Learning the Grammar of Distant Change in the World-Wide Web

Dirk Kukulenz

University of Luebeck, Institute of Information Systems,
Ratzeburger Allee 160, 23538 Luebeck, Germany
kukulenz@ifis.uni-luebeck.de

Abstract. One problem many Web users encounter is to keep track of changes of distant Web sources. *Push services*, informing clients about data changes, are frequently not provided by Web servers. Therefore it is necessary to apply intelligent *pull strategies*, optimizing reload requests by observation of data sources. In this article an adaptive pull strategy is presented that optimizes reload requests with respect to the 'age' of data and lost data. The method is applicable if the remote change pattern may approximately be described by a piecewise deterministic behavior which is frequently the case if data sources are updated automatically. Emphasis is laid on an autonomous estimation where only a minimal number of parameters has to be provided manually.

1 Introduction

Access to the World-Wide Web is currently mainly achieved by keyword-based search, e.g. through Web search engines, or by browsing, which denotes the process of accessing Web objects by the use of hyperlinks. Both strategies assume the Web to be a static information source that doesn't change during a search. However frequently the dynamics of the Web is the focus of the actual search. If a user wants e.g. to keep track of news, stock prices or satellite images, 'static' search strategies like browsing or query search are not suitable.
Basically there exist two strategies to acquire this knowledge, denoted as the *push* and the *pull* strategy [13]. The push strategy implies that a data source itself informs the client about the times of changes. An investigation concerning the push model is presented in [16]. However a push service is difficult to realize in a heterogenous information system as the World-Wide Web. In the pull model it is necessary for a client to predict remote changes in order to initiate reload operations. In order to predict changes it is first necessary to acquire information about a data source, i.e. to detect changes. Basically two strategies are available for this purpose. One is to reload the data source periodically and to compare previous and current data objects. A second strategy in order to detect data changes is provided by the HTTP protocol [7]. Different strategies were developed to acquire optimal cache behavior based on http-headers [9], [19]. Since according to [3] 'last-modified'-headers are only available in 65% of Web pages in

this article we apply the first method in order to register page changes. The main problem discussed in this article is to estimate and to predict change behavior based on an observed history of changes acquired with the above method with a certain precision. The goal is to find an optimal reload policy of remote sources according to quality parameters like amount of lost data and age, that will be introduced in detail in section 2. The reload policy has to minimize costs with respect to network load and the number of Internet connections. This aspect is especially important if a large number of remote data objects is considered.

1.1 Related Research

The problem of realizing continuous queries over data streams seems to be similar to the described problem. However in this field the data stream is usually assumed to be already available [2] while the focus in this article is to make the data stream available, i.e. to optimize the data stream itself. *Web prefetching* is an issue well known from the field of intelligent agents and network optimization [4]. One method applied in this field is the use of Markov processes [8]. In contrast to this article the main concern is usually to predict different data objects from recorded sequences of Web accesses, but not to optimize the times of requests. The consideration of dynamical aspects of the World-Wide Web is a main issue for the development of Web crawlers, Web archives and Web caches. With respect to these applications, diverse statistical methods were presented in order to optimize requests for remote data sources. In [1] an introduction into search engine design and into the problems related to optimal page refresh is given. In [6] aspects of optimal (Web-)robot scheduling are discussed from a theoretical point of view, modeling page change intervals by independent Poisson processes. An empirical analysis of Web page dynamics with respect to statistical properties of intervals between data changes is presented in [3]. One main result is that the distribution of intervals between changes is similar to an exponential distribution. In [20] the problem of minimizing the average level of 'staleness' for web pages and the development of an optimal schedule for crawlers is analyzed based on the assumption of independent and identically distributed time intervals. Assuming a Poisson process for intervals between update times, in [5] an optimized reload frequency estimation is presented.

1.2 Contribution

Many data objects in the Web are updated according to a well defined pattern, e.g. every working day, not at night and not at weekends. For similar update patterns, the assumption of an independent and identical distribution of consecutive time intervals, which is applied in many statistical approaches, is not suitable. In this article we present a reload policy which is optimized for similar 'quasi regular' update patterns. In contrast to a previously presented algorithm for the realization of continuous queries [15], [14] the presented reload policy is adaptive and may register, if the remote change behavior changes and adapt automatically. The estimation depends on specific initial values for the estimation process. One focus in this article is to learn some of these parameters automatically in order to simplify the application of the algorithm for a user. The sequence of time intervals between updates is piecewise approximated by a special kind of a regular grammar we will denote as a *cyclic regular grammar*. Regular grammar inference is a problem well-known from machine learning [10], [12]. Many algorithms were presented to learn regular grammars from positive and negative examples [17], [18]. The cyclic-regular case as defined in section 2 is simpler than the general regular inference problem and may be computed efficiently. We will apply the grammar estimation for the development of a new adaptive reload policy in order to optimize local copies of remote sources.

