Internet object caching based on semantics and access history

Alcides Calsavara , Marcelo Roberto Schuck

[1]Programa de Pós-Graduação em Informática Aplicada
Pontifícia Universidade Católica do Paraná
Rua Imaculada Conceição, 1155, Prado Velho
80215-901 Curitiba, PR

{alcides,schuck}@ppgia.pucpr.br

Abstract. *This paper presents a novel Internet object cache replacement strategy named LSR/H where semantics of objects and subject access history are employed as heuristics to evict objects from cache. It is shown that LSR/H performs better than well-established replacement strategies when multiple threads of interest exist and the multiple semantics stability property is verified. Our work shows that, depending on cache size, LSR/H can perform around 15% better than well-established replacement strategies, thus permiting an important gain in network performance. It can be a starting point to redesign the caching scheme currently in use in order to achieve better network performance. Also, our research work can be exploited in specific domains of applications, such as digital libraries and e-commerce, where taxonomy for objets are often well-established.*

1. Introduction

The Internet is strongly dependent on a good caching scheme in order to achieve reduced latency, scalability and quality of services. Basicaly, a caching scheme allows Internet objects – Web pages, text documents, binary files, etc – to be accessed faster by clients and reduces network and server load, by keeping copies of accessed objects in special storages, i.e., caches.

One of the most relevant matters in a caching scheme is that caches are limited in size. So, should an object enter the cache when there is not enough room for it, a replacement strategy must be employed to decide which objects should be evicted from the cache. Naturally, a good replacement strategy should evict objects which have less chance to be requested again, at least in the near future, in order to increase cache's performance and, consequently, the network performance. For that reason, replacement strategy was a subject of intense research for years and, as a result, nowadays a few well-established strategies – discussed in Section 2. – are widely employed on the Internet. Invariably, these strategies employ some heuristics based on operational information, such as object size, last access date, frequency of access, and so on. In fact, the research carried out in this area was very successfull and, as discussed in [Wolman et al., 1999], many authors stated that effort does not need to be spent anymore in the design and implementation of highly cooperative and scalable cache systems.

More recently, though, the Internet started to be rethought, mainly due to the Semantic Web effort[1], in order to add mechanisms and services that would exploit the semantics that is intrinsic to every Internet object. One such service, for example, is the DMOZ Open Directory Project[2], where a large taxonomy for all the Internet objects is under development, in a collaborative work of thousands of people around the world.

Tuned with this process of rethinking the Internet, a few years ago, we started a research project which aims at creating a replacement strategy that exploits the semantics of objects. The first version of the newly created strategy was called *Least Semantically Related*

[1]URL: http://www.w3.org
[2]URL: http://www.dmoz.org

[dos Santos, 2001, Calsavara and dos Santos, 2002, Calsavara et al., 2003], LSR for short, which is described in Section 3.. The rationale beyond LSR is to evict objects which are less semantically related to the current object to be entered into cache. Its performance was measured and compared with well-established replacement strategies through simulation, by employing a small base at that time. The first results have shown that LSR performs better than well-established replacement strategies under the following condition: there must exist sequences of object requests where the objects are semantically related; LSR performs well when such sequences are long and the semantics of the objects are close. Particularly, such condition is verified when the cache serves users that access objects related to a single subject at a time. In other workds, LSR performs well when there is a *single thread of interest* at a time. Hence, our initial experiments showed that LSR suits well in a particular case. That encouraged us to proceed with our research in the pursuit of a more general solution.

The idea of exploiting semantic information, i.e., related to the content of the documents, instead of operational information, is not exclusive to our work. The research work described in [Brunie et al., 2002] was carried in parallel to our work and without knowing about each other. The way they designed and implemented an algorithm to exploit semantic information, however, is distinct from the way we do. They developed a strategy named *Temperature*, where subjects and documents become "hot" as they are more accessed, or "cool" otherwise. This permits to identify objects which are of more interest and then should be kept in cache. The authors claim a considerable gain in hit rate compared to the Least Recently Used (LRU) strategy. However, the actual gain is difficult to measure because they used a small base of objects in their experiments. Nevertheless, their work, like the first phase of our work with LSR, shows that there is a potential area of research by exploiting semantic information to design new cache replacement strategies.

In this paper, we present the succeeding version of LSR, the *Least Semantically Related with Access History*, LSR/H for short, which is described in Section 4.. It improves its first version in the sense that it relaxes the necessary condition to perform well: there can be *multiple threads of interest* at a time. The main artifact used for this purpose is a history of subject accesses; such a history permits to determine how distinct threads of interest evolute in time. The idea of employing some form of access history is influenced by a related work – discussed in Section 2. – where history is applied to improve the performance of several well-established replacement strategies. Our experiments – described in Section 5. – have shown that, under this new relaxed condition, while, as expected, the original LSR has a very modest performance, LSR/H outperforms all typical well-established replacement strategies. Moreover, the experiments were based on a considerably large base of objects and long sequences of simulated object access, such that the stability of the simulation results obtained was properly verified. Since we got very positive results, we concluded that LSR/H was a worth step and still can be improved, as discussed in Section 6..

2. Well-established replacement strategies

Cache replacement strategies have a fundamental role in the project of any storage component. They have been intensively studied in the context of virtual memory management systems [O'Neil et al., 1993]. There are dozens of such strategies documented in the literature. Currently, a number of cache replacement strategies is in use on the Internet [Arlitt et al., 1998]. Typically, these strategies are based on operational information about objects, such as their size, their access frequency rate, their last time of access, and so on. In other words, the well-established cache strategies treat objects as black boxes and discard them according to some information that is not related to their contents at all. Our intention in this Section is just emphasize that they all use operational information about objects as criteria to evict objects. We neither analyze nor compare such strategies; their own bibliographical refecences present comparative results already. We briefly describe the most relevant cache replacement strategies because either

they are widely employed on the Internet or their research stage is quite mature. Good surveys on web caching schemes for the Internet can be found in [Wang, 1999], [Pinheiro, 2001] and [Podlipnig and Boszormenyi, 2003].

SIZE This strategy simply elects for removal the biggest object in cache. When two objects with the same size are elected, the less frequently accessed is removed [Aggarwal et al., 1999].

LEAST RECENTLY USED (LRU) This strategy elects for removal the object which is the one least recently used by clients [Aggarwal et al., 1999].

LEAST FREQUENTLY USED (LFU) This strategy elects for removal the object which is the one least frequently used by clients [Williams et al., 1996]. There are two versions of this strategy:

1. *In-Cache LFU*: In this version, the access counter for an object is zeroed everytime the object enters the cache.
2. *Perfect LFU*: In this version, the access counter for an object is zeroed only the first time the object enters the cache. If an object that has been previously removed comes back to cache, its access counter will have the same value as when it was last removed.

LOWEST RELATIVE VALUE (LRV) This strategy assigns a relative cost value for each object in cache in order to calculate their utility; the object with the lowest relative value is elected for removal [Rizzo and Vicisano, 2000].

LRUMIN This strategy tends to keep in cache objects of smaller size in order to minimize the number of replacements. Suppose that a new object of size *S* has to enter the cache and there is not enough room for it. If there is any objects in cache which size is at least *S*, the least recently used (LRU) one is removed. Otherwise, objects with size at least $S/2$ are sequentially removed following the LRU strategy. If still necessary, objects with size at least $S/4$, $S/8$, and so on are removed until the needed room is created [Aggarwal et al., 1999].

GREEDYDUAL-SIZE (GD-SIZE) This strategy is a generalization of the LRU. It is concerned with the case where objects have the same size, but incur different costs to fetch from a secondary storage. The strategy associates a value, H, with each cached object. The initial value of H is the cost to fetch the object. When a replacement has to be made, the object with the lowest H, say h, is replaced, and then all objects reduce their H values by h. Everytime an object is accessed, its H value is restored to the corresponding original value [Cao and Irani, 1997].

LOWEST-LATENCY-FIRST This strategy tries to minimize average latency by removing the object with the lowest download latency first [Wooster and Abrams, 1997].

HYBRID This strategy aims at reducing the total latency [Cao and Irani, 1997]. It replaces the object which results the lowest value for the following function:

$$\frac{(c_s + \frac{W_b}{b_s}) \times (n_p)^{W_n}}{z_p}$$

where s is a server, p is an object located in s, c_s is the time to connect to s, b_s the bandwidth to server s, n_p is the number of times p has been requested since it was brought into the cache, z_p is the size (in bytes) of p, and W_b and W_n are constants.

FIRST-IN, FIRST-OUT (FIFO) This strategy replaces the object that enters the cache first [Silberschatz and Galvin, 1994].

FUNCTION-BASED REPLACEMENT This strategy employs a general function for different factors, such as the last access, entry time of an object into cache, transfer cost and the time-to-live of an object [Wooster and Abrams, 1997].

SIZE ADJUSTMENT LRU (SLRU) This strategy is known as the *Knapsack Problem* – it orders the objects in cache according to their cost and size; the object with largest index is evicted from cache when a replacement is needed [Aggarwal et al., 1999].

Pyramidal Selection Scheme with Award This strategy classifies objects according to their size using a logarithmic function, combined with frequency access rate. [Cheng and Kambayashi, 2000].

Partitioned Cache This strategy splits the cache into partitions that store classes of documents based on their size, instead of having a single cache to store all documents. Its aim is to take into consideration the high variability noticed in the WWW [Murta et al., 1998].

As discussed above, the known strategies for cache replacement are based only on operational information about objects, and each strategy uses a particular heuristic to elect objects to remove from cache. Therefore, there is a condition where each strategy performs optimally, as well as there is a condition where it performs badly (worst case), that is, there is no optimal solution for the problem and the choice of the most appropriate strategy will depend always on the user access pattern.

Several of these strategies, including LRU e LFU, were extended to consider also an object access history [Vakali, 2001]. For each cached object, there is a record saying the last time it was requested. This approach improved significantly the performance of all strategies, showing the importance of keeping an access history.

There are, as well, some attempts to create semantic caching strategies for database and query systems, where data is well-structured, thus providing a means of organizing objects according to some semantic information. Some of those attempts include [Dar et al., 1996], [Chidlovskii et al., 1999], [Keller and Basu, 1996] and [Ren et al., 2003]. They all conclude that the semantic-based approach to cache replacement performs better than traditional strategies.

More recently, in the field of wireless infrastructure and applications, there is a preocupation to develop more appropriated caching schemes where physical locality can be exploited. The research work presented in [Zheng et al., 2002] combines such an idea with semantic information and obtains good results as well.

Finally, there are efforts to create caching mechanisms for distributed objects systems, such as [Atzmon et al., 2002], where an object can encapsulate a Web object; it builds a hierarchy for each cached object based on client's access pattern and objects are explicty evited by clients by invoking operations to re-register objects.

3. LSR strategy

The Least Semantically Related (LSR) replacement strategy [dos Santos, 2001] assumes that every user tends, for a certain period of time, to seek objects which are related to a given subject and, therefore, have close semantics, before changing to another subject. This user access pattern is what we call *semantics stability*. Moreover, LSR assumes that, at any time, all the users served by a cache have a common subject of interest, that is, LSR supports *single thread of interest*.

LSR discards objects which are less related to a new entry with respect to their semantics because they might be of less interest to users, thus favoring the permanence of objects in cache which are close with respect to their semantics and discarding objects which might be of less interest to users. LSR associates a semantics to each object in cache in such a way that it is possible to determine the semantics distance between any two objects; the semantics of an object is (either staticaly or dynamicaly) defined according to its contents. When a new object n enters the cache, LSR evicts the objects from cache which are the lest semantically related to n, that is, the objects which are furthest away from n with respect to their semantics.

One approach to associate semantics to each object is through a taxonomy, i.e., objects can be organized according to a tree-based hierarchy of subjects: each tree node represents a subject and contains objects whose contents have the corresponding semantics, as

well as a tree node may have children, which represent more specific subjects. An example of hierachy of subjects can be seen in Figure 1. The example is, in fact, a partial view of a real-world taxonomy defined by the DMOZ Open Directory Project. Thus, there is a function that, when applied to an object, returns the semantics of such an object in the form of a sequence of tree nodes that represent subjects, from more general to more specific ones. In the example, when such function is applied to the object identified by the URL *http://www.acm.org/crossroads/collumns/connector/july2000.html* (titled *Foundations of the Internet Protocol*) returns the semantics *Top/Computers/Internet/Protocols/IP*. Moreover, within this framework, a semantics distance between any two objects is given by the shortest path between them. So, it is straighforward to determine which objects are least semantically related to any object. Naturally, this framework is hard to achieve for the Internet as a whole in the present days, but there are efforts to create a standard taxonomy for Internet objects, such as the DMOZ Open Directory Project. Also, there is a proposal [Shmidt, 2002, Calsavara and Schmidt, 2004] based on the RDF (Resource Description Format) standard to create Web servers that provide the semantics of an object given its URL. Finally, it is perfectly feasible to create a taxonomy for more controlled environments, such as for a digital library and e-commerce, for a specific domain of knowledge or for a certain organization, where LSR could then be employed.

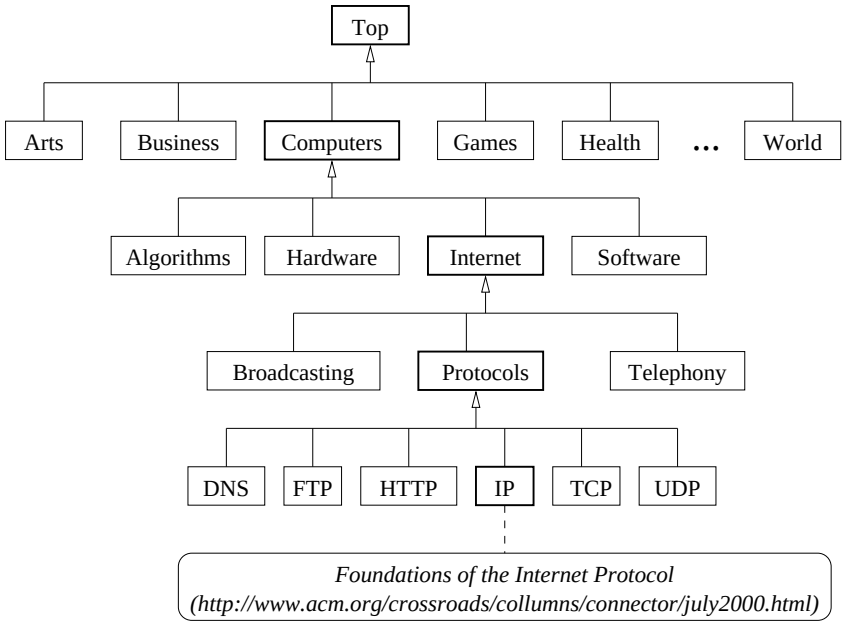


Figure 1. Partial hierarchy of subjects defined by the DMOZ Open Directory

LSR was properly formalized and implemented for the purposes of validation and performance analysis. We designed and implemented a simulation model that uses real-world data; since LSR assumes that it is possible to get the semantics for every Internet object, our first implementation is based on a tree of subjects to categorize objects which is a subset of the taxonomy given by the Yahoo! search engine[3]. Moreover, the simulated object accesses conform to a real distribution of objects per number of accesses, obtained from a Squid Proxy[4] log file; the Monte Carlo Method [Liu, 2001] is employed to implement such a distribution by categorizing Yahoo! objects according to the number of times they should be accessed in a certain period of time.

A standard metrics for the efficacy of a cache system is the so-called *hit rate*, which corresponds to the probability of finding in cache a certain object requested by an Internet client. The hit rate of a cache system depends on the cache size and on the present number of accesses. For

[3]URL: http://www.yahoo.com
[4]URL: http://www.squid-cache.org

that reason, a number of sequences of object accesses was produced and submitted for simulation in caches of different sizes. The experiment was carried out for thirteen sequences of accesses (varying in size from 1,000 to 30,000 accesses), six different sizes for cache (from 1 million to 10 million bytes) and four policies (LSR, SIZE, LRU and LFU), giving a total of 312 simulation scenarios. Each one of the thirtheen sequences was rearranged in order to reflect the assumed semantics stability.

The graphics in Figure 2 shows the effect of number of object accesses on the hit rate for a cache of a fixed size: it grows, tending to stabilize, as the number of object accesses grows. The graphics in Figure 3 shows the effect of the cache size on the hit rate: it grows as the cache size grows and, after a certain cache size, the hit rate is identical for all the strategies, as expected – in theory, the cache can be big enough to store all the requested objects. The most important fact observed is that, practically in all simulation scenarios, LSR perfomed – in terms of hit rate – better than well-established strategies, which, certainly, implies a gain in terms of network load.

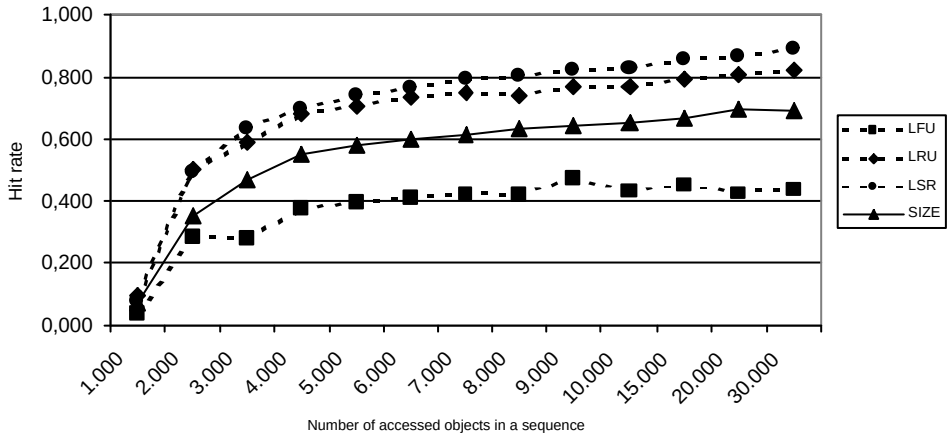


Figure 2. Hit rate for a cache of size 2MB

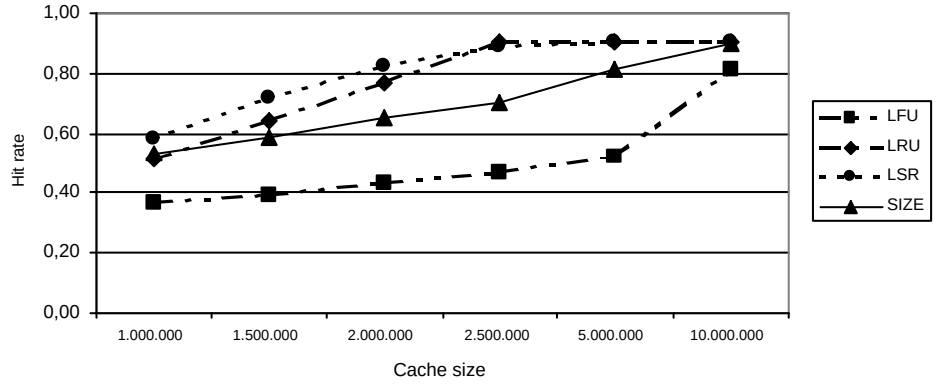


Figure 3. Hit rate for different cache sizes

subject semantics	weight
Top.Computers.Algorithms	10
Top.Computers.Software	9
Top.Business	8
Top.Games	7
Top.Computers.Algorithms	6
Top.Computers.Internet.Protocols.IP	5
Top.Computers.Internet.Protocols.HTTP	4
Top.Games	3
Top.Business	2
Top.Computers.Internet.Protocols	1

Table 1. A history of 10 subject accesses

4. LSR/H strategy

The Least Semantically Related with Access History (LSR/H) replacement strategy extends LSR in order to support multiple threads of interest: at any time, there can be a number of subjects of great interest to users. Naturally, the subjects of great interest may change over time, but it is assumed that such change happens smoothly. This seems to be the access pattern in typical groups of users. We refer to such property as *multiple semantics stability* and, for simplicity, we call the current subjects of great interest as *hot subjects*. Thus, LSR/H makes the best effort to keep in cache objects related to hot subjects. As a consequence, when necessary, it evicts objects whose semantics is far from the hot subjects.

A subject becomes "hot" when, for some period of time, there is a considerable number of accesses to objects whose semantics is close to that subject. Since it is not important to know the objects actually accessed, a simple means to detect hot subjects is to keep a *history of subject access*: each access weights for its corresponding subject, and recent accesses might weight more than older accesses. In the end, each subject should have a weight equal to the sum of all weights corresponding to that subject's accesses.

The current version of LSR/H implements semantics through a previously established taxonomy of subjects, like LSR does. Thus, subjects are organized in a hierarchy where each node represents a subject: nodes placed at levels close to the hierachy's root represent more generic subjects. Moreover, we may have objects placed on nodes at any level of the hierarchy. In this case, the identification of hot subjects is a little more complex because each access history record may refer to a node placed at any level of the hierarchy (that is defined by the semantics associated to the corresponding accessed object). So, for each record in the access history, its corresponding weight is added to the weight of the corresponding node (subject) and, recursively, to the weight of its parent nodes.

As an example, let us take the hierachy of subjects shown in Figure 1. Now, let us suppose LSR/H is configured to keep a history of subject access of length 10. In a certain moment, where we need to evict objects from cache, we may have the history shown in Table 1. The weights vary from 10 to 1, from the most recently accessed subject to the latest accessed one, respectively. We should note that this linear function that assigns weights to subject accesses is just an example; other functions, like a constant function, an exponencial function or a logarithmic function could be applied instead. The most appropriate function to employ is a subject of future research. According to the history, we can notice that there are at least three current threads of interest: (i) *Computers*, (ii) *Games* and (iii) *Business*.

LSR/H will evict objects from cache which belong to subjects of less weight, that

is, subjects of less interest. Firstly, LSR/H computes the weight of each subject, according to the given history of accesses. For example, the last access record says that the subject *Top.Computers.Algorithms* weights 10. Thus, 10 is added to all nodes in the corresponding path, except the *Top* node, i.e., 10 is added to the weight of nodes *Algorithms* and to the weight of its parent node *Computers*. By the end of the weight computation, our hierachy of subjects will have the weights as shown in Figure 4. Clearly, at the top level, *Computers* is the hottest subject. The subjects *Business* and *Games* are also hot. Within the *Computers* subject, the hottest suject is *Algorithms*, while *Internet* comes in second place. Therefore, LSR/H will evict first objects from subjects *Arts*, *Health* and *World*. Next, it will evict objects from subjects *Business* and *Games*. Finally, if still necessary, it will evict objects from subject *Computers*. Now, LSR/H applies the same criteria, recursively: it will evict objects from subject *Hardware*, next from subject *Internet* and then from subject *Algorithms*. Therefore, weights assigned to subjects according to a history of access is a proper way to identify current threads of interest in a group of users.

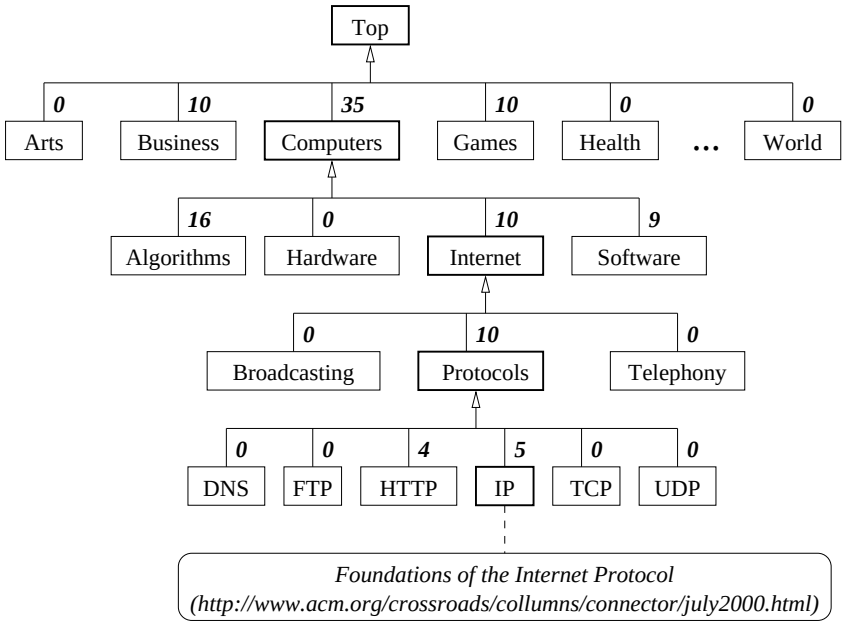


Figure 4. Hierarchy of subjects of Figure 1 with weights

5. Implementation and Experiments

LSR/H was implemented for the purposes of validation and performance comparison with well-established replacement strategies, as well as with its original version – LSR. All the strategies were implemented from scratch – in the Java programming language – to permit full control over the simulation process. The performance in hit rate and in byte hit rate of each strategy was measured by simulating several sequences of object requests against a base of Internet objects.

The use of real-world traces in our experiments would require automatic classification of objects in order to obtain their semantics (traces don't say what the semantics of accessed objects are; they only say their identification). It is our intention to do it, as explained in Section 6., similarly to what has been done in [Brunie et al., 2002]. For a while, we generate traces containing object identification and semantics in order to validate our strategy, by assuming that real-world traces might present the expected access pattern (threads of interest).

We built a base of Internet objects composed of a sample from the Internet objects indexed by the DMOZ directory. We selected 161,440 out of nearly 4 million Internet objects indexed by DMOZ by selecting a subtree of the whole hierarchy of subjects, which is composed of 590,000 nodes – the taxonomy defined by DMOZ. Then we implemented a program that reads metadata –

in XML format – provided by DMOZ, in order to rebuild within our simulation context the exact subtree defined by DMOZ, including all the corresponding object references.

The next step was to create sequences of object requests, which should fullfil two requirements. Firstly, they should contain multiple threads of interest and should comply with the multiple semantics stability property. Thus, we chose two top subjects, namely *Business* and *Reference*, to be the hot subjects. In real-world traces, we might have many more than two hot subjects at any time. So, a natural step ahead in our research work is to assess the correlation between the number (and intensity) of hot subjects and the performance of our replacement strategy. Secondly, they should comply with real-world access pattern in terms of the number of times objects are accessed within a given period of time. Thus, we collected data from a log file produced by a Squid Proxy Server containing 9,657,359 access records, in a institution composed of about 500 users. We discovered that 1,066,303 objects were accessed just once, 228,771 objects twice, 95,162 objects three times, and so on, reaching a total of 1,615,363 objects distributed over 1,591 access categories (category n contains all objects accessed n times). Then, we applied the Monte Carlo Method to reflect the verified distribution on the sequences of objects requests that we should create.

The relative performance of LSR/H can be verified through the graphics in Figure 5, which shows the cache performance after 0.5 million, 1.0 million and 1.5 million object requests. Each graphics shows the performance of the well-established replacement strategies LRU, LFU and SIZE, the performance of LSR (version designed to support a single thread of interest) and the performance of three variants of LSR/H where the length of the access history changes: LSR/H(n) denotes the variant where the access history length is n. A linear function was employed to define the weight of each subject access; in the case of a history length equal to 100, the last access weights 100, the previous one weights 99, and so on. The graphics show the performance in hit rate and in byte hit rate of each strategy depending on the cache's size. By observing the graphics, we can draw the following conclusions:

1. The simulation is stable since the results obtained with 0.5 million, 1.0 million and 1.5 million object requests are practically the same.
2. All strategies perform better as the cache size grows. Naturally, when the cache size tends to infinite, all requested objects would be cached and, consequently, there would be no replacement at all. Hence, from a certain point in time, all strategies would have exactly the same performance and would stabilize on it.
3. As expected, LSR has a very modest performance in the presence of multiple threads of interests.
4. From a certain cache size on (32MB in our experiment), LSR/H outperforms all other strategies (LSR/H performs 15% better than the second placed stragegy for a cache size of 128MB).
5. For any given cache size, LSR/H performs better as its access history grows, but such performance growth is not linear meaning that there is a threshold where it is not worthwhile to increase access history length.

Therefore, it has been shown that LSR/H brings a relevant performance gain over well-established replacement strategies when the multiple semantics stability property is verified.

6. Conclusions and future work

This paper introduced a novel Internet object cache replacement strategy named LSR/H (Least Semantically Related with Access History), where semantics of objects and subject access history are employed as heuristics to evict objects from cache. It has been shown, through simulation, that LSR/H performs better than well-established replacement strategies when multiple threads of interests exist and the multiple semantics stability property is verified.

The positive results obtained in our initial experiments with LSR/H motivates to proceed its verification and development. Some improvements to the current work include:

1. Analyse its performance when a larger number of threads of interests co-exist. (We experimented with only two threads.)
2. Usage of non-linear functions to define the weight of each subject access recorded in history. We would like to discover, for example, how an exponential function would affect cache performance.
3. Calculate how much LSR/H algorithm complexity depends on acess history length and estimate its impact on real-world cache servers (such as the Proxy Squid), for distinct platforms.
4. Find a method (either by analysis or simulation) to calculate the ideal access history length and cache size, according to a given object access pattern and the level of multiple semantics stability.
5. Implement a method for automatic classification of objects in order to make it possible to try LSR/H on the Internet as it is nowadays, where objects don't carry explicit semantic information. The experimens done by [Brunie et al., 2002] already implemented a method for that purpose, where concepts are extracted with a simple heuristic mixing the meta-tag keyword of the document (if present), the number of times a word is referenced in the document, the title of the document and the reference links to other web pages.
6. Try LSR/H in the real world, i.e., implement it for a typical Internet server such as the Proxy Squid in order to validade the performance results obtained through simulation.
7. Check whether it is possible to partition the cache according to object class (hipertext, binary, image, etc) and object size. Previous works, as discussed in [Pinheiro, 2001] and [Murta et al., 1998], have shown that there is a significant gain in hit rate when cache is partitioned. However, at present, we don't know how to combine such partitioning with our hierarchy of subjects.

Also, semantic information about Internet objects can be useful beyond optimization of the object replacement strategy implemented by a proxy. As suggested by [Brunie et al., 2002], people browsing objects related to the same subject define a *virtual community*, so we can investigate the use o LSR/H in the following ways in order to benefit such a community:

1. Implementation of prefetching heuristics, i.e., to pre-load objects before they are actually requested by users.
2. Optimization of collaboration procedures between proxies. When a subject becomes hot in some proxy, neighbour proxies can prefetch related objects from the first proxy. Similarly, proxies can share information about current threads of interest to optimize their cache management strategies.
3. Monitoring of the evolution of the interest of a virtual community.
4. Broadcasting of currently hot subjects, so users can be notified about what other people are looking at.

Therefore, the research carried so far contributed to show and measure the benefits of employing semantics combined with access history to replace objects in Internet caches. So, if initiatives to exploit semantics to provide new services on the Internet, such as the Semantic Web, really succeed, we already have a starting point to redesign the caching scheme in current use. On the other hand, the concepts developed by our research work can be exploited in specific domains of applications, such as digital libraries and e-commerce, where taxonomies for objets are often well-established. Finally, the use of semantic information about objects can open very refreshing research areas which have a good potential for new applications on the Internet.

References

Aggarwal, C. C., Wolf, J. L., and Yu, P. S. (1999). Caching on the World Wide Web. *Knowledge and Data Engineering*, 11(1):95–107.

Arlitt, M., Friedrich, R., and Jin, T. (1998). Performance evaluation of Web proxy cache replacement policies. *Lecture Notes in Computer Science*, 1469:193+.

Atzmon, H., R.Friedman, and R.Vitenberg (2002). Replacement policies for a distributed object caching service. In *International Symposium on Distributed Objects and Applications (DOA)*.

Brunie, L., Pierson, J.-M., and Coquil, D. (2002). Semantic collaborative web caching. In *Proceedings of the 3rd International Conference on Web Information Systems Engineering (WISE'02)*.

Calsavara, A. and dos Santos, R. G. (2002). Um algoritmo de substituição de objetos em cache na internet baseado em semântica. In *Anais do 20. Simpósio Brasileiro de Redes de Computadores*, pages 135–150, