Alexandria Engineering Journal

Harvesting OAI-PMH repositories using adaptive synchronization

Noha Adly
Computer and Systems Engg. Dept., Faculty of Engg., Alexandria University, Alexandria 21544, Egypt
noha.adly@alex.edu.eg

Metadata harvesting requires timely propagation of up-to-date information from thousands of *Repositories* over a wide area network. It is desirable to keep the data as fresh as possible while observing the overhead on the *Harvester*. An important dimension to be considered is that Repositories vary widely in their update patterns; they may experience different update rates at different times or unexpected changes to update patterns. In this paper, we define data Freshness metrics and propose an adaptive algorithm for the synchronization of the Harvester with the Repositories. The algorithm is based on meeting a desired level of Freshness while incurring the minimum overhead on the Harvester. We present a comparison between different policies for the synchronization within the framework devised. It is shown that the proposed policy outperforms the other policies, especially for heterogeneous update patterns. Further, we propose a tool for the administrators of the Harvesters that enable them to choose the level of Freshness to operate at while balancing the tradeoff between the penalties incurred from staleness of the data and the overall performance.

إن حصاد البيانات يتطلب نشر سريع للمعلومات من آلاف المستودعات عبر الشبكات الواسعة المدى. ويجب أن تكون البيانات حديثة مع مراعاة الحمل الواقع علي الحاصدة. وينبغي النظر لبعدا هاما هو أن المستودعات تتفاوت على نطاق واسع في أنماط التحديث؛ فقد يتعرضون لمعدلات تحديث مختلفة في أوقات مختلفة أومن الممكن حدوث تغييرات غير متوقعة لأنماط التحديث. ويقدم هذا البحث مقاييس لحداثة البيانات ويقترح خوارزم قابل للتكيف لتزامن الحاصدة مع المستودعات. والخوارزم المقترح قائم على تحقيق مستوى محدد من حداثة البيانات مع تحميل الحاصدة أقل تكلفة ممكنة. ويقدم البحث مقارنة بين خوارزميات مختلفة للتزامن وقد أظهرت النتائج أن أداء الخوارزم المقترح يفوق الخوارزميات الأخرى، وخاصة لأنماط تحديث غير متجانسة. وعلاوة على ذلك ، يقترح البحث أداة لمديري الحاصدات التي تمكنهم من اختيار مستوى الحداثة الأمثل والذي يعمل على تحقيق التوازن بين قدم البيانات وتكلفة الأداء على الحاصدة.

Keywords: Data synchronisation, Freshness constraints, Distributed objects, Harvesting, Digital libraries, OAI-PMH

1. Introduction

There is an exponential growth of online material and digital libraries that play a key role in managing this information by structuring the content so that it is discovered easily and effectively. Many repositories use the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) [1] to expose metadata about their resources and contents. OAI-PMH is based on the standard technologies HTTP and XML as well as the Dublin Core metadata scheme. It is a set of six verbs or services that provides an open interface for metadata exchange and harvesting. Within OAI-PMH, a *Data Provider* is a *Repository* that exposes its structured metadata; and a *Harvester*, operated by a *Service Provider,* makes OAI-PMH service requests to harvest that metadata from *Repositories*. *Service Providers*, then, provide value-added services, such as federated search [2, 3], on the harvested data extracted from the *Repositories*. A general configuration of OAI-PMH is shown in fig. 1.

Selective harvesting allows *Harvesters* to limit harvest requests to portions of the metadata available from a *Repository*. The OAI-PMH supports selective harvesting and *Harvester*s are expected to exploit this property to limit the load placed on *Repositories* and *Harvester*s while maintaining fresh data for services offered by the *Service Provider*. Selective harvesting is supported in OAI-PMH through timestamps, included as `from` argument in the `ListRecords` requests and expressed in seconds' granularity, which are used to harvest only those records that were created, deleted or modified within a specified range.

The synchronization problem addresses how to keep the metadata records of the *Repositories* and *Harvester* consistent. Frequent harvesting results in the data at the *Service Provider* being up-to-date and consistent with the *Data Provider*. However, frequent harvesting results in a high overhead on both the *Harvester* and the *Repositories*, which renders the harvesting inefficient, especially if the *Data Provider* has not been updated during the harvest interval. On the other hand, without frequent harvesting, *Service Providers* may become inconsistent with *Data Providers*: not only can new records be missed, but deletions and modifications as well and hence mislead the results offered to the user by the *Service Provider*. The challenge is how to design a harvesting algorithm that strikes the balance between the *Freshness* of the data and the overhead incurred.

A large number of repositories have been using OAI-PMH to expose their data, which are of different domains; ranging from scholarly publishing data such as E-print repositories [4-6] or education material such as HEAL [7], multimedia resources, biomedical data [8], and archeological data [9]. These applications are likely to have a small but steady stream of daily or weekly updates. However, different applications started to arise that manage data of different nature. Recent initiatives [10] have proposed making usage data of scholarly information service, collected from web logs, available using OAI-PMH and focused on promoting its applications and creating value-added services on this data such as derivation of global measures of impact and the identification of global trends. Also, recently, there has been a growing interest in harvesting news, annotations [11], reviews of articles and RSS feeds. Those applications are likely to have a large number of updates and with high frequency.

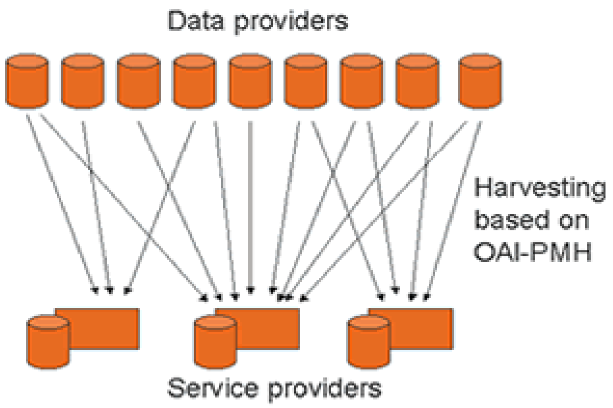

Fig. 1. General configuration of OAI-PMH.

