Source Update Capture in Information Agents

Naveen Ashish*, Deepak Kulkarni and Yao Wang
NASA Ames Research Center
MS 269/3 Moffett Field CA 94035
{ashish, kulkarni, yxwang}@email.arc.nasa.gov

Abstract
In this paper we present strategies for successfully *capturing* updates at Web sources. Web-based information agents provide integrated access to autonomous Web sources that can get updated. For many information agent applications we are interested in knowing when a Web source to which the application provides access, has been updated. We may also be interested in capturing all the updates at a Web source over a period of time i.e., detecting the updates and, for each update retrieving and storing the new *version* of data. Previous work on update and change detection by polling does not adequately address this problem. We present strategies for intelligently polling a Web source for efficiently capturing changes at the source.

1 Introduction

An important issue with internet information agents is that of addressing the problem of updates at the remote Web sources being integrated. Information agents (Cohen 2000; Knoblock, Minton et al. 2001; Barish and Knoblock 2002; Doan and Halevy 2002; Kambhampati, Nambiar et al. 2002; Zadorozhny, Raschid et al. 2002) and other Web-based information extraction and integration systems (Davulcu, Yang et al. 2000; Kushmerick 2000; Byers, Freire et al. 2001; Popa, Velegrakis et al. 2002) provide integrated access to data residing in different Web sources. These Web sources are autonomous and the data on the Web pages at these sources may change. For performance optimization, information agents often cache or materialize data from the remote Web sources locally (Adali, Candan et al. 1997; Ashish, Knoblock et al. 2002). When updates or changes occur at Web sources, the cached data becomes inconsistent with the original data. To avoid providing the user with stale or inconsistent data, the information agent must update the cache as changes take place at the original Web sources. The information agent may also require access to the different updated *versions* of data at a Web source over a period of time. For instance the main headline story at the CNN news site (www.cnn.com) gets updated every hour or so (the same news story may get updated or a different news item appears as the headline news) and an information agent may require access to all the different headline news stories [we refer to the distinct data items (i.e., stories) as *versions*] that appeared as headline news over a particular day. We use the term *capture* for the process of detecting an update and then retrieving and storing the new updated version of the data from a source. The information agent may also be monitoring (Barish and Knoblock 2002) a source (via wrappers) and want to be notified when an update has taken place.

The time (and frequency) of changes at many Web sources are not known in advance. As a result, the information agent must poll the Web source(s) to check for updates and changes. To minimize the probability of missing an update we must poll the sources very frequently. However this high polling frequency may not be feasible due to limited network and computational resources. In fact many sources would not allow polling the source at a high frequency as this causes an undesirable load on their Web server. In this paper we present the initial results of our work in progress on capturing changes at a Web source while polling the source only a limited number of times. Our approach is based on our observation of regularities of update times at many autonomous Web sources.

The problem of detecting changes at a source and synchronizing the local copy has been studied in many contexts such as Web data sources, Web proxy servers, Internet crawlers and client-server database systems. (Cho and Garcia-Molina 2000) describes an approach to refreshing the local copy of an autonomous data source to keep the copy up-to-date. (Cho and Ntoulas 2002) presents a sampling-based strategy for keeping local copies of data up-to-date in a World Wide Web or data warehousing environment. (Barish and Obraczka 2000) presents a survey of a variety of caching techniques for the World Wide Web. (Bright and Raschid 2002) presents a Web caching approach where a trade off can be made between the recency of the retrieved information versus the latency to retrieve it. Finally there is work on synchronizing updates in data warehousing (Labrinidis and Roussopoulos May 2000) and in client server database system (Gal and Eckstein 2001) environments. The above efforts have provided

* Naveen Ashish is with the USRA Research Institute for Advanced Computer Science at NASA Ames.

