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.

capture almost all the updates and achieve a Change Recall of close to 1.0 The key problem is thus of 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 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" page (http://www.hollywood.com/movies/this_week.asp) 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.

at the end of this interval at t=20). Similarly we poll once at t=10, t=30 ,t=35 and t=40.

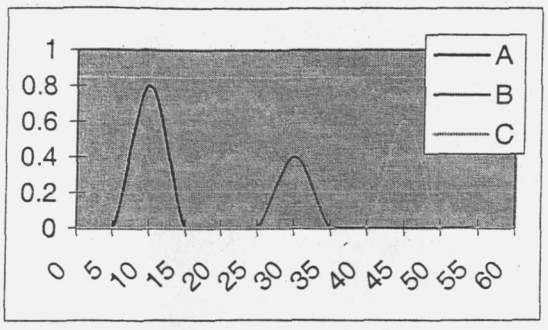

Fig 2. Update Probability Distributions

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 ? It is possible to determine this assignment (and in a computationally efficient manner) under an important assumption about the update probability distribution , which we state below.

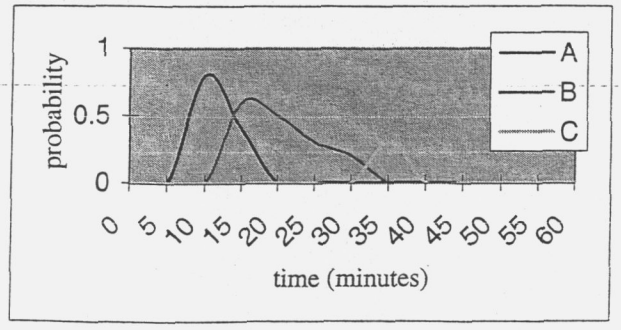

Fig 3. Update Probability Distributions

Assumption: *Within* any multiple update interval i, the probability of update of any individual process is *uniform* throughout that interval. In other words the probability density function representing the update probability of any update process is a constant within any interval[1].

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

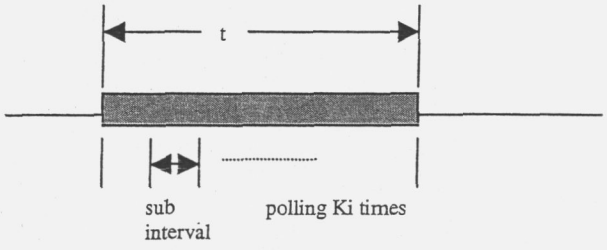

Fig 4. Polling in a multiple update interval.

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:

---

[1] This assumption is reasonable for relatively small intervals (such as intervals of 5 min in the ATIS case).

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.