Therefore, it is expected with current applications that *Repositories* would be heterogeneous in nature: different *Repositories* may have different update rate and a *Repository* may have different update rate at different times of the day. The update pattern of *Repositories* plays a major role in determining the balance between frequent harvesting, which guarantees Freshness of data at the expense of high overhead and infrequent harvesting which could result in stale data. Inconsistency or stale data, although could be acceptable in some applications, would be undesirable for some other applications e.g. news feeds, which are sensitive to data Freshness. Therefore there is a need for an adaptive policy that adjusts the harvesting according to the update patterns of the *Repositories*.

Other major potential users of OAI-PMH are search engines. Although, current commercial search engines make a limited use of OAI-PMH to index their data, a study [12] performed on 10 millions records of OAI-PMH repositories revealed that Google, Yahoo! and MSN indexed only 60%, 44% and 7% respectively of these records. However, as interest in revealing site content to web crawlers in a structured manner has increased recently, it is expected that major search engines will support more of OAI-PMH in order to index more content. This will lead to a larger number of *Repositories* registering and implementing OAI-PMH to be able to share their contents. Hence, efficient *Harvesters* are needed that will be able to pull data with dynamic behavior from a large number of *Repositories*.

This growing interest in variety of applications suggests that the environment will be much more dynamic than before, with a larger number of *Repositories* to be harvested and with a variety of natures of *Repositories*, with different update patterns.

In this paper, we present an adaptive pull-based policy for harvesting data from a set of *Repositories* that aims to reduce the overhead on the *Harvester*, and consequently on the *Repositories*, while maintaining *Freshness* of data at a certain level. In order to ensure *Freshness* of data without wasting resources, we provide a framework for measuring the *Freshness* of data at the *Harvester* as well as the cost, which allows devising an optimal algorithm for harvesting that is able to adapt to changes in the update patterns at *Repositories*. The algorithm presented is compliant with the OAI_PMH protocol with minimal changes required at the *Repository* and the *Harvester*. It relies on piggybacking compact representation of the *Repository* workload on `ListRecords` response. It is shown that the proposed policy results in a reduction in the cost on the *Harvester* compared to other policies while providing comparable level of *Freshness*. The benefits have shown to be maximized for heterogeneous update patterns. Further, we propose an alternative formulation of the overall cost which combines the penalties incurred from the *Staleness* of the data and the overhead on the *Harvester* and devise a tool that would help the administrator of the *Harvester* to choose an adequate level of *Freshness* that would balance the tradeoff between the *Freshness* of the data and overall performance.

The structure of the paper is as follows. section 2 discusses previous work in synchronization and measuring data *Freshness*. In section 3, we present a framework for deriving measures for the *Freshness* of data and *cost* on the *Harvester* that allows us to formulate the optimization problem and derive its solution. Section 4 presents the Optimal Adaptive Policy *OAP*(θ). In section 5, we provide a comparison between *OAP*(θ) and three other policies for harvesting. In section 6 we introduce an alternative metric for the cost and devise an approach that helps in the selection of the level of *Freshness* and tuning the overall performance. Finally, section 7 concludes the paper.

2. Related work

Synchronization and Freshness problems arise in various contexts. In [13], several definitions of data *Freshness* and the metrics measuring them are introduced according to the applications where they are used; whether replications systems, federated databases, data warehousing, web portal, cashing systems, etc. They presented a taxonomy based upon the nature of the data, the type of the application and the synchronization policy used. Our work is driven by synchronizing *Harvester* with *Repositories* within OAI-PMH protocol.

Synchronization of large collection of objects, for example, web crawlers, has been addressed in [14, 15] where they have defined age and freshness metrics by modeling the average update frequency of individual elements of a database as well as the whole database. They analyzed different synchronization policies based on the frequency of synchronizing the local database, the frequency of synchronizing individual elements, the synchronization order and the synchronization points over time. However, their approach relies on discovering the update time of each individual web page, which is different from than the incremental harvesting model of the OAI-PMH.

Labrinidis [16] considered freshness in the context of view materialization in caching dynamic web content. They studied selecting which views to materialize in order to maximize performance while keeping data freshness at acceptable level. A Quality of Data (QoD) metric was defined to evaluate how fresh the data served to the users is. They propose an algorithm which constantly monitors the QoD of served data and periodically adjusts the materialization plan by allocating more (or less) resources when there is a QoD deficit (or surplus). This study also focuses on individual web pages.

Driven by results showing that in web caching 30–50% of cache hits result in unnecessary validations, which incur high latency, Bright et al. [17] presented two history-based policies, that establish their prediction on the repetitive nature of update history. They are an extension to the TTL

(Time-To-Live) policy, where each object is assigned a TTL and a validation occurs for any cached object whose TTL has expired. The TTL value is usually estimated as a function of the time that an object was last modified. However, in [17] they are targeting an environment where updates patterns are non-homogenous, capturing updates in a timely manner is critical and some degree of staleness is unacceptable. Hence, they modeled update history of an object as a cyclic stochastic model that can be extended with bursts or deviations from the cyclic history. It has been shown that the history based policies outperform TTL either for cyclic update pattern or acyclic history that exhibits bursts. It should be noted that this approach is different from the OAI-PMH synchronization because within web caching, synchronization is done on the object level, that is, only when this particular object is accessed it is refreshed; while in OAI-PMH the synchronization is applied to all objects of the repository.

In [18], they studied the synchronization problem of the OAI-PMH. By examining the harvest logs of Arc [19], an OAI harvester for e-print services, they concluded that most *Repositories* change at a steady rate, but the rates vary dramatically from site to site. They suggested four possibilities for adaptive policies for synchronization. The first is based on the *Harvester* estimating the update frequency by learning the harvest history and the second is based on the *Repository* notifying the *Harvester* of its update frequency as a response to an `Identify` request. Although both policies are OAI-PMH compliant, the details of the algorithms were not discussed. Also, relying on information sent through the `Identify` request is not adequate since this verb is used only for newly registering *Repositories*. Further, the metrics introduced for studying the synchronization were mainly used for formalization of some definitions and did not allow for quantification of the *Freshness* or the overhead that could help in evaluating the proposed algorithms. The other two algorithms were based on either *Repositories* notifying the *Harvester* whenever content is changed or on a Push-based mechanism. Other than they have not been presented in details or evaluated, both proposals are not OAI-PMH compliant and would require major changes in the protocol.

3. Framework

To study the synchronization problem, we present a framework that allows us to study and measure the metrics that affect the performance. One important measure is the quality of the data or, *Freshness*. The other metric which we take into consideration is the overheard, or the *Cost*, incurred on the *Harvester*.

3.1. Freshness measure

When an element is updated at a *Repository R*, this element becomes stale with respect to the *Harvester*. The element remains stale until a harvest occurs where the value of the element at the *Harvester* is updated. Obviously, it is required that the data elements harvested be as fresh as possible, that is more up-to-date. Let $\{R_1, R_2, \ldots R_M\}$ be M *Repositories* to be harvested and $R_i = \{e_1, e_2, \ldots e_{N_i}\}$ be *Repository i* with N_i elements.

An element in a *Repository* is considered *fresh* at time *t* if it is up-to-date at time *t* w.r.t. to the *Harvester* i.e. if its value at *Repository* is equivalent to its value at *Harvester* at time *t*. Otherwise the element is considered stale.

Definition 1: *Freshness* of element e_j at time *t*:

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

The *Freshness* of R_i, $F(R_i,t)$, is defined as the fraction of the R_i that is up-to-date. $F(R_i,t)$ is a rational number between 0 and 1, with a value of one, if all elements of R_i are up-to-date and would be zero if all elements are stale. Given that R_i contains N_i elements, $F(R_i,t)$ is the average of the freshness values of all elements that compose R_i .

$$F(R_i, t) = \frac{1}{N_i} \sum_{j=1}^{N_i} F(e_j, t)$$

Note that *Freshness* is hard to measure exactly in practice, since we need to instantaneously compare the data elements of the *Repository* to the *Harvester*. But it is possible to estimate *Freshness* given some information about how the elements of the *Repository* change. In order to measure *Freshness*, we observe the synchronization stream and the update stream for a certain observation period T. Assume that the average update rate of R_i is λ_i and that the *Harvester* performs P_i pulls for each R_i at regular intervals $I_i=T/P_i$. Fig. 2 shows the evolution of updates with the horizontal axis representing the time and the vertical axis representing the number of stale items.

The synchronization stream during the observation period T is viewed as a sequence of harvests requests made at time I_i , $2I_i$, $3I_i$, …. Then the number of stale items at R_i at time t:

$$S_i = \lambda_i * t \qquad t = 0,1,2,\ldots I_i$$

Assume we synchronize at t=0 and t= I_i, then, from fig. 2 the average number of stale items:

$$\overline{S_i} = \lambda_i * \frac{I_i}{2}$$

The *Freshness* of a *Repository* R_i:

$$F_{Ri} = 1 - \overline{S_i}/N_i$$

$$F_{R_i} = 1 - \frac{\lambda_i I_i}{2N_i} = 1 - \frac{\lambda_i T}{2N_i P_i}.$$

The *Freshness* of *Harvester H* is defined as the average *Freshness* of all the *Repositories* it harvests.

$$F_H = \frac{\sum_{j=1}^{M} N_j * F_{R_j}}{\sum_{i=1}^{M} N_i} = 1 - \frac{\sum_{j=1}^{M} \lambda_j I_j}{2\sum_{i=1}^{M} N_i}. \tag{1}$$

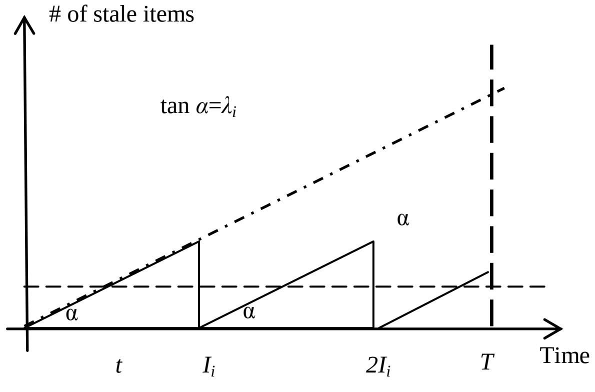

Fig. 2. Evolution of updates at a *Repository* R_i.

3.2. Cost measure

Another important measure that affects the performance is the overhead incurred on the *Harvester*. The *Cost* on the *Harvester* depends on the update rate λ_i and the pull rate P_i for each *Repository* it harvests. For each harvest, the *Harvester* extracts new records from R_i, which incurs a communication cost as well as a processing cost. This cost is paid even if there are no new records to harvest. Also, the *Harvester* extracts, processes and applies every new update to his local copy of the database. Let

C_U = Cost incurred from extracting and processing a single update.