approaches for optimizing various important aspects in synchronizing cached data such as minimizing the "age" of objects (i.e., ensuring that the data is refreshed very soon after it is updated), maximizing the average "freshness" (i.e, ensuring that most of the data is consistent with that in the original source) etc. However, an important problem that has not been addressed by existing approaches is that of capturing all the changes over a period of time. As another example, a Web source that we have extensively studied is a source in the aviation domain – the Digital Automatic Terminal Information services (D-ATIS) messages published at the ARINC Website[1] where air traffic messages are published at the rate of 1-2 messages per hour. A new message overwrites the existing message at the Web site. In one of our applications, the information agent requires access to all the different messages (versions) published over a particular day. In this application, the information agent provides integrated access of the ATIS data with other aviation related data sources (such as radar data, weather data etc.) and some typical queries (performed by aviation safety analysts) require access to all the distinct ATIS messages over an entire day. Capturing all versions of data from a Web source is also important in archival applications such as Web archive (Cho 2003) and the Wayback machine (http://www.archive.org/) where we wish to archive all the different versions of an entire Web source as it changes over time. Existing work on polling and change detection addresses issues such as optimizing the age or freshness of cached data items but does not provide a way to effectively capture changes at a Web source while polling it a limited number of times. In this paper we present an approach to capturing all (or the maximum possible) updates at a Web source over a period of time with limited resource constraints i.e., we will poll the source only a limited number of times. Our approach is motivated by and based on our observation that for many sources, though autonomous, the updates not only occur with a regular frequency but also (mostly) at or around certain times or between certain time intervals.

The rest of this paper is organized as follows. In section 2 we formalize the problem and our optimization goal. In section 3 we present our observations of the distributions of updates at Web sources. We then present strategies for polling the Web sources and synchronizing data based on the fact that the update distributions follow regularities at many sources. We also present experimental results supporting the validity of our hypotheses and effectiveness of the approach. Finally in section 4 we discuss on going work and conclusion.

2. Formalizing the Problem

We first define some metrics that will allow us to state our goal of effectively capturing updates formally. The metrics also will be a means to evaluate the effectiveness of various strategies for capturing updates. We then present a formal statement of the change capture optimization problem.

2.1 Metrics

(a) Change Recall: We introduce the *Change Recall* metric, which is a measure of how successful we have been in capturing the changes at a source. Formally, Change Recall is defined as the number of changed items downloaded, over the total number of changed items in a particular time period. For instance if the ATIS source was updated 30 times a particular day (i.e., there were 30 different messages published over the day) and we captured 27 of these, then the Change Recall would be 27/30 = 0.9. Similarly if there were 18 distinct headline stories that appeared as the top story on the CNN Web site and we captured 15 distinct stories, the Change Recall would be 15/18 = 0.83.
Ideally we would want the Change Recall to be 1. This may not be possible given resource constraints, so our goal is to maximize Change Recall.

(b) Freshness and Age
The Freshness and Age metrics were defined in (Cho and Ntoulas 2002). The freshness of a cached data item F is defined as:
F = 1 if the cached data item is up to date
= 0 otherwise

The age A of an object is defined as:
A = 0 if the cached object is up to date
= t - t_u if the cached object is not up to date, where t= current time and t_u = time of last update

2.2 Problem Statement

If we are polling a source for detecting updates, and polling with limited frequency, the particular times at which we poll can significantly affect the Change Recall. For instance, consider again the updates at the ATIS source, where the update times are as shown in Table 1.

1:05, 1:14, 2:04, 2:15, 3:03, 3:14 ….

Table 1. Update Time Log

Let's say that we have the constraint that we can poll the source only at most 2 times per hour. A naive strategy of polling the source once at the turn of every hour and once again at 30 min past the hour (i.e., poll at 1pm, 1:30pm 2pm, 2:30pm .. ..) would cause us to miss half the updates and result in a poor Change Recall of about 0.5. A more intelligent strategy would be to poll the source at 5 minutes past and 15 minutes past the hour (1:05, 1:15, 2:05, 2:15 ...). With this strategy we would capture almost all the updates and achieve a Change Recall of close to 1.0 The key problem is thus of

[1] http://www.arinc.com/products/voice_data_comm/d_atis.html

deciding at what times to poll a source such that the Change Recall is maximized. We define this formally.

Definition: *Polling Strategy*
A "polling strategy" is defined as a tuple <T, S> where T is a time period (such as an hour, day month etc.) over which the polling times repeat in a cycle and,
$S = \{S_1, S_2, ..., S_m\}$ is a set of times at which we poll the source within each time period T.
So a strategy defined by <hour, {5,15,45}> implies that in each hour we poll 3 times, at 5 minutes past, 15 minutes and 45 minutes past the hour.

We now state the Change Recall optimization problem formally:

Given:
O = a Web source
T = time period
N = maximum number of times we can poll S in the time period T
H = previous history of updates at the source
Generate:
A polling strategy <T,S> such that the expected Change Recall is maximized, where we poll at most N times in the time period T.

Note that in certain applications we may also be interested in optimizing other metrics i.e., minimizing the average age or maximizing the freshness. A polling strategy that maximizes Change Recall, can also be used to minimize the average age of cached data items and in fact performs better than existing strategies in many cases !

3. Polling Strategy

We make use of the historical data for update times at a Web source to estimate the probability of missing updates with any polling strategy. Like existing approaches, our approach is based on the assumption that the historical pattern of updates (over an appropriate time period) at a Web source is a good predictor of the future pattern of updates at that source. We thus first talk about our observations of update time distributions at Web sources and then present approaches for generating an optimal polling strategy.

3.1 Update Time Distributions

While a source may change anytime, the times of updates at many sources do follow certain regular distributions . In (Cho and Ntoulas 2002) it was shown that the *Poisson* process effectively models change at the Web sources they sampled. However there is a difference in behavior between all Web pages of the entire Web and a particular set of Web pages. While hundreds of millions of Web pages in an entire set can be considered to have been changed by a random process on average, for a particular set of pages as well as different scales of study, the randomness of the change occurrences has to be addressed before we can make confident predictions about the polling. While the Poisson process may model updates of web sources in general, specific sources may exhibit update distributions that are distinctly different. It is our observation that for many sources we can use more accurate models to fit the distribution of update times at a Web source. For instance for the ATIS source a log of update times for a particular airport is shown in Table 1. Most of the updates occur around 5 min past or 15 min past the hour. This distribution is consistent across several months. Or consider a source such as Hollywood.com. The "new movies this week" [2] page changes once a week, mostly on the thursday of the week, announcing new movies releasing on Friday or the weekend. The fact that a source gets updated according to some such distribution and knowledge of this distribution can be exploited to come up with a smart strategy for polling that source. For instance from the observation that for the above ATIS messages, there are mostly 2 messages published per hour, the first by 5 min past the hour and the second by 15 min past the hour , we could poll the source at 5 min and 15 min past the hour and we would capture most of the updates. For the hollywood.com. source we could just poll once a week, every thursday when the movie screenings change. Of course many update distributions will not be that simple.
Our second observation is that the distribution of updates of a web page would depend on semantics of the web page itself. For example, the likelihood of updates to the CNN.com home page in a short time are higher if the page is reporting a breaking story or a very rapidly changing event.

So the update distribution is indeed helpful in deciding a good polling strategy. The problem is to come up with an approach to generate such a strategy automatically given the update distribution. We now describe two alternative approaches to generating the optimal polling strategy.
(1) Empirical Approach: We can systematically consider all possible polling times for an interval of interest and can use the historical information to compute how many changes would have been missed if we had used particular polling strategy. If this can be done in a computationally efficient manner, the approach can be used to find an optimal strategy.
(2) Theoretical Modeling Approach: We can model the update patterns using an appropriate probability distribution and do analysis based on this probability distribution to infer the best polling strategy. This approach has been taken in previous work and is computationally efficient.

3.1.1 Empirical Approach

Suppose $\{T_1, T_2, \ldots T_s\}$ is the set of time points at which

[2] http://www.hollywood.com/movies/this_week.asp

one would consider polling in the interval T, where $T_{(i+1)} > T_i$. For example, {1, 2, … 60} (minutes) is the set of time points at which one would consider polling ATIS data in a typical hour (T = 60 minutes). We define a probability function, NumMisses(T_i, k) as the number of missed updates in the interval (T_i, T_{i+k}) if we poll at T_i and T_{i+k} and additionally poll l times in the interval (T_i, T_{i+k}) using an optimal strategy. Let PollingSet(T_i,k,l) be the corresponding set of time points at which one would poll using the optimal strategy. Also, N is the maximum number of times we can poll in the interval T as defined in the problem statement. NumMisses(T_i, k, 0) can generally be derived from historical data for all possible values of i and k.

We propose the following algorithm for computing the NumMisses function efficiently for all possible l of interest:

```
for l = 1 to (N-2)
 for i = 1 to s
  for k = 1 to s
   NumMisses(T_i,k,l) = Min j=1 to (k-1)(NumMisses(T_i,j,1)
                + NumMisses(T_i+j , k-j, l – 1))

   Let jmin be value of j for which above
   expression is minimum

   PollingSet(Ti,k,l)=PollingSet(Ti,k,1) U
                  PollingSet(T_i+jmin,k-jmin,l – 1)
  End for
 End for
End for
```

The set of time points in the best polling strategy in the interval T is PollingSet(Ti_{min}, k_{min}, N-2) where i_{min} and k_{min} are i and k for which NumMisses(T_i, $_k$, N-2) is minimum.

In the above algorithm, NumMisses is computed for o(s*s*N) input values. Each computation of NumMisses(T_i,k,l) and PollingSet(T_i,k,l) can be done in at most s steps. So the above algorithm can compute NumMisses(T_i, k , l) and PollingSet(T_i,k,l) using at most s ** 3 * N computations for all i, k and l. As the final step involves computing the minimum of s ** 2 values, this algorithm can compute the best polling strategy in O(s ** 3 * N) steps.

Using the above algorithm we thus systematically consider all possible combinations of polling times given the polling frequency and determine which is the best polling strategy. This approach is likely to be practical for small values of s, but not for large values of s. In the next section, we will describe another approach that can be used even for large values of s.

3.1.2 Theoretical Modeling Approach

Generating the optimal polling strategy by exhaustive search may be prohibitively expensive in many cases i.e., when there are a very large number of possible combinations of possible polling times and searching through the entire space is expensive. We present an efficient algorithm for determining a near optimal polling strategy. The algorithm is based on the assumption that the probability density function representing the update probability of a particular data item on a Web source being updated is *uniform* in small time intervals. This assumption is reasonable but not completely accurate for many Web sources. Thus the algorithm is not guaranteed to find an optimal polling strategy but instead finds a near optimal strategy that is almost as good as the optimal strategy.

The algorithm is based on a couple of very elementary characteristics of updates in different kinds of time intervals. First, there may be intervals where only at most one update can occur. There is no need to poll multiple times in such intervals, rather one can poll just once, at the end of the interval. Next, for intervals where two or more updates may occur any time we need to poll as many times as we can. For such intervals, if we know how the probability of missing an update changes as a function of the number of times we poll in that interval, we can systematically determine how many times to poll in each such interval (given a limited number of total polling times).

. There are thus two primary steps in the algorithm:

(i) Find the time intervals where there are zero or at most one updates (we call such intervals *single update intervals*) and assign polls to those intervals appropriately.

(ii) For the remaining intervals i.e., intervals where 2 or more updates may occur (we call such intervals *multiple update intervals*), and assign the remaining polls appropriately.

Consider again the ATIS messages. Let's say at most 3 messages are published (i.e., updated) every hour. We tag these messages (the first, second and third) as A, B and C. Say the update probability distributions (represented by probability density functions) of each of these messages is as shown in Fig 2 which shows a plot of update probability distributions versus time (in minutes). A is updated only sometime between t=5 and t=15 min etc. If we poll only thrice an hour, at t=15,t=35 and t=50 we will capture all the updates A,B and C.

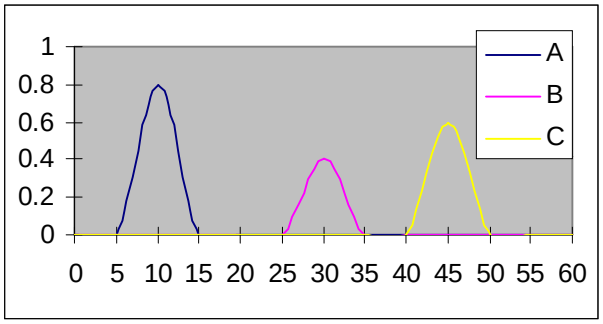

Fig 2. Update Probability Distributions

This is possible because the probability distributions of A,B and C do not *overlap* anywhere. Only at most one message (A,B or C) can get updated in a time interval and we simply poll once at the end of that interval. What if the probability distributions do overlap ? For instance consider a different distribution as shown in Fig 3. Both A and B can get updated between t=10 and t=20 and both B and C can get updated between t=30 and t=35. These are the multiple update intervals. There are also single update intervals. For instance only A may occur between t=5 and t=20 (so we need poll only once at the end of this interval at t=20). Similarly we poll once at t=10, t=30 ,t=35 and t=40. Note that there is a possibility of missing an update in this case. Two or more updates (A and B) could occur between t=10 and t=20 and we will capture only one of them. Also two or more updates (B and C) could occur between t=30 and t=35. So far we have assigned a total of 5 polls per hour. Suppose we could poll more than 5 times. At what times should we poll additionally ? Polling more in a multiple update intervals decreases the probability of missing an update in that interval. We will examine shortly as to how exactly this probability varies with the number of times we poll in the interval. So any additional polls should be assigned to the multiple update intervals. But there could be many such multiple update intervals. So how do we relatively assign the additional polls between these intervals ? For instance in the current example we have 2 multiple update intervals and if we had a total of 5 additional polls we could assign 1 additional poll to the first multiple update interval and 4 to the second or 2 to the first and 3 to the second etc. Which assignment of these minimizes the total probability of missing an update ? Having a model of the update probability distributions of the updates in the different multiple update intervals will allow us to determine the probability of missing updates for various assignments. In general an update distribution may be of any form within an update interval. However for many sources we can approximate the probability density function for an update distribution to be *uniform* in that interval, for intervals that are sufficiently small.

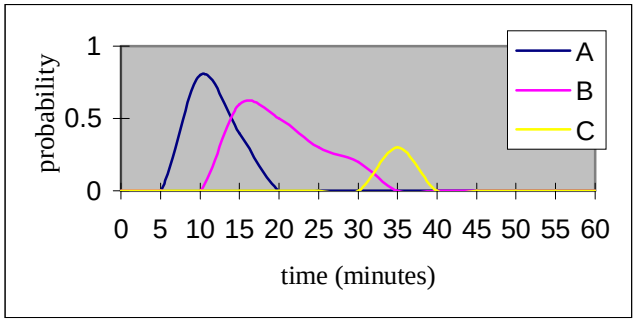

Fig 3. Update Probability Distributions

For such cases, i.e., where approximating the probability distribution as uniform in small intervals is reasonable, we can evaluate the probability of missing updates for different assignments and thus find the optimal assignment. We describe how we do this below. Let's say we have i such multiple update intervals. Suppose we poll K_i times in an interval i. What is the probability of missing an update in the interval i now? We poll at uniform sub-intervals within interval i as shown in Fig 4. We will miss an update in interval i if and only if the two updates occur together in any one of the K_i sub-intervals. The probability of both updates occurring in a particular sub interval is given by:

$(Pr_A\ t/K_i) * (Pr_B\ t/K_i) = Pr_A Pr_B t^2/K_i^2$

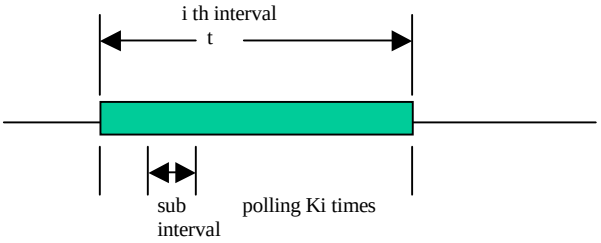

Fig 4. Polling in a multiple update interval.

where Pr_A and Pr_B are the probability densities of A and B in that interval respectively. The probability that two updates occur together in *any* of the Ki sub-intervals is simply:

$K_i*(Pr_A\ t/K_i) * (Pr_B\ t/K_i) = Pr_A Pr_B t^2/K_i$

This expression is of the form $C_i t^2/K_i$ where $C_i=Pr_A Pr_B$ is a constant. Although we have illustrated the above for the case where 2 updates can occur in an interval, the expression representing the probability of missing an update is of the form where 2 or even more updates can occur in an interval.

Now the probability of missing any update in any of the i multiple update intervals is:

$\Sigma^n_{i=1}\ C_i t^2/K_i$

Note that we take all multiple updates to be of equal length i.e., t. If the multiple update intervals are not originally of equal length we can sub divide them into intervals of length of the greatest common divisor of the lengths of the (original) multiple update intervals.

We have to find Ki such that $\Sigma Ki=K$

and $\Sigma^n_{i=1}\ C_i/K_i$

is minimized. This is a well known optimization problem and the minima lies when:

$C_1/K_1^2=C_2/K_2^2 = \ldots\ldots\ C_n/K_n^2$ (condition I)

Thus we simply assign the K_is according to the above equation. The algorithm to find a (near) optimal polling strategy using theoretical modeling can be stated as follows:

1. Find the update probability distributions of the various updates.
2. Find the single update intervals.
   a. Poll once at the beginning and once at the end of each such interval.
3. Find the multiple update intervals.
   a. Assign the remaining polls to the multiple update intervals in the proportion defined by condition I above

While we do not present a proof here, the above algorithm is linear in the number of possible polling points one would consider in an interval.

3.1.3 Experimental Results

We evaluate the effectiveness of our approaches by measuring the Change Recall from the ATIS Web server using various strategies[3]. We used real historical update time data collected over several months from the ATIS Web server and tested the strategies for Change Recall over the actual ATIS source. We evaluate three different strategies:

(i) A naïve uniform strategy where given N polls in a time period (an hour in this case) we simply poll N times at uniform intervals in the time period.

(ii) An optimal polling strategy generated by exhaustive search.

(iii) A near optimal polling strategy generated by theoretical modeling.

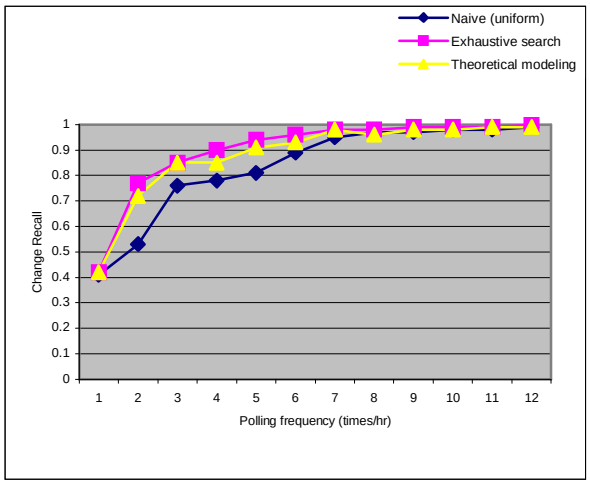

Fig 5. Effectiveness of strategies.

As we can see in Fig 5 above, the optimal polling strategies (both by exhaustive search and theoretical modeling) result in significantly better Change Recall than the naïve uniform strategy. The improvement is more significant when the polling frequency is less. Thus exploiting the update time distribution indeed helps in achieving a better Change Recall versus existing sampling based approaches that would result in Change Recall obtained by the naïve (uniform) approach. Also the theoretical modeling strategy performs almost as well as the exhaustive search strategy. So, this indicates that in cases where the exhaustive search strategy is computationally expensive, finding a near optimal strategy by theoretical modeling is a good alternative. We must note that in some other scenarios the naïve strategy may perform significantly worse. For instance suppose we had an update pattern that was of the form {1:00, 1:05, 1:10, 1:15, 2:00, 2:04, 2:09, 2:15,3:01,3:05 …..}. With a naive strategy with N=4 (i.e., polling at 1:00,1:15,1:30,2:00, …) we would get a very poor Change Recall of ~ 0.25 whereas with the optimal strategies we would get a Change Recall of close to 1.0 when polling 4 times an hour.

4. Work in Progress and Conclusion

In this paper, we introduced change recall as an important metric to be considered in remote data source synchronization. Then, we noted that it is possible to utilize knowledge of specific update probability distributions and their dependence of domain semantics in devising a polling strategy. We discussed two different polling strategies we are using in our applications. Based on preliminary results, these strategies are more effective for capturing changes than existing strategies, which do not focus on the change capture problem in particular and are based solely on the frequency of updates. There are several tasks and issues that we are working on right now, namely:

- More extensively studying the update patterns at a variety of different autonomous Web sources.
- Testing the effectiveness of the polling strategies with many other Web sources.
- Extending the theoretical modeling approach to cases where the uniform distribution approximation is *not* reasonable.
- Testing the effectiveness of our strategies in optimizing factors other than change capture, such as age and freshness of cached objects.
- Utilizing the semantics of the data to predict the probability of the next update and incorporating this knowledge in generating the polling strategy.

Besides information agents, our change capture strategies are also applicable to a variety of other systems such as Web crawlers, Web proxy server caches and Web archiving systems where it is important to synchronize data cached from autonomous sources.

[3] At this point we have only evaluated the strategies with the ATIS Web source. However by the time of the workshop we expect to provide evaluation results with several other Web sources.

References

Adali, S., K. S. Candan, et al. (1997). Query Caching and Optimization in Distributed Mediator Systems. Proceedings of the ACM SIGMOD International Conference on Management of Data, Tucson, AZ.

Ashish, N., C. A. Knoblock, et al. (2002). "Selectively Materializing Data in Mediators by Analyzing User Queries." International Journal of Cooperative Information Systems (IJCIS) **11**(1-2): 119-144.

Barish, G. and C. Knoblock (2002). An Expressive and Efficient Language for Information Gathering on the Web. Proceedings of the Sixth International Conference on AI Planning and Scheduling (AIPS-2002) Workshop, Toulouse, France.

Barish, G. and K. Obraczka (2000). World Wide Web Caching:Trends and Techniques. IEEE Communications Magazine.

Bright, L. and L. Raschid (2002). Using Latency-Recency Profiles for Data Delivery on the Web. Proceedings of the 28th VLDB Conference, Hong Kong, China.

Byers, S., J. Freire, et al. (2001). Efficient Acquisition of Web Data through Restricted Query Interfaces. Poster Proceedings of the Tenth International World Wide Web Conference, WWW10, Hong Kong, China.

Cho, J. (2003). Web History and Evolution Archiving (WHEN).

Cho, J. and H. Garcia-Molina (2000). Synchronizing a Database to Improve Freshness. Proceedings of 2000 ACM International Conference on Management of Data (SIGMOD), Dallas.

Cho, J. and A. Ntoulas (2002). Effective Change Detection Using Sampling. Proceedings of the 20th VLDB Conference, Hong Kong, China.

Cohen, W. (2000). "WHIRL: A Word-based Information Representation Language." Artificial Intelligence **118**(1-2): 163-196.

Davulcu, H., G. Yang, et al. (2000). Computational Aspects of Resilient Data Extraction from Semistructured Sources. Proceedings of the Nineteenth ACM SIGMOD SIGACT-SIGART Symposium on Principles of Database Systems, Dallas, TX.

Doan, A. and A. Halevy (2002). Efficiently Ordering Query Plans for Data Integration. Proceedings of the International Conference. on Data Engineering (ICDE), San Jose, CA.

Gal, A. and J. Eckstein (2001). "Managing Periodically Updated Data in Relational Databases: A Stochastic Modeling Approach." Journal of the ACM **48**(6): 1141-1183.

Kambhampati, S., U. Nambiar, et al. (2002). Havasu: A Multi-Objective, Adaptive Query Processing Framework for Web Data Integration. Tempe, ASU CSE TR-02-005.

Knoblock, C. A., S. Minton, et al. (2001). "The Ariadne Approach to Web-based Information Integration." International Journal of Cooperative Information Systems (IJCIS) Special Issue on Intelligent Information Agents: Theory and Applications **10**(1/2): 145-169.

Kushmerick, N. (2000). "Wrapper Verification." World Wide Web Journal **3**(2): 79-94.

Labrinidis, A. and N. Roussopoulos (May 2000). WebView Materialization. Proceedings of the ACM SIGMOD International Conference on Management of Data, Dallas, TX.

Popa, L., Y. Velegrakis, et al. (2002). Translating Web Data. Proceedings of the International Conference on Very Large Databases (VLDB), Hong Kong, China.

Zadorozhny, V., L. Raschid, et al. (2002). Efficient Evaluation of Queries in a Mediator for WebSources. Proceedings of the ACM SIGMOD Conference on Management of Data, Madison, WI.