we address some of the challenges that arise in this context: How can we detect and download as many changed data items as we can, when the source data is updated independently and when we have limited resources? In this scenario, it is important exactly what item we decide to download and check, because we may waste a significant portion of our resources, if we repeatedly download unchanged items.

As we will discuss in more detail later, our main idea is to use *sampling*. That is, we first download a small number of data items from each data source as *samples*, and use the samples to decide which sources we download more data items from. While the idea is simple, our later analysis and experiments will show that sampling-based policies have great potential and lead to significant improvement.

Although the problem of change detection and download arises in various contexts, our work is mainly motivated by our need to manage Web data. In our WebArchive project [20], we try to store multiple versions of Web pages over time, so that users can access the Web of, say, 10 years ago. Due to our limited network resources, however, we cannot constantly download every page to check for changes, so we need to carefully select what pages to download and check. A similar service is currently provided by the *WayBack Machine* [19]. Web search engines also have to address the same problem, because they have to periodically revisit Web pages in order to maintain their indexes up-to-date. This task is typically performed by a program, called a *Web crawler*.

Recently, Cho et al. [7] and Coffman et al. [10] stud-

Proceedings of the 28th VLDB Conference,
Hong Kong, China, 2002

ied how a crawler can detect more changes by predicting *page change frequencies*. That is, the crawler constantly estimates how often a page changes based on the past change history of the page, and uses this estimate to decide how often it will revisit the page in the future. Differently from the existing work, this paper studies how we can detect more changes using *sampling*. As our later experiments will show, our sampling-based policy leads to significant improvement from the frequency-based policy in many cases. In an experiment on real Web data, our sampling-based policy detected *twice* as many changes as the frequency-based policy in certain cases!

In order to design and implement a good sampling-based policy, there are many questions to address. For example, how many samples should a crawler take from each data source? Can a crawler dynamically adjust the sample size to improve effectiveness? How should a crawler use the results from sampling? Can we combine a sampling-based policy with the change-frequency-based policy? To address these questions, we organize the rest of the paper as follows:

- In Section 2, we present a framework to study the change detection and download problem. We discuss various change-detection policies and present evaluation metrics to compare different policies.
- In the first half of Section 3, we propose two sampling-based policies, *proportional* and *greedy*, and analyze their effectiveness. We derive the optimal sample size that maximizes the effectiveness of a sampling-based policy.
- In the second half of Section 3, we propose an adaptive-sampling policy that can dynamically adjust the sample size, based on the changes detected so far. We also study the scenario where we cannot sample enough pages from each data source due to our very limited download resources.
- Finally in Section 4, we experimentally compare our sampling-based policy to others, including the state-of-the-art frequency-based policy. Our experiments will show that our sampling policy is often significantly better than existing ones. The experiments will also reveal the respective merits of our sampling-based policy and the frequency-based policy. To the best of our knowledge, our work is the first one to study the effectiveness of the frequency-based policy *experimentally on real data*. The results will shed light on how we may use various policies in an actual system.

2 Framework

In this paper, we assume that the sources are updated *autonomously* and *independently* of the local copy. That is, we assume a *pull model* where the local copy needs to periodically check the data sources to *detect* and *download* changes. This model is in contrast to a *push model* where the data sources are *cooperative* and willing to *push* their updates to the local copies. Recently, Olston et al.[17] started investigating the push model, but we believe the pull model is more suitable for many existing applications, including the World-Wide Web.

We also assume that the local copy downloads data items *periodically in batch*, say, every weekend. That is, every weekend, we download a fixed number of data items from the sources and update the local copy using the downloaded items. We call this interval – in this case one week – as a *download cycle*. Our goal is to download as many changed items as possible in each download cycle, using the same fixed download resources. This assumption is valid for an environment like the World-Wide Web, where we maintain a large number of data items residing in many different sources, and we do not have enough resources to update them all in a short period of time. The following example illustrates a typical scenario that we assume.

Example 1 We maintain local copies of 10 million Web pages downloaded from 10,000 sites. The 10,000 sites do not inform us of any changes, so we need to periodically download pages to detect and save changes in our copies. Since many users heavily access these pages during weekdays, we can download the pages only on weekends. Given our network bandwidth we can download up to one million pages every weekend. We want to use our limited download resources effectively so that we can download as many changed pages as possible in each week. □

2.1 Download policies

When we can download only a subset of data items in each download cycle, we need to carefully decide what data item to download. There exist a multitude of ways for this decision, including the following:

1. *Round-robin:* We download data items in a round-robin fashion in each download cycle. In case of Example 1, for instance, we download the first 1 million pages in the first week, the second 1 million pages in the second week, etc. Because we maintain 10 million pages locally, every page will be updated exactly once every 10 weeks in this policy.
2. *Change-frequency-based:* Based on the past change history of a data item, we estimate how often the item changes and decide how often to revisit the item. For instance, if we have downloaded an item once every month for one year, and if we detected 4 changes, we may estimate that the item changes once every 4 months and revisit the item accordingly. For more detailed description of this policy, see references [7, 10].
3. *Sampling-based:* We first sample a small number of data items from each data source (e.g., a Web site) and estimate how many items in that source have changed. We then allocate download resources to each data source accordingly, based on the estimates. For instance, in case of Example 1, we may download 10 pages from each of the 10,000

Web sites as samples (a total of 100,000 page samples) and count how many pages in the samples have changed. (For now, we assume that we need to actually download a page to see whether the page has changed or not.) Then based on the counts, we allocate the remaining 900,000 download resources to each Web site accordingly. Later in Section 3, we will discuss this policy in more detail.

The above three policies have their own merits and advantages. The round-robin policy is currently being used by many systems [5, 14] due to its simplicity. It also guarantees that every data item is downloaded at a regular interval. The frequency-based policy has the following advantages and disadvantages:

- **Advantage:** The frequency-based policy is proven to be optimal when we can estimate the change frequencies of data items accurately [7].
- **Disadvantage:** 1) It is very difficult to estimate the change frequency of a data item accurately. Unless we have a long change history of a data item, existing estimation methods often lead to unreliable predictions [8], which in turn lead to an undesirable download policy. In addition, the change frequency itself may change over time, but we may not realize that it has changed.

  2) In order to estimate the change frequencies, we need to keep track of the change history of *every* data item. When we maintain a large number of items, this tracking may incur significant storage and maintenance overhead

A sampling-based policy does not have the drawbacks mentioned above, because it makes a download decision purely based on the samples taken in the *current* download cycle. It does not need to keep track of the previous change history of data items. Later in Section 4, we will compare the effectiveness of the frequency-based policy and the sampling-based policy using real Web data.

At this point, some of the readers may expect that a sampling-based policy would work only when the changes of the data items in the same source are *correlated*. However, we emphasize that this is not the case. If we can take *random* samples from each data source, we are guaranteed that the fraction of changed items in the samples is proportional – in a probabilistic sense – to the fraction of changed items in the data source. So a sampling-based policy does *not* assume any correlation between changes of data items. We should only be able to take *random* samples from each data source.

We also note that it is possible to combine two or more policies to achieve desirable properties. For example, we may use half of our download resources in a round-robin fashion and use the remaining half for a sampling-based policy. This way, we can detect more changes than a simple round-robin policy, while downloading every item at least at a certain interval. Our study will help us employ a combined policy better, through better understanding of the sampling-based policy.

2.2 Evaluation metrics

In order to compare various download policies, we need an evaluation metric. We list three potential evaluation metrics in this subsection:

1. *ChangeRatio* metric: Informally, the *ChangeRatio* metric counts how many changed items we download in a download cycle and uses this number as its performance. More precisely, the *ChangeRatio* metric is defined as the number of downloaded *and* changed items in a download cycle over the total number of downloaded items in the cycle. For example, if we downloaded 1 million items and detected 700,000 changed items, the *ChangeRatio* is 0.7. Since the *ChangeRatio* may vary in different download cycles, we take its average over multiple download cycles. Our goal is to maximize the averaged *ChangeRatio*.