C_p = Cost of initiation, negotiation and communication of a pull. Then,

$C_H|_{Pi}$ = Cost of *Harvester* for pulling R_i

$C_H|_{Pi} = T\lambda_i * C_U + P_i * C_P$

$$C_H = C_u T \sum_{i=1}^{M} \lambda_i + C_p \sum_{i=1}^{M} P_i\ . \tag{2}$$

3.3. Optimal harvest intervals

This section will study how often a *Harvester* should pull each *Repository*, when it knows how often they change, in order to minimize the *Cost* while maintaining a certain level of *Freshness*. We formulate the problem as an optimization problem with the objective to determine the optimal harvest interval *I.*

Problem: Given λ_i, N_i, $F_H = \theta$, find I_i which minimize the cost C_H

$$C_H(I) = C_u T \sum_{i=1}^{M} \lambda_i + C_p T \sum_{i=1}^{M} I_i^{-1}\ .$$

Given freshness F_H=θ or $\theta = 1 - \frac{\sum_{j=1}^{M} \lambda_j I_j}{2\sum_{j=1}^{M} N_j}$.

We can solve the above constrained optimization problem using the method of

Lagrange multipliers [20], where the constraint function is

$$g(I)=\sum_{j=1}^{M}\lambda_j I_j-2(1-\theta)\sum_{j=1}^{M}N_j=0$$

Define the Lagrangian Λ as

$$\Lambda(I,\mu)=C_H(I)+\mu.g(I)$$

$$=C_uT\sum_{i=1}^{M}\lambda_i+C_pT\sum_{i=1}^{M}I_i^{-1}+\mu\left[\sum_{j=1}^{M}\lambda_j I_j-2(1-\theta)\sum_{j=1}^{M}N_j\right]$$

Solving for points I where

$$\nabla\Lambda(I,\mu)=\nabla C_H(I)+\mu.\nabla g(I)=0$$

From the partial derivatives with respect to I, we can deduce

$$I_j=\pm\frac{\sqrt{\lambda_1}}{\sqrt{\lambda_j}}I_1 \qquad \forall\ j=2\rightarrow M$$

Substituting into the partial derivate with respect to the Lagrange multiplier μ

We get $$I_i=\frac{2(1-\theta)\sum_{j=1}^{M}N_j}{\sqrt{\lambda_i}\sum_{j=1}^{M}\sqrt{\lambda_j}}, \qquad (3)$$

and

$$C_H(I)=C_uT\sum_{i=1}^{M}\lambda_i+C_pT\frac{\left[\sum_{i=1}^{M}\sqrt{\lambda_i}\right]^2}{2(1-\theta)\sum_{j=1}^{M}N_j}. \qquad (4)$$

4. Optimal adaptive policy algorithm

Current harvesting algorithms are based on a fixed uniform harvest interval that is applied to all *Repositories*. Such algorithms will not work well in an environment where updates patterns change dynamically. The heterogeneous nature of *Repositories* workloads mandates that the harvesting algorithm, be adaptive in order to evolve under changing workload pattern.

In this section we propose an Optimal Adaptive Policy algorithm, OAP(θ), a harvesting algorithm that is executed at the *Harvester*, where θ is a threshold specified by the *Harvester*. OAP(θ) strives to maintain the overall *Freshness* above the specified threshold θ and also keeps the cost at the *Harvester* as low as possible.

OAP(θ) is inherently adaptive. The algorithm relies on the *Harvester* collecting statistics from the *Repositories* concerning their workloads and computes the optimal intervals at which it pulls each *Repository* to achieve the level of *Freshness* desired θ while minimizing the cost incurred. Namely, the *Harvester H* estimates λ_i and N_i for each R_i and computes the optimal intervals from eq. (3). One main concern while devising OAP(θ) is to be compliant with the OAI-PMH protocol with minimum or no changes introduced to the protocol.

The main OAI-PMH verb used by OAP(θ) is the `ListRecords` verb, which is used to harvest records from a *Repository* based on a timestamp, where the `from` argument specifies the lower bound for the timestamp-based selective harvesting. OAI-PMH controls the return of large number of records through partitioning the records into batches and the use of a `resumptionToken` with each batch. This partitioning is accomplished as follows: a *Repository* replies to a `ListRecords` request with an *incomplete list* and a `resumptionToken`; in order to retrieve the next portion of the complete list, the next request from the *Harvester* must use the value of that `resumptionToken` element as the value of the `resumptionToken` argument of the request. Finally, the response containing the incomplete list that completes the list must include an empty `resumptionToken` element. The complete list then consists of the concatenation of the *incomplete lists* from the sequence of requests.

OAP(θ) will be using the `resumptionToken` as the mean to pass on the information needed from the *Repository* to the *Harvester*. It makes use of the fact that the `resumptionToken` is already incorporated into the protocol and has associated attributes that are useful to the implementation of OAP(θ), without the need to change the operation of a verb or to introduce a new verb or to change the XML schema. Namely, `completeListSize`, an attribute associated with the `resumptionToken`, is an integer indicating the cardinality of the complete list to be sent; which basically represent the number of updates sent from the *Repository* during this harvest cycle. OAP(θ) introduces a new attribute to the `resumptionToken`, which is `totElements`, indicating the total number of elements at the *Repository* at the time. These two values are passed with every response to a `ListRecords` request from the *Repository* to the *Harvester*. Therefore, OAP(θ) suggests a minor change in the implementation of the `ListRecords` verb at the *Repository* side. More precisely, it suggests that the *Repository* includes a `resumptionToken` in every response to `ListRecords`, even if the need for partitioning does not arise. The `resumptionToken` will be empty if the whole set of updates are to be sent in one partition, and will have an identifier if the set of updates is partitioned. It should be noted that making `resumptionToken` mandatory for the *Repository* does not present an overhead since these attributes are sent with the whole list and not on the record level. Further, the values of the attributes `totElements` and `completeListSize` sent are already known to the *Repository* and do not need to be computed.

Harvester H estimates the update rate of R_i, λ_i from the number of updates it receives from R_i in the current harvest. Let the number of updates H receives from R_i at Pull j is U_i^R and the total number of elements at R_i received is N_i^R. The value of U_i^R and N_i^R represent the value of the `completeListSize` and `totElements` attributes of the `resumptionToken` transmitted from R_i along with the response to the `ListRecords` request. H can use the *recursive prediction error method* [21] to estimate the update rate in the near future. Namely, $\lambda_i^H = (1-g)\lambda_i^P + g\,\lambda_i^R$, where

- λ_i^H = new estimate of update rate of R_i for the next period
- λ_i^P = old estimate for update rate in the last interval
- λ_i^R = update rate for the current interval=U_i^R/I_i, where I_i is the interval at which these updates occurred.
- g = gain factor, 0<g<1 suggested [21] to be set to 0.25

Although more sophisticated methods could be used by the *Harvester* for estimating the number of updates, it is believed that this heuristic is simple and incurs a small overhead. Basically, H needs just to keep an array λ_i^P of size M that keeps the actual rates of updates at the current interval received from *Repositories* for use of the estimate of the number of updates for the next interval. So the storage space and the computational complexity are negligible. The Pseudo code for OAP(θ) is as follows:

Algorithm OAP(θ):
while (true) **do** {
  find k such that $I_k \leftarrow \mathrm{Min}\{I_i\} \quad \forall\ i \rightarrow 1\ldots M$
  Send `ListRecords` request to R_k
  Extract from response U_k^R and N_k^R
//Estimate update rate for R_k for the next period
  $\lambda_k^R \leftarrow U_k^R / I_k$
  $\lambda_k^H \leftarrow (1-g)\lambda_k^P + g\,\lambda_k^R$
  // Compute new intervals I_i for all R_i
  for i←1 to M **do** {

$$I_i = \frac{2(1-\theta)\sum_{j=1}^{M} N_j^R}{\sqrt{\lambda_i^H}\sum_{j=1}^{M}\sqrt{\lambda_j^H}}$$

  }
  $\lambda_k^P = \lambda_k^R$
  // Get next R to be harvested
}

5. Comparison between different policies

In order to evaluate the potential benefits of the OAP(θ), we provide a comparison between the OAP(θ) and other policies for variant workloads. We represent the variation in the workload by considering four types of *Repositories* that exhibit different behaviors.

Namely, we assume that a *Repository* can have a small number of elements (N_i =1000) and others may have a large number of elements (N_i =10,000). Further each *Repository* can have a small update rate (λ_i = 10% N_i) while another *Repository* can experience a large update rate (λ_i = 50% N_i). This generates four types of *Repositories* simulating different behaviors as shown in table 1.

It is assumed that the number of *Repositories* to be harvested M=1000 and the observation period is taken to be T=1 day. The cost of a harvest cycle is set to 50 units (C_P = 50), while the cost of extracting and processing a single update is set to 0.1 (C_U=0.1).

5.1. OAP(θ) vs. Uniform adaptive policy

The objective of this experiment is to compare the Optimal Adaptive Policy OAP(θ) with a Uniform Adaptive Policy (UAP). The UAP is set such that the *Harvester* pulls all the *Repositories* at a uniform (fixed) update interval I_U. This is compared to OAP(θ) which sets a different pulling interval for each *Repository* according to the behavior of the *Repository* relative to the workload patterns of all other *Repositories*. In order to choose the Uniform Interval I_U, we assume that the *Harvester* is aware of the workload on each *Repository*, and hence computes I_U as the Optimal Interval to achieve the required *Freshness* θ given that all *Repositories* are combined into a single site, which would result in the same *Freshness* as the OAP(θ).

In this experiment, we assume that the *Repositories* are a mix of T1 and T4 workload; and we vary the percentage of *Repositories* that belong to T1 versus T4. That is, we evaluate a workload where 75% of the *Repositories* follow the pattern of type T1 and 25% of them follow T4. Then we change this percentage till we reach 30% of *Repositories* of Type T1 and 70% of type T4. We evaluate the overhead incurred on the *Harvester* for each policy for different *Freshness* thresholds θ, as shown in fig. 3. We plot the ratio of the cost of UAP, C'_H, versus the cost of OAP(θ), C_H, (C'_H / C_H) for different mixes.

Table 1
Four different types of workloads

	T1	T2	T3	T4
λ_i	100	500	1000	5000
N_i	1000	1000	10000	10000

Results show that the gains of OAP(θ) are higher when the mix of *Repositories* is inclined towards T1, with UAP suffering from increase in the cost ranging between 21% to 71% for different *Freshness* θ. As the workload mix moves toward T4, the cost of UAP decreases, but is still higher than OAP(θ), showing a degradation of 10% to 34% when the workload is evenly distributed between T1 and T4 and being 16% when the majority of *Repositories* are of T4. It is observed that as θ increases, the benefits of OAP(θ) are more obvious where the degradation in UAP ranges between 71% to 16% for θ=0.98 and 46% to 10% for θ=0.95. This shows that the OAP(θ) benefits are more dramatic for systems demanding high *Freshness*. Also, the OAP(θ) is more adjustable to the variation of the workload mix than UAP.

5.2. OAP(θ) vs. Uniform non-adaptive policy

In this experiment, we compare OAP(θ) with a *Harvester* that will apply a Uniform Policy as well; however, we assume that the *Harvester* is not aware of the actual mix between the *Repositories* he is about to harvest. Namely, he knows that the *Repositories* are a mix of T1 and T4, and that the mix would range between 80% to 60% of T1 versus T4. Hence he estimates that the mix would be 70% of T1 and 30% of T4 and it computes the uniform interval I_U based on this estimate. We compare UNAP with OAP(θ) in case the actual mix is ranging between the estimate ±10%. So we plot the variation in the mix between 80% and 60% and we evaluate the *Freshness* and *Cost* of both policies for different *Freshness* thresholds θ, as shown in figs. 4 and 5.

It is observed that when the actual mix is of T1=70%, which matches the estimates of UNAP, both policies have the same *Freshness*, while UNAP has a higher overhead in the cost ranging between 18% to 60% for different θ. When the actual mix moves towards T1, UNAP experiences degradation in the cost ranging between 30% to 95% while the *Freshness* of UNAP is superior to that of OAP(θ) by a range of 0.1% to 0.001%. When the actual mix moves towards T4, the cost of UNAP decreases, but still is higher than OAP(θ) by a range of 11% to 4%. This comes at the expense of the *Freshness* which decreases by a range of 0.1% to 0.001%.

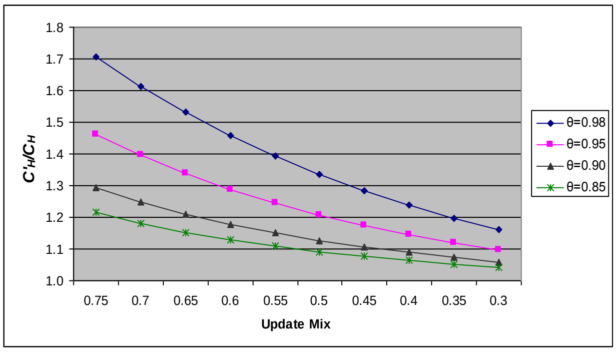

Fig. 3. The cost of UAP vs. OAP(θ) varying the workload mix.

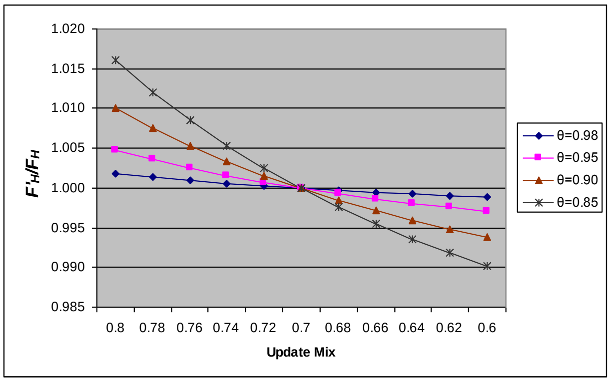

Fig. 4. The freshness of UNAP vs. OAP(θ).

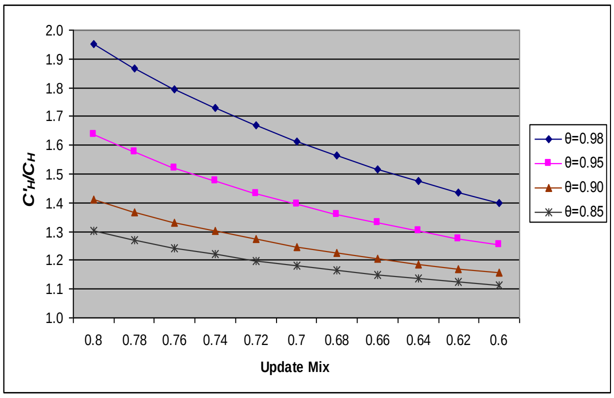

Fig. 5. The cost of UNAP vs. OAP(θ).

5.3. OAP(θ) vs. Individual optimal adaptive policy

In this experiment, we compare OAP(θ) with a different Adaptive Policy IOAP. In IOAP, the *Harvester* chooses the Optimal Interval for each R_i, based on the overall *Freshness* desired and the workload on this particular R_i, independently, rather than relative to the workload on all *Repositories*. This policy is simpler, since the *Harvester* would not need to recompute the optimal intervals each time he receives an update in the workload of one of the *Repositories*, as is the case in the OAP(θ). IOAP results in same *Freshness* as OAP(θ) but different costs, so we compare the cost of both policies for different *Freshness* thresholds θ.

Fig. 6 shows the ratio of the cost of IOAP C'_H to the cost of OAP(θ) C_H while varying θ from 0.5 to 0.95. Results are shown for five cases representing different workload mixes of T1, T2, T3 and T4. In the first four cases, case *i* represents a mix of a majority (70%) of *Repositories* following type T_i, while 30% of the *Repositories* are uniformly distributed among the three other types. The fifth case represents a uniform mix of the *Repositories* between the different four types.

Results shown in fig. 6 show that when the *Repositories* are evenly distributed between the different types of workloads, the IOAP incurs a higher cost ranging from 3% to 18%, with higher overhead for higher θ. When majority of *Repositories* are of type T2, IOAP behaves very badly with degradation reaching 36%. A majority mix of T1 or T3 show similar behavior as the even mix while T4 is the least sensitive.

The above experiments show that OAP(θ) captures the different mixes of workload and adjusts itself such that it provides major improvement over other policies in the cost, given a required threshold of *Freshness*.

It is expected that the performance of OAP(θ) is dependent on the estimates of λ_i. However, we can show that OAP(θ) is insensitive to the variations of λ_i as long as the actual λ_i deviates from the estimate of λ_i by a value of $\pm\delta\lambda_i$. That is in the variations of the actual arrival rate, the amount of $+\delta\lambda_i$ is equal to $-\delta\lambda i$. For the cost, C_H, eq. (2) shows that the second term is independent of the actual λ_i.

The first term, $C_u \sum_{i=1}^{M} \lambda_i$ is a summation of actual λ_i. Since $|+\delta\lambda_i| = |-\delta\lambda_i|$, then the total cost incurred by the variation of actual λ_i would be equal to 0. Similarly, for the *Freshness*, from eq. (3) it is clear that *Ij* are independent of the actual arrival rate since the *Harvester* computes *Ij* based on the estimates of λ_i, not the actual. The term $\sum_{j=1}^{M} \lambda_j I_j$, which depends on actual λ_i would lead to $\lambda_i \pm \delta\lambda_i$ canceling each other.

6. Alternative cost metric

In this section, we introduce a different perspective of viewing the *Freshness* and the Cost C_H, the Combined Cost CC_H. The Combined Cost represents the combination of the loss resulting from the Staleness of data and the communication and processing overhead on the Harvester. That is, $CC_H = \alpha *$ *Staleness* + C_H, where α is a normalization factor.

Fig. 7 plots the Combined Cost CC_H against different values of *Freshness* for various workload mixes for α=10,000. The results show that choosing a small value for *Freshness*, although would result in lower C_H, it leads to a high CC_H due to the loss incurred from the staleness of the data. While a very high value of *Freshness*, although reduces the staleness of the data, it incurs a very high cost C_H and hence would result in a high CC_H. The curves suggests to the managers of the Harvester, the *Freshness* which would result in the optimum Combined Cost.

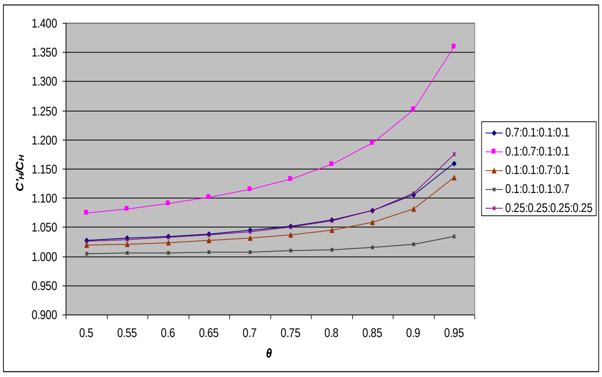

Fig. 6. The Cost of IOAP vs. OAP(θ) varying θ.

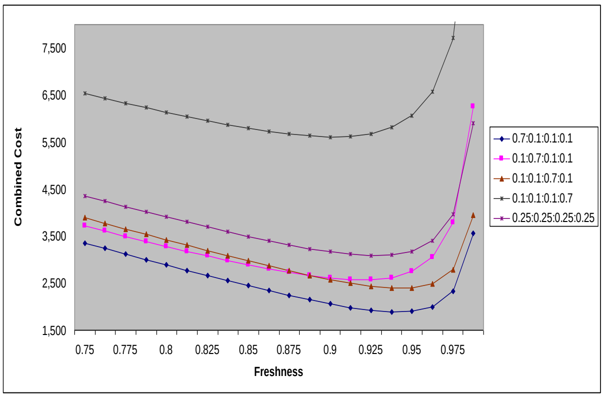

Fig. 7. Varying the *Freshness* for different workload mixes, with α=10,000.

Actually, the value of α can be viewed as a representation of the priority of the *Freshness* relative to the Cost C_H, with higher values of α, leading to higher values of *Freshness*. To generalize, we introduce the factor α_i for every *Repository* R_i, denoting how important the *Freshness* for R_i is. Therefore, we can formulate the problem as to minimize the Combined Cost CC_H and the *Minimum Combined Cost* Min_CC_H will be:

$$Min\_CC_H = \sqrt{\frac{2C_p T}{\sum_{j=1}^{M} N_j}} \sum_{i=1}^{M} \sqrt{\alpha_i \lambda_i} + C_u T \sum_{i=1}^{M} \lambda_i$$

Fig. 8 plots Min_CC_H along with the corresponding actual cost C_H and the *Freshness* at the *Harvester* while varying the factor α for a workload mix where 70% of the *Repositories* belonging to Type T4 and the remaining 30% distributed evenly between Types T1, T2 and T3. It is shown that as α increases the optimal Combined Cost results in an increase in the *Freshness* at the expense of a corresponding increase in the cost C_H. Therefore, the factor α acts as a regulator in the system, determining at runtime the adequate level of *Freshness* that would realize the balance between an acceptable level of *Staleness* of the data and an acceptable overhead that we are ready to pay. This tool enables the administrators at the *Harvester* to tune the desired level of *Freshness* against the Cost.

Further, the factor a_i allows us to introduce different priority of *Freshness* for different *Repositories*. Fig. 9 plots the *Min_CC$_H$* with the corresponding *Freshness* and C_H while varying α for the same workload mix. However, for 50% of the *Repositories* of T4, their *a* is set to double the value of the other Repositories. That is, it is desired to double the priority of *Freshness* for those selected *Repositories*. As shown in fig. 9, and comparing it with fig. 8, the curves results in different optimum values of CC_H, with lower global *Freshness* ranging from 6% to 1%, resulting from prioritizing the *Freshness* of the selected *Repositories*, and with a slight increase in the C_H ranging from 1% to 5%.

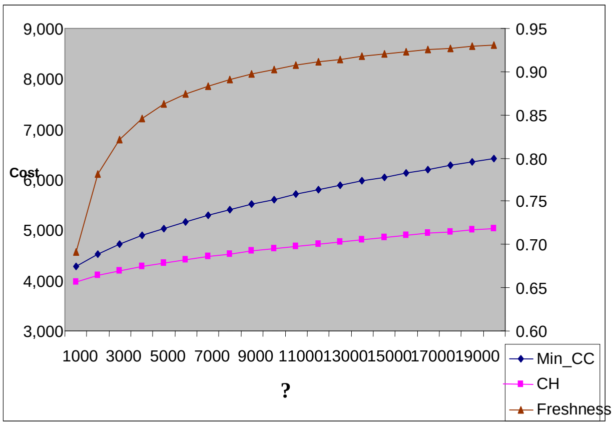

Fig. 8. The *Min_CC$_H$* with the corresponding Freshness and C_H, while varying α.

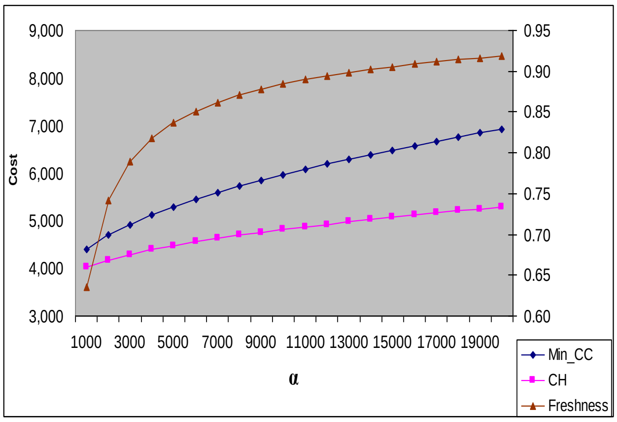

Fig. 9. The *Min_CC$_H$* with the corresponding freshness and C_H, while varying α, with different a_i for different R_i.

The suggested Combined Cost and the solution derived offers a tool that could be used by the managers of the Harvester in order to choose the adequate level of *Freshness* to operate with that would result in the desired balance between the staleness of the data and the incurred cost.

7. Conclusions

In this paper, we introduced an adaptive policy for harvesting OAI-PMH *Repositories* that experience different workload patterns. A framework is provided within which the *Harvester* can decide on the pulling frequency based on a desired level of *Freshness* while incurring a minimum overhead. It has been shown that the adaptive policy reduces the overhead on the *Harvester*, and hence on the *Repositories*, compared to other adaptive or uniform pull-based policies, while offering comparable level of *Freshness*. This is especially obvious when the *Repositories* are heterogeneous and experience different update patterns. Further, we presented an instrument, based on a combined cost metric, that allows choosing an adequate level of *Freshness* to operate at while tuning the overall performance.

References

[1] C. Lagoze, H. Van de Sompel, M. Nelson and S. Warner, "The Open Archives Initiative Protocol for Metadata Harvesting, Version 2.0. http://www.openarchives.org/OAI/openarchivesprotocol.html

[2] K. Maly, M. Zubair and L. Xuemei, "A High Performance Implementation of an OAI-Based Federation Service", In Proceedings of the 11th International Conference on Parallel and Distributed Systems (ICPADS'05) (2005).

[3] K. Maly, M. Zubair, V. Chilukamarri and P. Kothari, "GRID Based Federated Digital Library", In ACM Proc of the 2nd Conf. on Computing Frontiers, Ischia, Italy, May (2005).

[4] arXiv. http://arxiv.org

[5] D-lib Magazine. http://www.dlib.org/

[6] D. Contessa and J. Moreira de Oliveira, An OAI Data Provider for JEMS, In Proceedings of the 2006 ACM

Symposium on Document Engineering, Netherlands (2006).

[7] S. Mclntyre, S. Dennis, S. Uijtdehaage and C. Candler, "A Digital Library for Health Sciences Educators: The Health Education Assets Library (HEAL)," in Proc. of 4th ACM/IEEE Joint Conference on Digital Libraries, June (2004).

[8] Y. Fu and J. Mostafa, "Integration of Biomedical Text and Sequence OAI Repositories, in Proc.", of 4th ACM/IEEE Joint Conference on Digital Libraries, June (2004).

[9] N. Vemuri, R. Shen, S. Tupe, W. Fan and E. Fox, "ETANA-ADD: An Interactive Tool for Integrating Archeological DL Collections, in Proc", of 6th ACM/IEEE Joint Conf. on Digital Libraries, North Carolina, June (2006).

[10] J. Bollen and H. Herbert Van de Sompel, An Architecture for the Aggregation and Analysis of Scholarly Usage Data, in Proc. of 6th ACM/IEEE Joint Conference on Digital Libraries, North Carolina, pp. 298-307, June (2006).

[11] J. Hunter, I. Khan and A. Gerber, "HarvANA – Harvesting Community Tags to Enrich Collection Metadata", in Proc. of 8th ACM/IEEE Joint Conference on Digital Libraries, Pittsburg, Pennsylvania (2008).

[12] F. McCown, M. Nelson, M. Zubair and X. Liu, "Search Engine Coverage of the OAI-PMH Corpus", IEEE Internet Computing, April (2006).

[13] M. Bouzeghoub and V. Peralta, "A Framework for Analysis of Data Freshness", In Proceedings of the international Workshop on Information Quality in Information Systems ACM IQIS, Paris (2004).

[14] J. Cho and H. Garcia Molina, "Synchronizing a Database to Improve Freshness", In Proc. of the 2000 ACM SIGMOD Intl. Conf. on Management of Data, Dallas, TX, May (2000).

[15] J. Cho and H. Garcia Molina, "Estimating Frequency of Change, ACM Transactions on Internet Technology", 3, pp 256-290 (2003).

[16] A. Labrinidis and N. Roussopoulos, Exploring the Tradeoff between Performance and Data Freshness in Database-driven Web Servers, The VLDB Journal, 13, 3, pp 240-255 (2004).

[17] L. Bright, A. Gal and L. Raschid, "Adaptive Pull-Based Policies for Wide Area Data Delivery", ACM Transactions on Database Systems, Vol 31, No. 2, pp 631-671, June (2006).

[18] X. Liu, K. Maly, M. Zubair and M. Neslon, "Repository Synchronization in the OAI Framework", in Proc. of 3rd ACM/IEEE Joint Conference on Digital Libraries, June (2003).

[19] X. Liu, K. Maly, M. Zubair and M. Nelson, Arc – An OAI Service Provider for Cross-Archive Searching, in Proc. of 1st ACM/IEEE Joint Conf. on Digital Libraries, June (2001).

[20] G.B. Thomas, Jr. Calculus and analytic geometry, Addison-Wesley, 4th edition (1969).

[21] V. Jacobson, "Congestion Avoidance and Control", in Proc. of ACM SIGCOMM, Stanford, CA, pp. 314-329 (1988).

Received October 20, 2008
Accepted December 16, 2008