Data Mining for Intelligent Web Caching

Francesco Bonchi Fosca Giannotti Giuseppe Manco Chiara Renso
CNUCE-CNR – Institute of Italian National Research Council
Via Alfieri 1, 56010 Ghezzano (PI) Italy
E-mail: {f.bonchi,f.giannotti, g.manco, c.renso}@cnuce.cnr.it

Mirco Nanni Dino Pedreschi Salvatore Ruggieri
Department of Computer Science, University of Pisa
Corso Italia 40, 56125 Pisa, Italy
E-mail: {nnanni,pedre,ruggieri}@di.unipi.it

Abstract

The paper presents a vertical application of data warehousing and data mining technology: intelligent web caching. We introduce several ways to construct intelligent web caching algorithms that employ predictive models of web requests; the general idea is to extend the LRU policy of web and proxy servers by making it sensible to web access models extracted from web log data using data mining techniques. Two approaches have been studied in particular, one based on association rules and another based on decision trees. The experimental results of the new algorithms show substantial improvement over existing LRU-based caching techniques, in terms of hit rate*, i.e., the fraction of web documents directly retrieved in the cache. We designed and developed a prototypical system, which supports data warehousing of web log data, extraction of data mining models and simulation of the web caching algorithms, around an architecture that integrates the various phases in the knowledge discovery process. The system supports a systematic evaluation and benchmarking of the proposed algorithms with respect to existing caching strategies.*

1 Introduction

If data mining is aimed at discovering regularities and patterns hidden in data, the emerging area of web mining is aimed at discovering regularities and patterns in the structure and content of web resources, as well as in the way web resources are accessed and used [3, 4, 5].

In this paper we describe one particular data/web mining application based on data warehouse technology: the development of an intelligent web caching architecture, capable of adapting its behavior on the basis of the access patterns of the clients/users. Such usage patterns, or models, are extracted from the historical access data recorded in log files, by means of data mining techniques.

More precisely, the idea is to extend the LRU (*least recently used*) cache replacement policy adopted by web and proxy servers by making it sensible to web access models, extracted from web log data. To this end, we introduce several ways to construct intelligent web caching algorithms that employ predictive models of web requests. The goal of these algorithms is to maximize the so-called *hit rate*, namely the percentage of requested web entities that are retrieved directly in cache, without requesting them back to the origin server.

If compared with the many alternatives and variations to LRU caching presented in the literature, and briefly discussed later, our approach has a unique feature: its adaptiveness to changes in the usage patterns, which are rather natural in the web. This is due to the fact that the proposed caching strategies are parametric w.r.t. the data mining models, which can be recomputed periodically in order to keep track of the recent past.

As a final step in the knowledge discovery process, we designed a reference web caching model as a means to evaluate the models extracted by data mining. The architecture, which emulates a cache, and is parametric to the replacement strategy, supports the evaluation and comparison of the various replacement policies, according to the hit rate.

The overall process, from log data acquisition to model extraction up to evaluation by the web cache architecture, is formalized and implemented within a database managements system, Microsoft's SQL Server 2000, using the DTS technology – Data Transformation Services.

2 Towards Intelligent Web caching

2.1 Web caching

It is recognized that deploying caching in the world wide web can improve the net traffic in several ways. In particular, it can reduce the *bandwidth consumption*, the *network latency* perceived by the client and the *server load*. Moreover, it can improve the *network reliability* perceived by the client, since in case of temporary unavailability of the network connection or of the server services, the local caches temporarily replace the server.

Web caching, however, poses several issues which risk to reduce its apply-ability and effectiveness, such as *consistency*, *dynamic objects* and several *security and legal issues*.

Evaluation measures and techniques. In order to evaluate the quality of a web caching system, several measures can be applied, depending on which resource we are focusing on – usually the bottleneck of the system under consideration. The most commonly used criteria are basically the following three: *Hit rate*, i.e. the ratio of requests fulfilled by the cache, and then not handled by the web servers, *Weighted hit rate*, i.e. the ratio of bytes served to the client by the cache, and *Latency*, i.e. the time that an end-user waits for retrieving a resource.

Following a common approach, in this work we choose to evaluate our novel caching strategies by simulating them over a set of collected logs from a server. Results of experimentations will be presented in terms of hit rates.

Traditional caching strategies. Several caching algorithms have been developed so far, characterized by the replacement strategy they implement, i.e. the criteria they follow in selecting the objects to evict from the cache when it is full. Among them, we present the following three, which will be referred in the rest of the paper: the *LRU* strategy, which orders objects by last access time, and removes first those with the older value; the *SLRU* strategy, which sorts objects by the product $\Delta T * Size$, ΔT being the number of requests received since the last access to the object, and removes first the entities with higher values; eventually, another strategy which we call *ORCL* (ORaCLe strategy), a modification of SLRU such that, at each time, the weight of an entity is $\Delta' T * Size$, where $\Delta' T$ is the number of requests that *will be* received until the next access to the object (of course, the ORCL strategy can be simulated only on historical data).

2.2 A reference model for intelligent caching

Traditional caching strategies can be modeled by considering entities in cache as belonging to a priority queue. In this way, the cache replacement strategy coincides with the priority queue weight assignment policy.

We call a policy that exploit knowledge extracted from past requests a `DataMiningWeightModel`. Of course, when the weight assignment policy is a fixed policy, then it boils down to traditional caching strategies, i.e. our definition is conservative. The generic model for intelligent caching strategies is reported in Figure 1.

```
        PriorityQueue Cache;
        DataMiningWeightModel DMM;
        CacheEntry t, t_fresh;
        long hits = 0;

1.      DMM.build();
2.      loop forever {
3.        do {
4.          get_request(t);
5.          if (Cache.contains_freshcopy(t)) {
6.            hits += 1 ;
7.            whits += t.bytes_sent_to_client ;
8.            Cache.update(t, DMM);
9.          } else {
10.           Cache.delete(t);
11.           Retrieve_freshcopy(t, t_fresh);
12.           Cache.push(t_fresh, DMM);
13.           while (Cache.size > Cache.maxsize)
14.             Cache.pop();
15.         }
16.       } while (condition);
17.       DMM.updatemodel();
18.     }
```

Figure 1. Intelligent caching reference model

First of all (line `1`), an initial data mining model `DMM` is built on past data. Then (line `2`) the following cycle is repeated forever.

For a requested entity `t`, if the cache contains the entity and it is fresh (line `5`), then the entity is returned to the client and this case is considered as a *hit* (line `6`). Also, the bytes sent back to the client are counted for the *weighted hit rate* measure.The weight of the entity in cache, and possibly the weights of other elements in cache, is updated (line `8`) according to the weight assignment policy expressed by `DMM`.

On the contrary, if the entity is not in cache or it is stale, (line `9`), we have a *miss*. The entity is deleted from the cache (line `10`) and a fresh version is retrieved and pushed into the cache (lines `11-12`). The `push` method consists of assigning a weight to the entity and, possibly, updating the weights of other elements in cache. If the inclusion makes the cache exceeding the maximum size, then entities are popped out from the cache accordingly to their weights (lines `13-14`).

Finally, the data mining model `DMM` may be periodically updated when some $condition$ becomes false (line `16`), e.g. at fixed time intervals or when the cache performance decreases. We model such an update by the method `updatemodel` (line `17`).

3 A data mart of web log data

The availability of a data warehouse may serve multiple purposes by providing a consistent and reliable repository of data over long periods of time. In addition, a data mart contains a subset of the data warehouse that is of value to a specific group of users, e.g. for data mining analysis. We have developed a data mart for web logs to support intelligent caching strategies. The data mart is populated starting from a web log data warehouse or, more simply, from raw web/proxy server log files that we assume containing some very basic fields. The data mart population consists of a number of preprocessing and coding steps that perform data selection, cleaning and transformation. Also, the data mart population computes additional derived fields for each transaction that are well-suited for input to data mining algorithms.

The data mart has been implemented as a relational database, using Microsoft SQL Server 2000 Beta 2 [7]. Interestingly, also the processes of populating the data mart is formalized and automated within the SQL Server 2000 framework.

The processes of data preparation and data mart population have been designed using SQL Server 2000 Data Transformation Services (DTS), a tool that allows to specify import / export / transformations processes of data through text files, databases or applications. Such processes concern several tasks, the main ones being: URL normalization, fields extraction, hash coding (for dealing with strings in a easier and more efficient way), approximating the size of entities on the origin server, computing the time distance among requests and consequently computing the user sessions (for this last task, the so called *reference length approach* [2] was followed).

4 Deploying the reference model with data mart data

In this section two instantiations of the general intelligent caching model are presented, the first one based on association rules, and the second one based on decision trees. For each of them, a brief introduction to the general mining task is given, followed by the description of its application to the caching strategy, a summary of the results of experimental simulations, and eventually an overview of the data mining modeling process.

4.1 Association rules

In general, given a database of transactions, each composed of a set of items, we define an *association rule* [1] as an expression of the form $A \Rightarrow B$ $[S, C]$, where A and B are sets of items; S is the *support* of the rules, defined as the rate of transactions containing all items in A and all items in B; C is the *confidence* of the rule, defined as the ratio of S with the rate of transactions containing A. In probabilistic terms, S approximates the probability that all items in the rule appear together in a transaction, while C approximates the (conditional) probability that items in B appear given that items in A appear. Usually, minimum thresholds are specified for for support and confidence, in order to consider only significant rules.

In the application domain under consideration, items correspond to web resources, while transactions correspond to *user sessions*. Thereby, a rule such as $\texttt{res}_1 \Rightarrow \texttt{res}_2$, informally means that if $\texttt{res}_1$ appears in a user session, $\texttt{res}_2$ too is expected to appear in the same session, though possibly in reverse order and not consecutively.

In our approach we keep the LRU criteria for assigning priorities, while the overall strategy is *extended* by modifying the priorities of the entities already in cache as *reaction* to the new incoming requests.

First of all, a set of association rules is extracted from a training dataset, obtained from past log data. Thereby, if at some time an URL A and $A \Rightarrow B$ is in the set of rules previously extracted, then we *predict* that the URL B too will be requested soon. This suggests that if B is already in cache, its eviction should be delayed: we accomplish that by increasing assigning to B the priority it would have if it was requested immediately after A, so that it moves to the top of the queue.

The workload (named `NatPort`) is split into a two days training set and a two days validation set: the first two days of log data have been used for extracting association rules over user sessions, with minimum support and minimum confidence thresholds respectively equal to 0.3125% and 20%. The resulting extended LRU strategy has been simulated over the last two days log data.

Figure 2(a) shows some of the results obtained by simulating an LRU cache and an *association-based extended LRU*. Performances are plotted for different sizes of the simulated cache, from 0.2 Mbytes to 102.4 Mbytes (corresponding respectively to the 0.0051% and 2.6% of the web server size), with exponential growth.

The two curves have a similar shape, and, as expected, converge as the cache size grows. The graph reveals a significant hits improvement of our approach over the standard LRU for all cache sizes, ranging from an absolute gain[1] of $+7.75\%$ (for the smallest cache) to $+1.20\%$ (for the largest cache).

[1] Here and in the rest of the paper, *absolute gain* and *absolute improvement* stand for the difference between two hits rates, so that the absolute gain of a 15% hits rate over a 5% one is a $+10\%$.

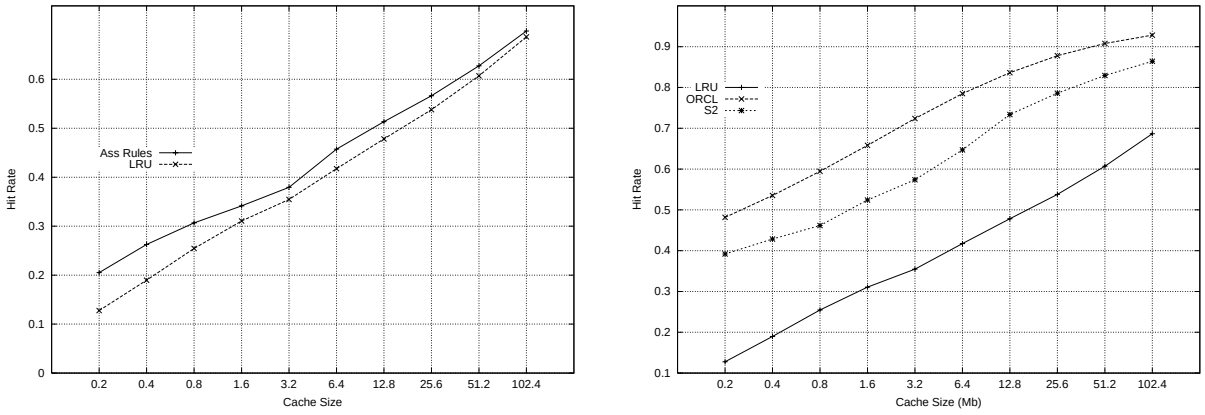

Figure 2. `NatPort`**: Hit rates w.r.t. Cache Size for the (a) association- and the (b) decision tree-based strategies**

4.2 Decision trees

Decision trees induction concerns the construction of a model – a *classifier* – that describes a discrete attribute, called the *class*, of an entity in terms of other attributes of an object, called the *observed attributes* or features. The model is constructed for a set of objects (called *training set*) whose class values are known, and can be used to predict the unknown class value of objects in another set.

The central idea of this section is to approximate the ORCL strategy, presented in Section 2.1. Since such strategy needs to known the *next access distance* – i.e., the distance between each request and the next one for the same URL – a classifier is trained that for every URL requested is able to *predict* such distance. Such value is therefore chosen as the class variable, and since it is a continuous attribute, we will actually use a discretization of its values into a set of a few discrete values. The observed attributes that can be used to build a classifier are those available by a cache at transaction time. Such a tree is a *data mining model* that can be used in an intelligent caching system in order to assign a weight to an entity.

Among the most popular classification algorithms (see [6] for a survey), we use decision trees as the classification model for our analysis. In particular, in our experiments we use EC4.5 [9], an efficient implementation of the well-known C4.5 [8] decision tree algorithm.

The general strategy described in Section 2.2 is to be instantiated by making choices about observed attributes, discretization of the *next access distance* and weight assignments.

A first choice is concerned with the selection of a set of observed attributes, i.e. of the attributes used to construct the decision tree, among those ones available at the time of URL request. The strategy described in this paper, called *S2*, restricts to consider the following basic fields: the size of the requested entities, the directory depth of the requested URL, and the hour part of the date as continuous attributes; the file extension and the main directory of the requested URL as discrete attributes. Moreover, a discretization for the *next access distance* has to be chosen, deciding the number of classes and how to discretize values into intervals. We experimented discretization into 4 classes.

Following the intuition of approximating the ORCL strategy, we adopt a weight assignment function that *corrects* the LRU weight by adding a displacement that is related to the priority of the request in an ORCL strategy, i.e. a displacement of the form $\alpha(next\_access\_time) * \beta(size)$, for some inversely proportional functions $\alpha()$ and $\beta()$.

As for the association based LRU extension described in the previuos section, the decision tree classifier is built over a two days training set extracted from the `NatPort` workload, and the performance of the S2 strategy is computed by simulation over the two days validation set. The performance of strategies S2, LRU and ORCL is compared in terms of hit rate in Figure 2(b). The hit rate exhibits an impressive improvement of S2 w.r.t. LRU: consistently around 25% absolute improvement. The simulation of ORCL achieved by S2 is rather impressive: the gain between LRU and S2 is consistently 50% through 75% of the gain between LRU and off-line, theoretical strategy ORCL. The achieved hit rate outperforms also other traditional strategies, not reported here for the lack of space.

5 Conclusions

We have presented two approaches to enhance LRU-based web caching with data mining models built on historical data, mainly aimed at increasing the hit rate. Also, the design of a suitable data mart has been presented, together the main problems that such design must solve. The approaches differ for the kind of data mining model adopted: association rules and decision trees.

The performance figures of the developed methods, compared with LRU from one side and the theoretical off-line strategy ORCL from the other side, indicate substantial increase in the hit rate: the decision-tree strategy S2 outperforms all best fixed strategies (LFU, LRUMIN, SLRU); moreover, in principle, S2 can be combined with the orthogonal association-rule strategy, in such a way that further enhancements may be achieved.

Although further extensive benchmarking is required, there is a strong indication that data mining-based caching yields systematically enhanced hit and weighted hit rates, due its adaptiveness to recent history; we believe that adaptiveness is indeed the reason that makes our approach preferable to any fixed caching strategy.

Directions for further research include: the completion of the *benchmarking phase* with log data from various web and proxy servers, as well as synthesized data with various artificial distributions; the *combination* of the association-rule and the decision-tree approach into one; to investigate other possible *metrics to optimize*, and other possible strategies to start with; considering approaches based on clustering.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 487–499, Santiago, Chile, 1994.

[2] R. Cooley, B. Mobasher, and J. Srivastava. Grouping web page references into transactions for mining world wide web browsing patterns. In *Proc. of the 1997 IEEE Knowledge and Data Engineering Exchange Workshop (KDEX-97)*, November 1997.

[3] O. Etzioni. The world-wide web: quagmire or gold mine? *Communications of the ACM*, 39:65–68, 1996.

[4] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Mateo, CA, 2000.

[5] R. Kosala and H. Blockeel. Web mining research: A survey. *ACM SIGKDD Explorations*, 2(1):1–15, 2000.

[6] T.S. Lim, W.Y. Loh, and Y.S. Shih. A comparison of prediction accuracy, complexity, and training time of thirthy-tree old and new classification algorithms. *Machine Learning Journal*, 1999. To appear, http://www.Recursive-Partitioning.com/datasets.html.

[7] MicroSoft Corporation. MicroSoft SQL Server 2000. http://www.microsoft.com/sql.

[8] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[9] S. Ruggieri. Efficient C4.5. *IEEE Trans. on Knowledge and Data Engineering*, 2001. To appear, http://www-kdd.di.unipi.it.