   Note that in certain cases data items may have different "importance," and we may want to detect more changes from more "important" items. To formalize this notion, we may extend the simple definition of *ChangeRatio* by assigning weight w_i to each item o_i and define

   $$\textit{ChangeRatio} = \sum_{i \in \mathbf{R}} w_i \cdot \mathbf{1}(o_i)$$

   Here, $\mathbf{1}(o_i)$ is an indicator function whose value is 1 when the item o_i has changed and 0 when it has not. $\mathbf{R}$ is the set of items that have been downloaded. w_i's are normalized so that $\sum_{i \in \mathbf{R}} w_i = 1$. When all w_i's are equal, this definition reduces to the simple definition.

   The *ChangeRatio* metric is particularly useful when we want to store the *change history* of data items, such as for the WebArchive project [20]. Because our goal is to store as complete change history as possible, we want to maximize the number of detected changes. A similar definition was used in [12].

2. *Freshness* and *Age* metrics: In [7], we proposed two other metrics, called *freshness* and *age*. The freshness of item o_i at time t is defined as

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

   (Up-to-date means that the locally stored image of the item is the same as the image at the source) and the freshness of the entire local copy at time t is

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

   Here, U is the set of all locally stored items. Informally, the freshness metric represents the fraction of data items that are up-to-date. For example, if we maintain 100 pages and if 70 pages are up-to-date at t, its freshness is 0.7. Our goal is to maximize the *time-averaged* freshness under our resource constraints.

The second metric, the age of item o_i at time t, is defined as

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

and the age of the entire local copy is

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

The age represents "how old" the local copy is. For example, if the source data item changed one day ago, and if we have not downloaded the item since then, the age of our local item is one day. Our goal is to minimize the *time-averaged* age using limited resources. Similarly to the *ChangeRatio* metric, we can incorporate different "importance" of objects, by assigning weight w_i's to items and taking an weighted average.

The freshness and age metrics are suitable when we need to keep the local items as up-to-date as possible. However, note that the metrics are hard to measure exactly in practice. That is, in order to estimate freshness (or age), we need to *instantaneously* compare the source items to the local ones, which is often very difficult when we maintain a large number of data items. In addition, we want to optimize the *time-averaged* freshness and age values, but the time average can be obtained only when we know the *entire* change history of *every* data item. Therefore, most of the studies on freshness and age are conducted through theoretical analysis, assuming some stochastic models for data changes.

3. *Divergence* metric: In [17], Olston et al. proposed a very general "staleness" metric called *divergence*. Intuitively, a divergence value represents how different a local data item is from the source item. For example, in a stock-market-monitoring application – where we locally copy stock prices – we may define the divergence of a stock quote as the difference between its current price and the locally-stored value. In general, a divergence metric can be defined as any monotonically-increasing function [17].

In this paper, we mainly use the *ChangeRatio* as our evaluation metric. We made this choice because 1) it is easy to measure in practice on real data and 2) high *ChangeRatio* indirectly implies high freshness, low age, and less divergence.

3 Sampling-based policies

In this section, we discuss sampling-based download policies in more detail. We start our discussion by clarifying our cost model for sampling.

3.1 Sampling cost model

A sampling-based policy needs to sample a few data items from each data source in order to estimate how many items in the source have changed. During sampling, we assume that we need to download an *entire data item* to check whether the item has changed or not. That is, we assume that the cost for *sampling* an item is the same as the cost for actually *downloading* the item. For example, if we can download 100,000 data items in each download cycle and if we sample a total of 10,000 data items, we can download 90,000 more data items in that cycle. We also assume that we do not need to download a sampled item again in the same download cycle, because the item was already downloaded during sampling. This assumption makes our discussion simple, and it is straightforward to extend our current model to the case where sampling cost is lower than downloading cost. For instance, if sampling cost is only 10% of actual downloading cost, we may assume that we can download $99,000$ $(= 100,000 - 0.1 \cdot 10,000)$ more data items for the above scenario.

3.2 Greedy and proportional policies

We now discuss two sampling-based policies, *greedy* and *proportional*. To make our discussion concrete, we use the following as our running example.

Example 2 We locally mirror two Web sites A and B. Each Web site has 100 pages. We can download 100 pages every weekend. To estimate how many pages have changed, we sample 10 pages from each site. Out of the 10 samples, 7 pages changed in A and 2 pages changed in B. We need to decide how to allocate the remaining 80 $(= 100 - 2 \cdot 10)$ page download resources to A and B. We assume that every page is equally important. □

Given the sampling results, we may allocate the download resources to A and B either proportionally or greedily.

1. *Proportional policy:* We allocate the remaining resources to a site proportionally to its number of changed samples. That is, we download $80 \cdot \frac{7}{7+2} = 62$ pages from site A and $80 \cdot \frac{2}{7+2} = 18$ pages from site B.
2. *Greedy policy:* We start from the site that has the most changed samples and download all pages in the site. If we still have remaining download resources, we download more pages from the second-most changed site. We continue this process until we run out of download resources. In the above example, we use all 80 remaining resources for site A, because A has more changed samples than B.

In both policies, we allocate more download resources to the sites with more changed samples, hoping that we will detect more changes. While both policies are reasonable, we can see that the greedy policy is expected to yield better *ChangeRatio* than the proportional policy from the following simple analysis.

Probabilistically, $\frac{7}{10} = 70\%$ of the pages in site A would have changed and $\frac{2}{10} = 20\%$ of B pages would have changed. Therefore, the proportional policy is expected to detect $0.7 \cdot 62 + 0.2 \cdot 18 = 47$ changes from page downloads. Including the 9 ($= 7+2$) page changes detected during sampling, we detect 56 changes in total (i.e., the *ChangeRatio* is $56/100 = 0.56$). In contrast, the greedy policy is expected to detect $0.7 \cdot 80 + 9 = 65$ changes in total (*ChangeRatio* of 0.65).

In general, it is straightforward to prove that the expected *ChangeRatio* of the greedy policy is the highest among *all* sampling-based policies.

Theorem 1 (Optimality of Greedy Policy) *We sample the same number of random pages from each data source and allocate remaining download resources based on the sampling results. In this scenario, the greedy policy is expected to give the highest* ChangeRatio *out of all sampling-based policies.* □

Proof See Appendix. ■

The above theorem shows that the greedy policy yields better *ChangeRatio* on average than the proportional policy. However, the greedy policy may have *larger variation* in its performance, because the greedy policy aggressively allocates *all* of its resources to the site with more estimated changes. When the estimation is correct, this choice yields very high *ChangeRatio*, but when the estimation is incorrect, it also yields very low *ChangeRatio*. In contrast, the proportional policy downloads pages from *every* site, so even when the estimation is inaccurate, it still shows relatively high performance. Later in Section 4, we will investigate this issue experimentally.

3.3 Optimal sample size

In a sampling-based policy, the size of samples affects performance significantly. In this section, we study the optimal sample size that yields the highest *ChangeRatio*. In order to understand the impact of sample size, let us consider a scenario similar to Example 2, but now assume that we sample 50 pages from each site (instead of 10). In this case, we use all of our 100 download resources just for sampling, so we cannot download any more pages from the site with more changed pages. Therefore, the performance of a sampling-based policy would be similar to that of the round-robin policy, because we download random pages during sampling. At the other extreme, if we sample only one or two pages from each site, there is a high chance that the estimated number of changes from samples is inaccurate and we make a wrong download decision.

Figure 1 illustrates this issue more precisely. We obtained the graph assuming that there are two sites, A and B, with 100 pages each, and we download 100 pages in one download cycle. We also assumed that 70 pages changed in site A and 20 pages changed in site B. The horizontal axis shows the sample size that a policy uses and the vertical axis shows the expected *ChangeRatio* of the round-robin, greedy and proportional policies at the given sample size. The graph was obtained analytically.

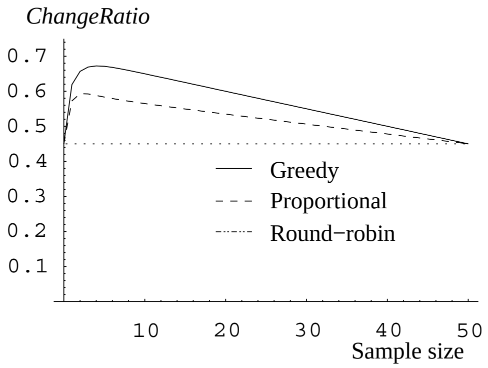

Figure 1: Expected *ChangeRatio* for various sample sizes

Note that the greedy and the proportional policies show the same expected *ChangeRatio*, 0.45, as the round-robin policy when the sample size is either 0 or 50. This is because when the sample size is 0, both policies select a random site for download, and when the sample size is 50, both policies use all its resources just for sampling. Also note that the proportional and the greedy policies show similar performance when sample size is small (≤ 2). This little difference is because the greedy policy is more likely to make an inaccurate download decision with small samples. It needs to sample “enough” pages to make a good decision. From the graph, we can see that the greedy policy shows the optimal performance when it samples about 5 pages from each site.

In general, we can derive the optimal sample size for the greedy policy analytically. To help derivation, we first introduce some notation.

We assume that all Web sites have the same number of Web pages, N. In practice, different Web sites may have different numbers of pages, but in this case, we may interpret N as the *average* number of pages in overall sites. We use r to represent the ratio of our download resources to the total number of pages that we maintain. For example, if we maintain 200 pages and if we can download 100 pages in each download cycle, r is 0.5. We use ρ_i to represent the fraction of changed pages in site S_i. For instance, if site S_1 has 100 pages and if 70 pages have changed, $\rho_1 = 0.7$. When sites have different ρ_i values, we can plot the histogram of ρ_i values as in Figure 2 and approximate it by a continuous density function $f(\rho)$. The goal of the greedy policy is to download pages only from the sites whose ρ_i values are the highest $100 \cdot r\%$ (say, the gray region in the figure). We use ρ_t to represent the threshold ρ value: The sites whore ρ values are higher than ρ_t belong to the top $100 \cdot r\%$ sites. We use $\bar{\rho}$ to represent the average ρ values over all sites. We use $\bar{\rho}_r$ to represent the average ρ values of the sites in the gray region (the sites whose ρ values are above ρ_t). In Table 1, we summarize our notation. Some of the notation in the table will be introduced later.

Under this notation, we can expect that the optimal

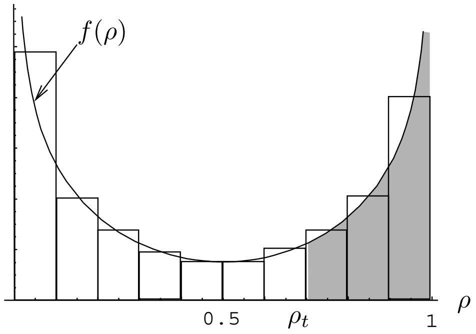

Figure 2: Histogram of ρ_i values of sites

Notation	Meaning
N	Average number of pages in all sites
R	Available download resources
r	Ratio of download resources to the total number of pages we maintain
s	Sample Size
k	Granularity (sample size) of adaptive policy
α	Confidence value of adaptive policy
ρ_i	Fraction of *changed* pages in site S_i
$f(\rho)$	Density function of all web sites' ρ values
ρ_t	Threshold ρ value. If $\rho_i > \rho_t$ for some S_i, then S_i belongs to the highest $100 \cdot r\%$ sites
$\bar{\rho}$	Average ρ values over all web sites
$\bar{\rho_r}$	Average ρ value of web sites having $\rho_i > \rho_t$

Table 1: Notation used throughout the paper.

sample size will depend on the distribution $f(\rho)$, our resource constraints, and the number of pages in the Web sites. The following theorem shows how these parameters affect the optimal sample size.

Theorem 2 (Optimal sample size) *The optimal sample size, s, under the greedy policy is approximately*

$$s \approx \sqrt{\frac{Nrf(\rho_t)}{6(\bar{\rho}_r - \bar{\rho})}} \qquad \square$$

Proof Due to its length and complexity, we give the proof in the appendix. ■

Intuitively, we can understand the result of Theorem 2 as follows: First, when r is large (i.e., when we have relatively large download resources compared to the number of pages that we maintain) we can use more of our resources for sampling, because we can still download many pages from high ρ sites using the remaining resources. Second, when N is large (i.e., when Web sites have more pages), we need to sample more pages from the sites to predict their ρ values better.

Another factor, $\bar{\rho}_r - \bar{\rho}$, indicates that we can sample less pages when $\bar{\rho}_r - \bar{\rho}$ is high (i.e., when the ρ values of the top $100 \cdot r\%$ Web sites are much higher than the average ρ value.) This is because when the ρ values are very different among the sites, the estimated ρ_i values from samples will be very different, so it becomes easier to identify the high ρ sites from the others.

The final factor $f(\rho_t)$ indicates that we need to sample more pages when the value of the density function $f(\rho)$ is high at ρ_t. This is because when many Web sites have ρ values close to ρ_t (i.e., when $f(\rho_t)$ is large), it is more difficult to tell exactly which sites have ρ values higher/lower than ρ_t.

Using the formula in Theorem 2, we can estimate the optimal sample size when we know the distribution of ρ values. In certain cases, however, the distribution may be unknown, and we may not compute the optimal sample size accurately. Even in this scenario, we believe the result of Theorem 2 is still useful, because it shows that the optimal sample size s is proportional to the square root of Nr. As a rule of thumb, therefore, when we do not know the exact distribution of ρ values, we may use $\sqrt{Nr}$ as a rough approximation for the optimal sample size. Clearly, other factors are important to determine the *exact* optimal size, but this approximation will be roughly in the same range as the optimal size, different only by a constant factor. Later in the experiment section, we will verify the result of this section using real Web data.

3.4 Adaptive sampling

The policies that we have discussed so far are *two-stage* policies. That is, we first take a fixed number of samples from each site at a *sampling stage*, and then we download more pages from high ρ sites at a *download stage*. Instead of a two-stage policy, we now discuss an adaptive sampling policy that tries to adjust the sample size dynamically and adaptively.

Our new adaptive policy is essentially based on the greedy policy: After sampling some pages from each site, if we are certain that the ρ value of a site is very high, we download all pages from the site. The difference is that the sample size is not determined in advance under the adaptive policy.

To illustrate, let us assume that we maintain local copies of 4 Web sites, S_1 through S_4. Their ρ values are $\rho_1 = 0, \rho_2 = 0.45, \rho_3 = 0.55, \rho_4 = 1$, and each site has 100 pages. We can download a total of 200 pages in each download cycle. Roughly, our goal is to identify the two Web sites with high ρ values that we will download pages from.

Given the high ρ value of the site S_4, we can expect that the samples from S_4 will have much more changes than the other samples. Therefore, it is relatively safe to pick S_4 for page download early in our sampling. Similarly, it is safe to discard S_1 early on, because of its low ρ value. Compared to S_1 and S_4, S_2 and S_3 require larger samples, because their ρ values are similar and it is difficult to tell which one has a higher ρ value. Based on this intuition, we propose the policy described in Figure 3.

The algorithm takes two input parameters, α and k, whose intuition is given later. Roughly, the algorithm proceeds as follows: It samples k pages from each Web site, and based on the samples it estimates the ρ_i value and its $100 \cdot \alpha\%$ confidence interval for each site (Steps [3] through [6]). Given the distribution of the es-

```
Algorithm 3.1   Adaptive-sampling policy
Parameters:
  $\alpha$: confidence level (a value between 0 and 1)
  $k$: number of pages to sample in each iteration
Procedure
  [1] S = $\{S_1, S_2, \ldots S_n\}$      // Set of sites to be sampled
  [2] Loop while we have download resources
  [3]    For each site $S_i \in$ S
  [4]       Sample $k$ pages from $S_i$
  [5]       $\rho_i$ = Estimate of $\rho$ value for $S_i$ base on the samples so far
  [6]       $(l_i, h_i)$ = 100 $\cdot\ \alpha$% confidence interval for $\rho_i$
  [7]    Compute threshold $\rho_t$ from the distribution of estimated $\rho_i$'s
  [8]    For each Web site $S_i$ in S
  [9]       If $(h_i < \rho_t)$ S = S - $S_i$
                // $\rho_i$ too low. We do not download from $S_i$
  [10]      If $(\rho_t < l_i)$ download all pages in $S_i$ and S = S - $S_i$
                // $\rho_i$ very high. We download pages from $S_i$
```

Figure 3: Algorithm of the adaptive-sampling policy

timated ρ_i values, it can predict the threshold ρ_t value (Step [7]). For example, if we can download about half of the sites in each download cycle, and if half of the estimated ρ_i's are above 0.6, $\rho_t = 0.6$.

After estimating ρ_t, it compares the confidence intervals of ρ_i to the threshold. If the confidence interval for S_i is strictly lower than the threshold ($h_i < \rho_t$), it stops sampling from the site (Step [9]); It has enough evidence that the ρ_i of S_i is below the threshold. Similarly, if the confidence interval of S_i is strictly above the threshold ($\rho_t < l_i$), it downloads all pages from the site (Step [10]).

The α and k values are configuration parameters set by the user. When the α value is low, the algorithm makes a download/discard decision "aggressively" and picks a site for download (or discard) even with low confidence. Thus, it allocates less resources to sampling and more resources to page downloads. The k value determines the granularity of sampling adjustment. When k is small, the algorithm re-estimates ρ_i values more frequently and makes a download (or discard) decision more often. Thus, the algorithm may show better performance but it may require more processing power. Later in Section 4, we will study the impact of α and k values on the effectiveness of the policy. We will try to identify good α and k values that yield high performance.

3.5 Subset sampling under low download resources

So far, we have implicitly assumed that we have a sufficiently large amount of download resources, so that we can sample a reasonable number of pages from each site and still download more pages from high ρ sites. In certain cases, however, this assumption may not be valid. We may not be able to sample enough pages from each site, due to our limited resources available. In this section, we study how we should handle low-resource scenarios.

Generally, there is an interesting relationship between the download resource size and the performance of a sampling-based policy. At one extreme, when we have few download resources and cannot sample enough pages from each site, a sampling-based policy would perform similarly to the round-robin policy: after sampling a couple of pages from each site, we cannot download any more pages from high ρ sites, and we end up visiting a small but different portion of the pages in each cycle, just like the round-robin policy. (Because we take random samples, we will visit different pages in different cycles with high probability.) At the other extreme, if we have enough resources to download every page in each download cycle, a sampling-based policy will perform similarly to the round-robin policy again, because both policies will download all pages in every cycle.

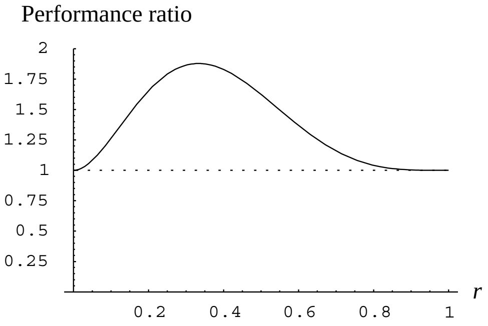

Figure 4: Comparison of the round-robin policy and a sampling policy for various resource constraints

Figure 4 shows a *hypothetical* graph that illustrates this relationship. The horizontal axis shows the resource ratio r (the number of download resources to the number of total pages we maintain). When r is 1, we can download all pages in each download cycle, and when r is 0, we can download no page. The vertical axis shows the performance ratio of a sampling-based policy to the round-robin policy (*ChangeRatio* of a sampling-based policy over *ChangeRatio* of the round-robin policy). When the sampling-based policy performs better than the round-robin policy, this ratio is higher than 1. In the graph, a sampling-based and the round-robin policies show similar performance (the performance ratio is 1) when resource ratio r is close to either 0 or 1, because of the reasons discussed above. In between these two extremes, a sampling-based policy shows better performance than the round-robin policy.

To improve performance for the scenario of very limited resources ($r \approx 0$), we propose that a sampling-based policy should select a small *subset* of its data in each download cycle, and sample and download pages only from the subset:

- *Subset sampling under low download resources:* When the download resources are too limited to sample enough pages from each site, we group the sites into m subsets. In each download cycle, we pick one subset and sample and download pages only from the sites in the subset. We revisit the subsets in a round-robin manner over multiple download cycles.

For instance, consider the following example:

Example 3 We maintain local caches of Web pages from 1000 sites. Each Web site has 100 pages. Every weekend, we can download 500 pages in total. In this scenario, our simple greedy (or adaptive) policy cannot work effectively, because we can sample less than one page from each site.

To handle this scenario, we may use the subset-sampling policy. First, we divide the sites into groups of, say, 10 sites. Every week, we select a group of 10 sites and sample, say 10 pages from each of the 10 sites. Assuming we use the greedy-policy, we can use the remaining 400 ($500 - 10 \cdot 10$) download resources to download pages from high ρ sites. □

When we need to use the subset-sampling policy, one important question is how many sites we should put in each subset. Should we sample 10 sites in one download cycle, sampling all 1000 sites over 100 cycles? Or should we sample 20 sites in each cycle? The answer depends on the amount of available resources and the distribution of ρ values among the sites. Although we cannot derive a closed formula for the optimal number of sites to sample, we believe that the number should be determined such that we can download all pages from high ρ sites after sampling.

For example, if the ρ values follow the distribution of Figure 2, and if roughly 30% of the sites belong to the grey region (high ρ region), we should be able to download all pages from these top 30% sites in each download cycle. If our subset is too small and if we have to download pages from lower ρ sites (given our download resources), performance would degrade. If our subset is too large and if we "waste" most of our resources for sampling, performance would also suffer. Later in Section 4, we experimentally study the effectiveness of the subset-sampling policy.

3.6 Is *Greedy* too greedy?

While the greedy (and adaptive) policy can improve the overall *ChangeRatio*, it may be possible that some pages are never downloaded, because the policy downloads pages only from the high ρ sites. The following theorem proves that this is not the case.

Theorem 3 *When every page changes at some points of time, every page is eventually downloaded.* □

Proof See Appendix. ■

Although the theorem proves that every page will eventually be downloaded, it does not guarantee that pages are downloaded within a "reasonable" period of time. It also does not address the case when some of the pages does not change at all. In the extended version of this paper [9], we examine how often each page is downloaded on real Web data under the greedy policy. The results show that most of the changed pages are downloaded within a reasonable period. We also note that if it is important to download every page within a certain interval, we may decide to combine the round-robin policy with the greedy (or other sampling-based) policy. For example, we may want to use, say, 30% of download resources in a round-robin fashion and use the remaining 70% for the greedy policy.

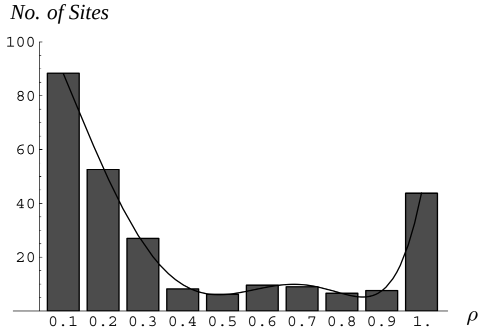

Figure 5: Histogram of ρ_i values in the dataset.

4 Experiments

Following on from our theoretical analysis, we conducted a number of experiments in order to study the behavior and performance of the aforementioned policies. Most of our experiments were conducted on real data collected from the Web. The dataset contained 6-month change history of approximately $353,000$ Web pages distributed among 252 Web sites. The data was collected by our WebArchive crawler, which visited the Web sites once every month for a period of 6 months. Since changes could be detected only from the second visit (in the first visit, we do not know whether a page has changed or not), we had a total of 5 change history data for each page. Thus, our experiments could run up to 5 download cycles.

It may seem that 5 download cycles is relatively small, but as we will see in the following sections it is enough to bring up the potential of the sampling policies. Also, when it is necessary to run experiments on a longer change history, we assumed that our 5 cycle data would repeat over time. That is, if we detected changes from a page in the 2nd and 5th cycles, we assumed that we detect changes in 7th, 10th, 12th, 15th cycles, etc. Our dataset is publicly available from our Web site [20].

We should emphasize that our later experiments did *not* actually crawl and download pages. All experiments were conducted on the *same* data collected by our WebArchive crawler. This setup enables a fair comparison among policies. Also, throughout experiments, we assume that the cost for sampling a page is the same as the cost for actually downloading it.

4.1 Distribution of ρ values

We start our discussion by investigating the properties of our dataset. In particular, we show the distribution of ρ values of the sites (Section 3.3) in Figure 5. The horizontal axis represents ranges of ρ values, and the vertical axis shows the number of Web sites with the given ρ value. Label 0.1 on the horizontal axis means the range of 0 to 0.1. Note that the ρ value of a site may vary between download cycles. However, we could not detect any meaningful fluctuation in ρ values between cycles from our dataset. We plotted the histogram using the *average* ρ value of a site over all 5 download cycles.

This figure shows that in our data, there exist quite

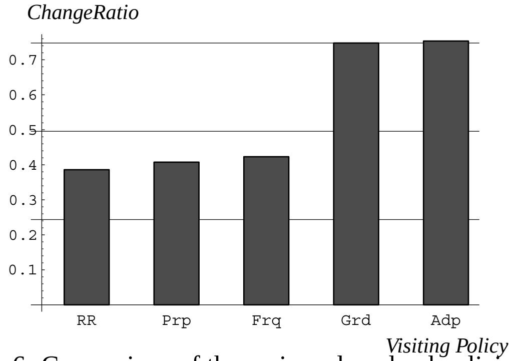

Figure 6: Comparison of the various download policies.

a few Web sites whose pages change very frequently. About 18% of the sites has ρ values between 0.9 and 1. Also, a lot of sites are static and remain (almost) unaltered throughout our experiment. More than 35% of the sites has ρ values between 0 and 0.1 This fact intuitively suggests that 1) it can be relatively easy to detect the high and low ρ sites using sampling and 2) if we can identify the Web sites with very high and low ρ values and allocate our download resources appropriately, we may observe a significant improvement in the number of detected changes.

While our data indicates that the ρ values of Web sites follow a V-shaped distribution, it will be also interesting how various download policies perform for different distributions. For this reason, in the extended version of this paper [9], we report some of our results on *synthetic* data whose ρ values follow a *normal distribution.* The results from the synthetic data strongly indicate that while the exact numbers are different, the general trend that we observed from the real data is still valid even for a normal distribution.

4.2 Rough comparison of download policies

In this section, we conduct a rough comparison of various policies using our real data. For the experiments, we assumed that each policy can download $R = 100,000$ pages in each download cycle. The greedy and proportional policies used sample size $s = 10$ and the adaptive policy used $k = 10$ (discussed in Section 3.4) and a confidence level $\alpha = 0.9$. Note that we did not try to optimize these parameters. We selected the numbers rather arbitrarily for this experiment. However, we believe that the results from this experiment would show the relative potential of various algorithms. In later sections we will be examining the impact of the various parameters more thoroughly.

Figure 6 shows the results. The horizontal axis corresponds to various policies and the vertical axis shows the *ChangeRatio*s of the policies (averaged over 5 download cycles). The frequency-based policy that we used in the experiment is the one proposed in reference [7]. From a first glimpse at this figure, the reader can observe that our greedy (Grd) and adaptive (Adp) policies perform surprisingly well compared to the round-robin (RR) and even the frequency-based (Frq) policy. Their *ChangeRatio*s are almost twice as high as the frequency-based policy! Since their *ChangeRatio*s are around 0.75, even if we could design a hypothetical *oracle* policy, which could magically download only changed pages, the improvement would be less than 25%. The performance of the proportional (Prp) policy is similar to that of the frequency policy, and the performance difference between the greedy and the adaptive policy is marginal.

While the results strongly indicate that the greedy and the adaptive policies are very effective, we note that the frequency-based policy could not show its full potential in this experiment, due to our small number of download cycles. Since the frequency-based policy did not know how often pages change, it visited *every* page *once* in a round-robin manner in the beginning, until the first half of the 4th cycle.[1] Only after that, the policy started to adjust revisit frequencies based on estimated change frequencies. Therefore, in the first three visits, the frequency-based policy showed the same performance as the round-robin policy and only from the second half of the 4th download cycle, it started to show some improvement.

Because of this fact, the comparison of the frequency-based policy and our sampling-based policies may not be fair, but we note that this is the situation in any practical system. Any system has to estimate page change frequencies in order to use the frequency-based policy, so it will suffer from poor performance in the beginning. In contrast, our sampling-based policies perform well without any change history data. Later on, we will compare the long term performance of the frequency-based policy and our greedy policy.

4.3 Optimal sample size

In this subsection, we examine the impact of the sample size on the performance of sampling-based policies. For this purpose, we ran the greedy and the proportional policies on our data set, keeping the resource size constant to 100,000 pages and varying the sample size from 1 to 400. The outcome of this experiment is drawn in Figure 7. The horizontal axis represents the sample size and the vertical axis shows the *ChangeRatio* at the given sample size. From the graph, we can confirm the trend that we discussed before:

- When the sample size is too small, a sampling-based policy shows poor performance. It often makes a poor download decision. This degradation is particularly noticeable for the greedy policy.
- When the sample size becomes too large, performance also degrades, because sampling-based policies waste more resources for sampling than they ought to.

We can see that the optimal sample size for the greedy policy is around 10–50. This range matches well with the prediction of Theorem 2. In Section 3.3 we argued that $\sqrt{Nr}$ is a good rule of thumb for the optimal sample size

[1] We had $353,000$ pages and we visited $100,000$ pages in each download cycle. Therefore, we need $3\frac{1}{2}$ cycles to visit every page once.

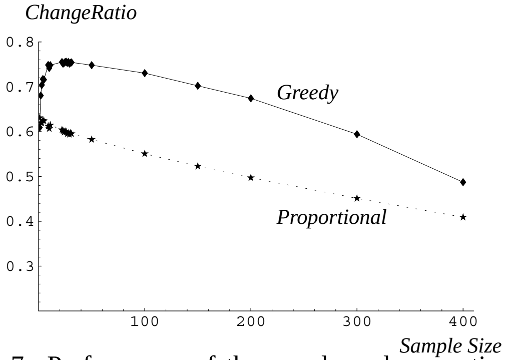

Figure 7: Performance of the greedy and proportional policies over various sample sizes.

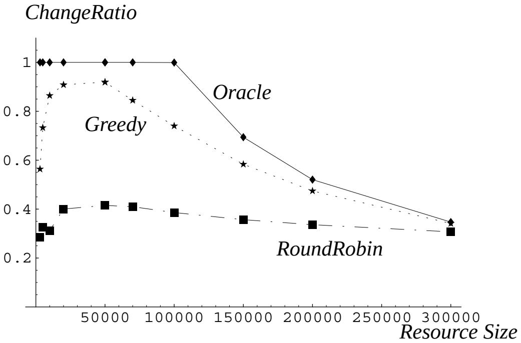

Figure 8: Performance of oracle, greedy and round-robin policies over various resource sizes.

when we do not know the exact distribution of ρ values. Given our parameters ($N = 353,000/252 \approx 1,400$ and $r = 100,000/353,000 \approx 0.28$), this formula predicts that the optimal sample size is $\sqrt{Nr} \approx 20$, which is in the range that we observe from our experiment.

From the graph we can see that for all sample sizes, the greedy policy shows better average *ChangeRatio* than the proportional policy. We expected this result from our discussion in Section 3.2, but we also discussed that the greedy policy may have a larger variation in *ChangeRatio* than the proportional policy. To compare their variations, we measured the standard deviation (s.t.d.) of *ChangeRatio* between download cycles for both policies. From this estimation, we could see that the s.t.d. of the greedy policy is larger than that of the proportional policy (e.g., 0.027 vs. 0.023 for sample size 1). However, because the variation is very small (~ 0.02) compared to average *ChangeRatio* (~ 0.75), we believe that the variation issue is of negligible importance.

4.4 Resource size and subset sampling

We now study the effect of varying resource size on the performance of the greedy policy. For the experiments, we ran the greedy and the round-robin policies on our data set. The greedy policy used the sample size 10 and we varied the resource size R from 3,000 to 300,000 pages.

Figure 8 shows the results from this experiment. The horizontal axis corresponds to the resource size and the vertical axis shows the *ChangeRatio* at the given resource size. The *oracle* policy is the one that can magically download only the changed pages. We show its performance for comparison purposes. Note that the *ChangeRatio* of the oracle policy goes below 1 for $R > 100,000$. This is because in each download cycle, only about $100,000$ pages changed and if our resource size is larger than $100,000$, the oracle policy starts to download unchanged pages. For most of resource sizes, the greedy policy shows much better performance than the round-robin policy.

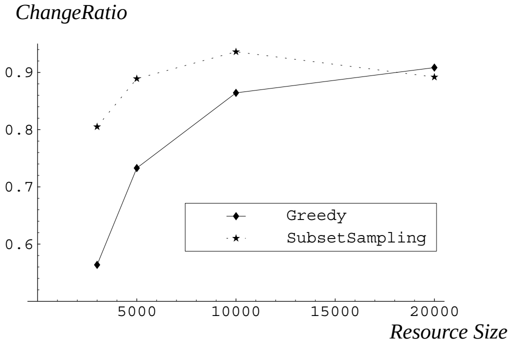

Figure 9: Comparison of *ChangeRatio* for *Greedy* and *SubsetSampling* for low number of resources.

The graph confirms our earlier discussion (Section 3.5): When the resource size is large, the performance of all policies become similar, because all policies download every page. When the resource size is too small, the performance of the greedy policy degrades. This degradation starts at $R < 20,000$.

To study the impact of the subset sampling policy, we divided sites into small subsets and sampled pages only from one subset in each download cycle when $R < 20,000$. The size of each subset was selected so that we can download about 18% of the pages in the subset in each download cycle. For example, when we have $10,000$ download resources, each subset had about $60,000$ pages. We selected 18% because about 18% of the sites in our dataset belonged to the right peak of the V-shaped distribution (Figure 5). The result from this experiment is shown in Figure 8. From the graph, we can see that the subset sampling policy improves the effectiveness of the greedy policy when the resource size is small. For instance, when $R = 5,000$ the *ChangeRatio* improves from 0.73 to 0.89 when we used the subset sampling policy.

4.5 Long-term performance of the frequency policy

The results in Section 4.2 showed that the performance of the greedy policy is significantly better than the frequency-based policy in a short term. In this section, we study the long-term performance of the frequency-based policy and compare it to the greedy policy.

Towards this goal, we ran the frequency and the greedy policies for longer download cycles, by assuming that the observed change history of the pages repeats forever. For example, if we detected changes from a page in the 2nd and 5th cycles, we assumed that we detect changes in 7th, 10th, 12th, 15th cycles, etc. Figure 10

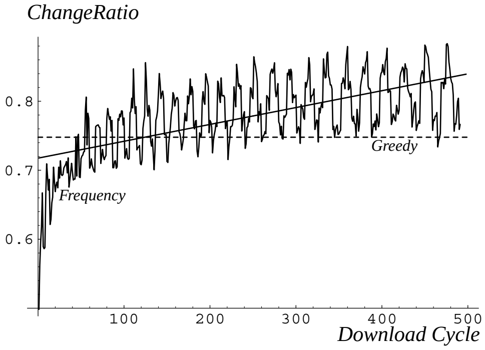

Figure 10: Performance of the frequency and the greedy policies over time

shows the results. The horizontal axis corresponds to a download cycle, and the vertical axis shows the *ChangeRatio* at the given download cycle. The dashed line is the result of the greedy policy and the solid line with wide fluctuation is the result of the frequency-based policy.

The wide fluctuation in the frequency-based policy is mainly because it periodically downloads pages that rarely change. Even if a page has never changed, we cannot be sure that its change frequency is zero, so we have to periodically go and check the page for change. The dips in the graph correspond to the points when the policy downloaded infrequently changing pages. Note that the interval between these dips increases steadily over time. This is because as we accumulate more change history data, we can be more confident that the page does not change, and thus need to check the page less often.

From the graph, we can clearly see that the performance of the frequency-based policy improves steadily over time. Its performance is significantly lower than the greedy policy in the beginning, but from around 100th download cycle, it starts to show better performance. Therefore, in the long run, the frequency policy can be better than the greedy policy However, keep in mind that 100 download cycles is a long period of time. Because we downloaded pages once every month, 100 cycles roughly correspond to 10 years!

4.6 Adaptive policy

We now study the impact of the k and α values (introduced in Figure 3) on the performance of the adaptive policy. To study their impact, we ran the adaptive-sampling policy for various k and α values. Figures 11 and 12 show the result. Figure 11 shows the *ChangeRatio* of the adaptive policy for various k values (the horizontal axis) when $\alpha = 0.9$. From the graph, we can see that the performance decreases as k increases. (There are small fluctuations, but we believe they are experimental variations.) This result is expected because when k is small, we try to re-estimate ρ_i values after small number of samples, and thus make a download decision more frequently with more accurate ρ_i values. From the figure, we can see that the performance decrease is relatively small until $k = 10$.

Figure 12 shows the *ChangeRatio* for various α values (the horizontal axis). From the graph, we can see that the confidence interval does not affect the performance of the policy significantly. We could not detect any meaningful difference in *ChangeRatio* for most of α values. One thing to note is that the performance for $k = 1$ is worse than others ($k = 5, 10$) when α is small ($\alpha \leq 0.75$). This is because when α is small and $k = 1$, the policy started to pick (or discard) a site for download after it took only 2 samples: Because it selected a site for download (or discard) too aggressively and too early, it often made wrong decisions. In other cases ($k = 5$ or 10, or α is large), these early decisions did not happen, because the policy had to sample 5 or more pages when $k = 5$ or $k = 10$, or because it made a download (or discard) decision conservatively when α is large.

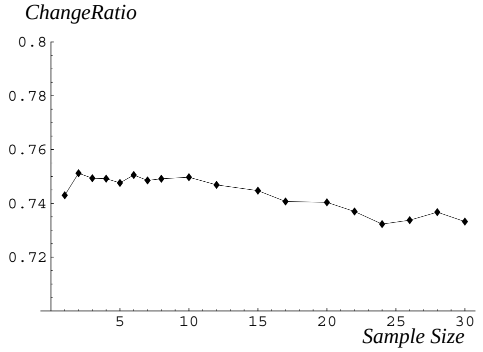

Figure 11: Performance of $Adaptive$ over various sample sizes k.

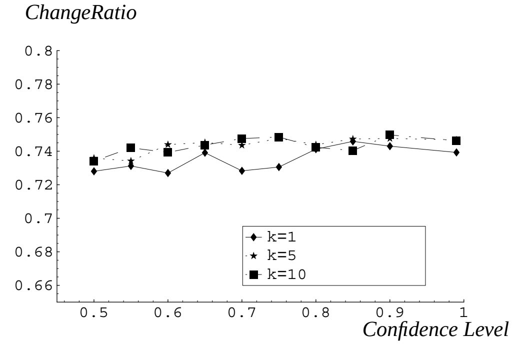

Figure 12: Performance of $Adaptive$ over various confidence values α.

Based on the results, we believe $k = 10$ and $\alpha \approx 0.9$ are good parameters to use for a scenario similar to our Web data.

5 Related work

References [7, 10] study how a crawler should download pages to maintain its index "up-to-date." Assuming that the crawler knows the exact change frequencies of pages, the references present an optimal algorithm. As we learned from our experiments, this change-frequency-based algorithm performs relatively well once it collects a large amount of history data. However, history collection incurs significant overhead, and until it collects enough data, the algorithm performs poorly. Our sampling-based policies do not need to track any change history, and it shows significant improvement without any history data. Reference [12] proposes another download algorithm based on linear programming. The algorithm shows promising results, but because algorithm becomes more complex over time, the authors report that the algorithm has to periodically "reset" and "start from scratch;" The algorithm takes (practically) infinite amount of time to finish after a certain number of download cycles. In contrast, the complexity of our sampling-based algorithms stay the same over time.

A lot of work has been done to maintain the consistency of replicated data [3, 1, 11, 15, 16]. This work studies the tradeoff between data consistency and read/write performance. In most of the existing work, however, researchers have assumed a *push* model, where the sources *notify* the replicated data of the updates. For example, Olston et al. [17] proposed a new architecture in which data sources can notify caches of important changes. In many contexts, particularly for the Web, this push model is not applicable, because data sources often do not inform others of their changes.

Sampling is a popular technique that has been used in multiple disciplines for various optimizations [13, 22, 21, 6]. The contribution of this paper is to apply sampling techniques to the context of change detection, and study a variety of issues arising in this context.

The *multi-armed bandit problem* is well known in the statistics and AI community. The problem is to identify the slot machine with the highest chance of winning through *exploration* and *exploitation*. The problem is proven to be NP-hard [4], and people have proposed a range of approximation algorithms [2]. The setting of the multi-armed bandit problem is slightly different from ours, because bandit-problem assumes that the user can play the best slot machine infinitely. In contrast, we can download only a limited number of pages from each data source, so we need to find the top $r\%$ sources, not just the top source. This difference makes the policies take quite different forms.

6 Conclusion and future work

In this paper, we studied how we can detect changed data items effectively using sampling. We proposed three sampling-based policies, *greedy*, *proportional* and *adaptive*, and evaluated their performance analytically and experimentally. We also compared the sampling-based policies to other existing policies. Our experiments showed that the greedy policy is easy and simple to implement and shows one of the best performance in many scenarios. Given its simplicity and performance, we believe that the greedy policy is good for practical systems. Its complexity is similar to the widely-popular round-robin policy, while its performance is close to (or even better than) the frequency-based policy. Also, we learned that the frequency-based policy is not very effective in certain cases, because it takes a long time to estimate the change frequencies of pages. We now briefly discuss a few avenues of future work.

- If we want to maximize performance, we may want to combine a sampling-based policy with the change-frequency-based policy. That is, we start with a sampling-based policy in the beginning, and once we collect enough change history data, we start using the frequency-based policy. When we should start this transition? What can we do if the change frequency itself may change over time?
- In this paper, we assumed that we sample a few pages from each *Web site* or each *data source*. But there is no inherent reason to sample at the level of a site. What if we sample a few pages from each *directory*? What if we group Web pages based on their *contents* and sample a few pages from each group? Would we get better performance?

References

[1] R. Alonso, D. Barbara, and H. Garcia-Molina. Data caching issues in an information retrieval system. *ACM TODS*, Sep 1990.

[2] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: The adversarial multi armed bandit problem. In *Proc. of FOCS*, pages 322–331, May 1995.

[3] P. Bernstein and N. Goodman. The failure and recovery problem for replicated distributed databases. *ACM TODS*, Dec 1984.

[4] D. A. Berry and B. Fristedt. *Bandit problems: sequential allocation of experiments*. Chapman and Hall, 1985.

[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proc. of WWW conf.*, April 1998.

[6] S. Chaudhuri, R. Motwani, and V. R. Narasayya. On random sampling over joins. In *Proc. of SIGMOD conf.*, 1999.

[7] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. In *Proc. of SIGMOD conf.*, May 2000.

[8] J. Cho and H. Garcia-Molina. Estimating frequency of change. Technical report, DB Group, Stanford University, Nov 2001.

[9] J. Cho and A. Ntoulas. Effective change detection using sampling (extended version). Technical report, UCLA Computer Science Department, 2002.

[10] E. Coffman, Jr., Z. Liu, and R. R. Weber. Optimal robot scheduling for web search engines. *Journal of Scheduling*, 1(1):15–29, June 1998.

[11] L. S. Colby, A. Kawaguchi, D. F. Lieuwen, and I. S. Mumick. Supporting multiple view maintenance policies. In *Proc. of SIGMOD conf.*, 1997.

[12] J. Edwards, K. McCurley, and J. Tomlin. An adaptive model for optimizing performance of an incremental web crawler. In *Proc. of WWW conf.*, 2001.

[13] P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proc. of VLDB conf.*, pages 311–322, 1995.

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

[15] N. Krishnakumar and A. Bernstein. Bounded ignorance: A technique for increasing concurrency in a replicated system. *ACM TODS*, 19(4), December 1994.

[16] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat. Providing high availability using lazy replication. *ACM Trans. Comput. Syst.*, November 1994.

[17] C. Olston and J. Widom. Best-effort cache synchronization with source cooperation. In *Proc. of SIGMOD conf.*, May 2002.

[18] D. D. Wackerly, W. Mendenhall, and R. L. Scheaffer. *Mathematical Statistics With Applications*. PWS Publishing, 1997.

[19] Internet Archive. `http://www.archive.org`.

[20] WebArchive Project. `http://webarchive.cs.ucla.edu`.

[21] Y.-L. Wu, D. Agrawal, and A. E. Abbadi. Using the golden rule of sampling for query estimation. In *Proc. of SIGMOD conf.*, 2001.

[22] Q. Zhu and P.-A. Larson. A query sampling method of estimating local cost parameters in a multidatabase system. In *Proc. of ICDE conf.*, 1994.

A Rough sketch of the proofs for various theorems

Theorem 1 *We sample the same number of random pages from each data source and allocate remaining download resources based on the sampling results. In this scenario, the greedy policy is expected to give the highest* ChangeRatio *out of all sampling-based policies.* □

Proof We use c_i to represent the number of changed items in the samples from source S_i. Without losing generality, we assume c_1 is the highest and c_2 is the second highest and so on. We assume that we download R_i items from source S_i and the total of R items from all sources (i.e., $\sum_i R_i = R$). We assume each data source has N data items.

When we take random samples and detect c_i changed items from s samples, we expect $100 \cdot \frac{c_i}{s}\%$ items in S_i have changed. Therefore, we expect to detect $\frac{c_i}{s} R_i$ changes from S_i when we download R_i items from S_i. Then the total number of detected changes is expected to be

$$\sum_i \frac{c_i}{s} R_i, \quad \text{where } \sum_i R_i = R$$

Our goal is to maximize the above value. From the equation, it is easy to see that the formula takes its maximum value when we assign N to R_i for low i values ($i = 1, 2, \dots$), because c_i values are the highest for low i's. That is, we have to start downloading data items from the sources with the most changed samples. ■

Theorem 2 *The optimal sample size, s, under the greedy policy is approximately*

$$s \approx \sqrt{\frac{Nrf(\rho_t)}{6(\bar{\rho}_r - \bar{\rho})}}$$ □

Proof To help our derivation, we first assume an *oracle greedy* policy, which can magically identify the sites with high ρ values and can directly download pages from those sites without any sampling. We use $P_o(\rho_i)$ to denote the probability that the oracle greedy policy downloads a site S_i when its ρ value is ρ_i. By the definition of the oracle greedy policy, $P_o(\rho) = 1$ when $\rho > \rho_t$ and $P_o(\rho) = 0$ when $\rho < \rho_t$. We show the graph of $P_o(\rho)$ as a dashed line in Figure 13.

We use the $P_g(\rho_i)$ to represent the probability that our greedy policy downloads a site S_i when its ρ value is ρ_i. We show its graph as a solid line in Figure 13. Because the greedy policy sometimes makes wrong decisions, the graph $P_g(\rho)$ is not a step function like $P_o(\rho)$. It approaches 0 as ρ decreases, but when ρ is close to ρ_t, $P_g(\rho)$ is larger than 0 even if $\rho < \rho_t$ because some of the sites are mistakenly downloaded. Similarly, $P_g(\rho)$ approaches 1 as ρ increases, but when ρ is close to ρ_t, $P_g(\rho)$ is smaller than 1 even if $\rho > \rho_t$. As we increase the sample size s, the graph of $P_g(\rho)$ will become similar to that of $P_o(\rho)$, because the greedy policy makes more accurate download decision with larger samples.

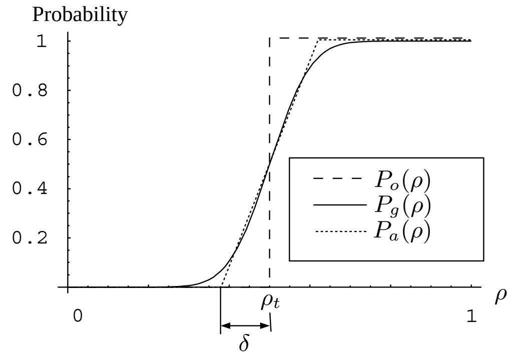

Figure 13: Probability $P_o(\rho)$ and $P_g(\rho)$

We now approximate the function $P_g(\rho)$ by a piecewise linear function $P_a(\rho)$ shown as a dotted line in Figure 13. The general form of $P_a(\rho)$ can be represented as

$$P_a(\rho) = \begin{cases} 0 & \text{for } \rho \in [0, \rho_t - \delta] \\ \frac{1}{2\delta}(\rho - \rho_t) + \frac{1}{2} & \text{for } \rho \in (\rho_t - \delta, \rho_t + \delta) \\ 1 & \text{for } \rho \in [\rho_t + \delta, 1] \end{cases} \tag{1}$$

and we need compute δ to find the approximation. Based on the result of Lemma 1 (which is given later), we will use $1/\sqrt{s}$ as the approximate δ value.

We now compute the expected performance of the greedy policy. Under the greedy policy, we download s sample pages from every site. Therefore, the expected number of changes that we detect during sampling is $Ms\bar{\rho}$, where M is the total number of sites that we maintain and $\bar{\rho}$ is the average ρ value for overall sites. Once the greedy policy decides to download more pages from site S_i, the expected number of changes that we detect from S_i is $(N - s)\rho_i$. (Remember that we download $N - s$ more pages from S_i, because we do not download the sampled pages again.) Because the site S_i is downloaded with the probability $P_g(\rho_i)$ under the greedy policy, the total number of changes that we expect to detect during actual download is $\int_0^1 P_g(\rho)(N-s)\rho f(\rho)d\rho$. Therefore, the total number of changes that we detect both from sampling and from actual download is

$$(N - s)\int_0^1 P_g(\rho)\rho f(\rho)d\rho + Ms\bar{\rho}$$
$$\approx (N - s)\int_0^1 P_a(\rho)\rho f(\rho)d\rho + Ms\bar{\rho}$$

Assuming $f(\rho) \approx f(\rho_t)$ for $\rho \in (\rho_t - \delta, \rho_t + \delta)$, and using the analytical form of $P_a(\rho)$ in Equation 1, we can compute the above integral, and the result is

$$(N-s)M\left(\frac{f(\rho_t)}{6s}+\bar{\rho}_r\right)+Ms\bar{\rho}$$

Our goal is to find the s value that maximizes the above formula. It is straightforward to show that the above formula takes its maximum when

$$s=\sqrt{\frac{Nrf(\rho_t)}{6(\bar{\rho}_r-\bar{\rho})}}$$ ■

Lemma 1 *The δ value for Equation 1 can be approximated as $1/\sqrt{s}$.* □

Proof From Figure 13, we see that δ is a value where $P_g(\rho_t+\delta)$ becomes close to 1. For this proof, we assume that we pick the δ value such that

$$P_g(\rho_t+\delta)=0.95 \tag{2}$$

We now try to approximate $P_g(\rho_i)$, the probability that we download more pages from site S_i when its ρ value is ρ_i.

The site S_i is downloaded when its estimated ρ_i value (denoted as $\hat{\rho}_i$) is greater than ρ_t. That is, $P_g(\rho_i) = Pr\{\hat{\rho}_i > \rho_t\}$. To estimate the ρ_i value, we use X/s, where X is the number of changed pages in the sample and s is the sample size. That is, $\hat{\rho}_i = X/s$. Because we take s random samples from S_i where $100\rho_i\%$ of the pages changed, the random variable X follows a binomial distribution with a success rate ρ_i, and we can approximate $\hat{\rho}_i = X/s$ by the normal distribution with mean ρ_i and the standard deviation $\sqrt{\rho_i(1-\rho_i)/s}$. When we do not know ρ_i value, the standard deviation $\sqrt{\rho_i(1-\rho_i)/s}$ is often approximated by $\frac{1}{2\sqrt{s}}$ [18], so we may assume that $\hat{\rho}_i$ follows the normal distribution $N(\rho_i, \frac{1}{2\sqrt{s}})$.

When $\hat{\rho}_i$ follows this normal distribution, the probability $Pr\{\hat{\rho}_i > \rho_t\}$ is 0.95 when $\rho_t = \rho_i - 1.65\frac{1}{2\sqrt{n}}$. That is,

$$P_g(\rho_i)=0.95 \text{ when } \rho_i=\rho_t+\frac{1.65}{2\sqrt{n}} \tag{3}$$

By comparing Equations 2 and 3, we can see that

$$\delta=\frac{1.65}{2\sqrt{n}}\approx\frac{1}{\sqrt{n}}$$ ■

Theorem 3 *When every page changes at some points of time, every page is eventually downloaded.* □

Proof The proof is straightforward. Let us select a random page that belongs to site S_i. Because every page changes, there exists a time t where all pages in S_i have changed. Then, in the download cycle after t, the site S_i will be downloaded under the greedy policy, because all sample pages in S_i have changed. ■