After a description of the applied model in section 2, in section 3 we propose an algorithm for the grammar estimation and it is shown how this knowledge may be applied to find optimal reload times. The estimation algorithm is illustrated by an example in section 4. In section 5 we present experiments concerning the application of the regular-grammar based reload policy to simulated and real Web data and in section 6 a summary is given and further aspects are discussed.

2 The Model

Let $u_i \in \mathbb{R}^+$ denote the points in time at which the i^{th} update of a Web data object occurs, where $0 \leq u_1 \leq u_2 \leq u_3 \leq u_4 \leq \dots u_n \leq T \in \mathbb{R}^+, n \in \mathbb{N}$. The interval of time between the $i-1^{th}$ and i^{th} update will be denoted by $t_i := u_i - u_{i-1}, i \in \mathbb{N}$. Let $a_1, a_2, \dots a_m \in \mathbb{R}^+$ denote the points in time where reload operations are executed, where $0 \leq a_1 \leq a_2 \leq a_3 \dots \leq a_m \leq T$. For $t \in \mathbb{R}^+$ let $N^u(t)$ denote the largest index of an element in the sequence u that is smaller than t, i.e. $N^u(t) := \max\{n | u_n \leq t\}$. Let $A^u(t) \in \mathbb{R}^+$ denote the size of the time interval since the last update, i.e. $A^u(t) := t - u_{N^u(t)}$. If t is the time of a reload ($t = a_i$ for $i \leq m$), we denote $A^u(t)$ as the *age* of a_i. The age of a local copy denotes how much time since the last remote data update has passed and thus how long an old copy of the data was stored although a new version should have been considered.[1]

Let $Q := \{t_j | j \leq n \in \mathbb{N}\}$ denote the set of time intervals between updates. We assign a symbol $s_i, i \in \mathbb{N}_{\leq n}$ to every element of Q. We call the set of symbols $\Delta := \{s_i | i \leq n\}$ the *alphabet* of the sequence u.

Let S denote a starting symbol, let $r_1, r_2, \dots r_n$ denote terminals and the symbols $R_1, R_2, \dots R_n$ non-terminals. In the following we refer to a regular grammar Γ corresponding to the non-deterministic finite automaton in figure 1 as a *cyclic regular grammar*. In figure 1, 'R_0' is a starting state which leads to

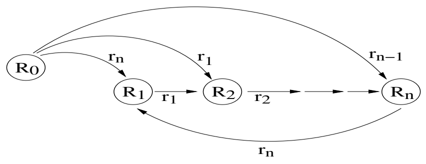


Fig. 1. Nondeterministic automaton corresponding to the grammar $(r_1 r_2 \dots r_n)^\circ$

[1] If a local copy is used as an information source for users, in the respective period of time these users receive the old instead of the new data.

any of n states $R_1, \dots, R_n$. After this, the list of symbols is accepted in a cyclic way. Every state is an accepting state. To abbreviate this definition we will use the notation: $(r_1 r_2 ... r_n)^\circ := \Gamma$.

The first problem in the following is to describe a sequence of symbols $s_1, s_2...$ corresponding to time intervals by a cyclic regular grammar of minimal size. The second problem is to predict further states of the automaton and to find optimal reload times. Finding the optimum means that after each update of the remote data source, the data should be reloaded as soon as possible, i.e. the sum of ages $sumage := \sum_{i=1}^{m} A^u(a_i)$ has to be minimal. The number of reloads should be as small as possible. No change of the data source should be unobserved. The number of lost (unobserved) data objects will be denoted as $\sharp loss$ in the following.

3 Algorithm for Adaptive Reload Optimization

3.1 Overview, Definitions

The presented algorithm for adaptive reload optimization consists mainly of four components.

A **first component** is responsible for the estimation of time intervals between successive updates. For this purpose the data source is downloaded after constant time periods. One main problem in this context is to find an adequate time period between reload operations in order to capture all remote updates as precisely as possible. In the subsequent section we present a method that is based on the observation of changes in a number of consecutive intervals.

A **second component** of the algorithm determines a sequence of symbols. A number of intervals corresponds to the same symbol. As a motivation for this definition it may be assumed that in many cases in the Web remote update operations are executed by daemon processes after certain time periods. However due to a server and network delay, the time intervals that are actually registered on the client side are slightly different. A symbol represents a cluster which combines these similar intervals, providing a maximal and a minimal length estimation for the interval. A *symbol* is a 3-tuple $s = (i, max, min)$ consisting of a unique identifier i and two length parameters $s.max$ and $s.min$.

A **third component** of the algorithm is responsible for the estimation of a grammar that represents the update behavior of a remote source. A *hypothesis* $H = (\Gamma, s)$ is a 2-tuple consisting of a cyclic-regular grammar Γ and the current state s of the associated finite automaton, according to the enumeration of states in figure 1. In every step of the algorithm after the detection of a symbol, the *default hypothesis* is added to the set of hypotheses. Taking the sequence of symbols registered by the system so far $(r_{i_1}, r_{i_2}, \dots r_{i_p})$, the *default-hypothesis* is the cyclic regular grammar $(r_{i_1} r_{i_2} \dots r_{i_p})^\circ$ with the corresponding automaton being in the state '1' according the enumeration of states in figure 1. This automaton accepts the sequence of input symbols. The last symbol is accepted by a transition from the last state to state '1'. A *prediction* of a hypothesis which is not in the start state (R_0) is the symbol, generated by a transition to the (unique) state following the current state. A *proceed* operation applied to a hypothesis H (*H.proceed*) which is not in the start state '0' converts the current state of H to the subsequent state.

A **fourth component** is finally responsible for the application of the estimation result to an optimal reload policy and for the decision when the estimation becomes false and has to be updated.

3.2 Reload Interval Size and Interval Estimation

Figure 2 shows the basic idea of the interval estimation. The data source is reloaded after constant time periods $a_i - a_{i-1}$ denoted as *sampling interval* (*sampsize*) in the following. By detecting two remote changes (update 2 and 3 in figure 2a) in subsequent reload intervals (interval [3,4] and [7,8]) we acquire a maximal and a minimal estimation for the interval length (min and max in figure 2a).

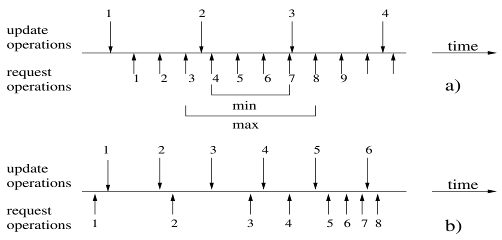


Fig. 2. Determination of a maximal and a minimal estimate for an interval length

An obvious problem with this estimation is that remote changes taking place between two successive reload operations may not be recognized. Therefore it is necessary to find an optimal time interval between successive requests, depending on the remote update behavior. One heuristical approach for this problem is shown in figure 2b. The algorithm is initialized with a reload interval which is known to be too large. After each reload operation the algorithm stores the information whether a change has been detected in the previous interval or not in a queue of size k. This queue contains information about changes being detected by the previous k reload operations. If a new change is detected, the algorithm tests if a previous change is contained in the queue. If this is the case, the reload interval is divided by 2 and the algorithm proceeds. A further question is when to stop with the interval size adjustment, i.e. when the interval size is sufficiently small. One heuristics is to stop the size determination after detection of m successive remote data changes without interval size adjustment.

The reload interval *sampsize* is sufficiently small if the time interval between requests $a_j - a_{j-1}$ is small compared to the time between updates (*sampsize* $\ll \min_i t_i$). It is obvious that the approach above may not lead to optimal results if the variance of update intervals is high. One possible improvement would be to analyses the 'fine' structure after the rough structure (i.e. the large gaps between updates) is known.

```
Symbol-Estimation( newInterval, symbols)

1    for each symbol s in symbols:
2      if newInterval similar to s
         return s
3    define new symbol sn with parameters of newInterval
     add sn to symbols
     return sn
```

Fig. 3. Algorithm component for insertion of new symbols or assignment of new intervals to previously registered symbols by incremental clustering according to a similarity measure

3.3 Symbol Estimation by Incremental Clustering

The previous algorithm generates a sequence of interval estimations some of which are similar. The next step is to cluster similar intervals which are assumed to represent similar remote update intervals. The problem is to find an adequate similarity measure for intervals and to apply it to the interval sequence in order to find the set (and the sequence) of symbols. Figure 3 shows the algorithm component *Symbol-Estimation*, responsible for the learning of new symbols or the assigning of a new interval to a previously registered symbol by incremental clustering of intervals [11]. It depends on the estimation parameters of a new interval (*newInterval*) and the set of previously registered symbols (*symbols*). In step 1 and 2 for each symbol in the set of symbols it is tested if the new parameters of *newInterval* are 'significantly different'. For this purpose the maximal and minimal interval sizes are compared with respect to the sampling size.[2] If this is not true for one symbol, i.e. if the current symbol has already been detected, the respective symbol is returned i.e. the set of symbols isn't changed. If the new interval-parameters are significantly different from all symbols defined so far, a new symbol is inserted into the set *symbols* in step 3. The algorithm is executed for each new detected interval. After the set of symbols has been determined it is easy to assign the previously registered intervals to the symbols by comparison and thereby to compute a sequence of symbols (denoted as *symbolSequence*) from the interval sequence.

3.4 Grammar Estimation

Based on the sequence of symbols detected so far it is now possible to develop hypotheses for the time-based grammar that generated the current symbol sequence. Figure 4 shows the algorithm which takes the sequence of symbols as the input. For every position in the sequence, the whole sequence of symbols ob-

[2] Different strategies are conceivable to compare the intervals; the applied method is described in detail in [15].

```
Grammar-Estimation( symbolSequence)

1    set $hset = \emptyset$
2    for i=1 to $symbolSequence.length()$
3      symbol $s$ := symbolSequence.get(i)
4      add the default-hypothesis to $hset$
5      for each $H \in hset \setminus \{$default-hypothesis$\}$ do
6        if symbol not equal to prediction of $H$
7          delete $H$ from $hset$
8        else
9          apply $H.proceed$
```

Fig. 4. Main algorithm component for the estimation of a cyclic regular grammar

served prior to this position is used to create the *default-hypothesis* (as defined in section 3.1) in step 4 of the algorithm in figure 4. In steps 5 and 6 it is tested for each hypothesis H in the set of hypotheses *hset* if the prediction of H corresponds to the newly observed symbol. If not, the hypothesis is deleted from the set of hypotheses (step 7). If the prediction is consistent with the observation, the state of the hypothesis is increased in step 9.

3.5 Optimized Adaptive Reload Policy

In this section based on the previous estimation a reload policy is presented for optimal data update that is adaptive and may register if the estimation doesn't represent the remote update pattern (figure 6). It is assumed that the alphabet and the grammar have already been determined. We assume that the current state of the automaton is already known. This state may simply be determined by observing remote changes with a high reload frequency until the symbol sequence observed so far uniquely describes the current state of the automaton (step 1, figure 6) as described in detail in [15]. In steps 2 to 16 a loop is executed that determines at first the unique subsequent symbol according to the automaton H in step 4. According to this symbol an optimal reload time is computed in step 4. The choice of the optimal reload time is depicted in figure 5. The subsequent reload time is chosen such that it is supposed to occur before the next remote update, it corresponds roughly to the estimated minimal interval length between two remote updates (e.g. request 3 has to occur before update 2 in figure 5). Thereby it may be assured that the (symbol-)estimation is not too large in step 8.

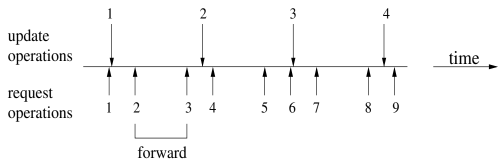


Fig. 5. Determination of an optimal forward interval

The system waits for the respective period of time in step 6, and reloads the remote data. High frequency reload is applied until a change with respect to the current and the previous version of the reloaded data is detected (steps 9 to 14). After the change detection (step 14), the algorithm continues with the prediction of the subsequent symbol in step 4.

One main aspect is how this algorithm may detect an error in the estimation and thereby adapt to changes in the update behavior of the data source. For this purpose the algorithm tests, if a currently predicted symbol corresponds to the currently observed update times. E.g. in the interval between reload 2 and 3 in figure 5 no change of the remote source should occur. This is tested in step 8 of the reload-control algorithm. A second aspect is if the maximal estimated interval length is correct. For this purpose the number of required reloads in the fast forward loop (steps 9 to 14) is counted in step 11. If it is loo large (step 12), the interval estimation is obviously false. If the predicted interval doesn't correspond to the observed update-interval the algorithm terminates in step 8 or step 12.

Reload-Control (Input: estimated grammar $\sim H$, sampsize)

```
1    find current state of the automaton
2    set $forward := 0$
3    do
4      find next symbol according to automaton $H$
5      compute optimal reload time $t$
6      wait until $t$
7      reload
8      confirm equality between old and new data
9      do
10       $t := t + sampsize$
11       $forward++$
12       if $forward > 3$: stop algorithm
13       wait until $t$, reload
14     until change detected
15     $forward := 0$
16   until end of observation
```

Fig. 6. Algorithm to determine optimal reload times based on the grammar estimation in section 3

4 Example

4.1 Grammar Estimation

In order to illustrate the *Grammar-Estimation* algorithm we assume that the system registers a sequence of symbols *ababcab*. After detecting the symbol a, the default hypothesis $H1 := (a)^\circ$ is inserted into the empty set *hset* (table 1) in step 4 of the *Grammar-Estimation* component. This hypothesis is state '1'. In step 2 in table 1 the second detected symbol is b which is different to the

Table 1. Computation steps of the *Grammar-Estimation*-algorithm for the sequence *ababc...*

step	input symbol	reject hypothesis	insert hypothesis	hypotheses/state
1	a		H1:=$(a)^\circ$	H1/state=1
2	b	H1	H2:=$(ab)^\circ$	H2/state=1
3	a		H3:=$(aba)^\circ$	H2/state=2 H3/state=1
4	b	H3	H4:=$(abab)^\circ$	H2/state=1 H4/state=1
5 ⋮	c	H2,H4	H5:=$(ababc)^\circ$	H5/state=1

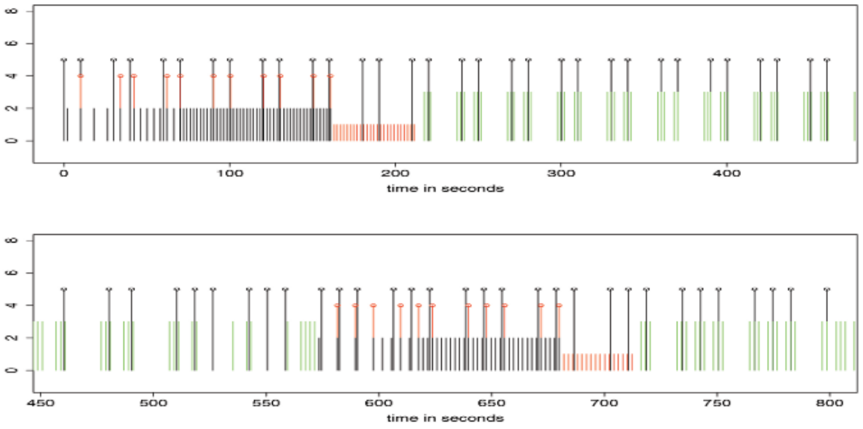


Fig. 7. Visualization of the adaptive reload optimization

prediction of $H1$ ($H1.prectict = a$). Therefore $H1$ is deleted from *hset*. Again, the default hypothesis $H2 := (ab)^\circ$ is added to *hset*. In step 3 the symbol a is detected. In this case the prediction of $H2$ is true. Therefore the state of $H2$ is increased. The default hypothesis $H3 := (aba)^\circ$ is added to hset. This procedure continues until in step 5 the symbol c is detected. In this step $H2$ and $H4$ have to be rejected and the default hypothesis $H5 := (ababc)^\circ$ is added. After consideration of subsequent symbols this hypothesis turns out to be consistent with the sequence *ababcab* and it also turns out to be the smallest one.

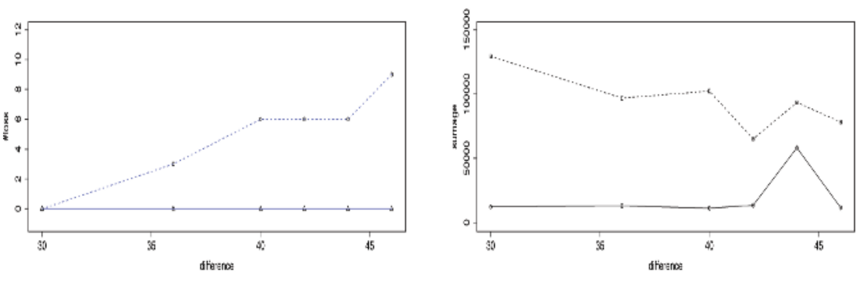
4.2 Application Example

Figure 7 shows an application of the complete algorithm. Impulses of length 5 in figure 7 denote times of remote updates. In this example at first the remote update pattern is $(ab)^\circ$ and changes to $(aab)^\circ$ at time ~500 sec (symbol a has a length of 10 seconds; symbol b has a length of 20 seconds). In the first phase of the algorithm (time 1..100 sec) the sampling interval is determined according to section 3.2. Impulses of length 2 denote reload requests. Impulses of length 4 denote detected changes. The detected changes in the interval from 70 to 160 seconds are used for the grammar estimation. Next, the state is determined (impulses of length 1) and the optimized reload is applied (impulses if length 3). At time 570 the algorithm detects an error in the estimation and starts again with the sampling interval estimation.

5 Experiments

In this section we demonstrate the advantages of the adaptive regular reload policy, in the following denoted as the *regular* method, to the policy based on a constant reload frequency, denoted as the *constant* method.[3] Optimization refers to a specific definition of costs, which have to be minimized. For the cost definition we consider the *age* of data *(sumage)* and the number of lost data objects $\sharp loss$ as defined in section 2. The different quality parameters may easily be computed from the sequence of reload operations, which is known and the sequence of remote updates, which is either known or has to be estimated by high-frequency sampling. In the experiments we select the number of downloads of the constant-frequency method such that it is equal to the number of downloads of the regular method. In these experiments the costs created by the estimation process are ignored. These costs only occur once at the beginning of the application. In this respect we consider the costs 'in the limit' and assume that changes in the update behavior that result in a re-estimation are sufficiently rare and therefore don't create significant costs. Figure 8 shows an experiment with generated data which demonstrates the basic properties of the new reload policy. In the experiment a grammar of two symbols $(ab)^\circ$ was considered with a total length of one cycle of 50 seconds (length(a)+length(b)=50 sec). The difference of the two symbol lengths ($|length(a) - length(b)|$) is increased in the experiment (x-axis in figure 8). If e.g. the difference is zero the interval lengths are identical (length(a)=25 and length(b)=25). Figure 8 a) shows that if the interval length difference of the two symbols gets larger, the number of lost data increases when the constant method is applied while it is zero for the regular method. The reason is that the constant method has to consider a reload interval smaller than the smallest update interval in order to capture all updates. Since in this method the reload frequence is constant, this may lead to a large number of (superfluous) reloads; otherwise, if the number of reloads is fix as in the described experiment, the $\sharp loss$ value is likely to increase. Figure 8 b) shows that the age of data (*sumage*) is significantly smaller if the regular method is applied.

[3] This method is currently applied by most Web crawlers, Web caches etc. as described in section 1.1.



(a) ♯lost for the regular method and the constant method (dashed line)

(b) *sumage* (in seconds) for the regular method and the constant method (dashed line)

Fig. 8. The quality parameters $\sharp loss$ and *sumage* for a cyclic grammar $(ab)^{\circ}$ for different interval differences $|length(a) - length(b)|$

Table 2. Comparison of reload policies for different Web pages. Page 1 (http://www.oanda.com) provides financial data. Page 2 (http://www.sec.noaa.gov/rt_plots/xray_5m.html) provides data related to space weather. The reduction of *agesum* applying the regular method ranges from 50% to 90%

	page	loss	sumage (seconds)
constant method	1	0	24143
regular method	1	0	12960
constant method	2	0	50331
regular method	2	0	6233

The previous result may also be shown for real Web pages. In the following experiment a time interval of 5 days is defined. In this time interval the two reload policies are executed and the original update times are determined by fast sampling in order to compute the quality measure. The number of downloads are equal for both systems, which requires that the update behavior has already been determined in an offline step. Table 2 shows the result of a comparison of the different reload policies for different Web pages. Page 1 has the URL http://www.oanda.com. It provides information related to currency exchange. Page 2 (http://www.sec.noaa.gov/rt_plots/xray_5m.html) provides data related to x-radiation from the sun. The results in table 2 show that values for lost data and superfluous data are very similar if as in this example the number of downloads is identical (the length of symbols is similar in this case in contrast to the simulation experiment above). The age of data (*sumage*) may be reduced from 50% up to about 80% for different pages.

6 Conclusion

The article presents an algorithm to estimate the parameters of the update behavior of a distant source that may be approximated piecewise by a specific kind of a regular grammar. The algorithm takes into account that the points in time where data sources in the Web change may usually not be registered exactly. The estimated grammar is used to determine optimal points in time for data reloads with respect to a minimization of the age of data and the amount of lost data. In the experiments it is shown that the age of data may be reduced significantly using the adaptive regular reload policy compared to a constant-frequency based method in the case that changes of the remote update behavior are rare and the respective costs for a re-estimation may be ignored.

The basic idea concerning this research is to develop a personal information system that makes the access to the dynamic properties of the Web for a user as simple as possible. In the article some of the parameters needed for the estimation are determined automatically like the sampling size. However several parameters still have to be provided by a user like the time needed for the sampling size determination, the period of time needed for the grammar estimation, the initial sampling size etc. If the presented results would be used for a personal information system it would still be inconvenient to actually use this tool especially if a large number of data objects has to be considered.

References

1. A.Arasu, J.Cho, H.Garcia-Molina, A.Paepcke, and S.Raghavan. Searching the web. *ACM Trans. Inter. Tech.*, 1(1):2–43, 2001.
2. Shivnath Babu and Jennifer Widom. Continuous queries over data streams. *SIGMOD Rec.*, 30(3):109–120, 2001.
3. Brian E. Brewington and George Cybenko. How dynamic is the Web? *Computer Networks (Amsterdam, Netherlands: 1999)*, 33(1–6):257–276, 2000.
4. Xin Chen and Xiaodong Zhang. Web document prefetching on the internet. In Zhong, Liu, and Yao, editors, *Web Intelligence*, chapter 16. Springer, 2003.
5. Junghoo Cho and Hector Garcia-Molina. Estimating frequency of change. *ACM Trans. Inter. Tech.*, 3(3):256–290, 2003.
6. E. Coffman, Z.Liu, and R.R.Weber. Optimal robot scheduling for web search engines. *Journal of Scheduling*, 1(1):15–29, June 1998.
7. World Wide Web Consortium. W3c httpd. http://www.w3.org/Protocols/.
8. Mukund Deshpande and George Karypis. Selective markov models for predicting web page accesses. *ACM Trans. Inter. Tech.*, 4(2):163–184, 2004.
9. A. Dingle and T.Partl. Web cache coherence. *Computer Networks and ISDN Systems*, 28(7-11):907–920, May 1996.
10. P. Dupont, L. Miclet, and E. Vidal. What is the search space of the regular inference? In R. C. Carrasco and J. Oncina, editors, *Proceedings of the Second International Colloquium on Grammatical Inference (ICGI-94): Grammatical Inference and Applications*, volume 862, pages 25–37, Berlin, 1994. Springer.
11. Brian S. Everitt. *Cluster Analysis*. Hodder Arnold, 2001.

12. E. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
13. Julie E. Kendall and Kenneth E. Kendall. Information delivery systems: an exploration of web pull and push technologies. *Commun. AIS*, 1(4es):1, 1999.
14. D. Kukulenz. Capturing web dynamics by regular approximation. In *WISE04, International Conference on Web Information Systems Engineering, Brisbane*, 2004.
15. D. Kukulenz. Optimization of continuous queries by regular inference. In *6th International Baltic Conference on Databases and IS*, volume 672 of *Scientific Papers University of Latvia*, pages 62–77, 2004.
16. C. Olston and J.Wildom. Best-effort cache synchronization with source cooperation. In *Proceedings od SIGMOD*, May 2002.
17. J. Oncina and P.Garcia. Inferring regular languages in polynomial update time. *Pattern Recognition and Image Analysis, Perez, Sanfeliu, Vidal (eds.), World Scientific*, pages 49–61, 1992.
18. Rajesh Parekh and Vasant Honavar. Learning dfa from simple examples. *Machine Learning*, 44(1/2):9–35, 2001.
19. D. Wessels. Intelligent caching for world-wide web objects. In *Proceedings of INET-95, Honolulu, Hawaii, USA*, 1995.
20. J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen. Optimal crawling strategies for web search engines. In *Proceedings of the eleventh international conference on World Wide Web*, pages 136–147. ACM Press, 2002.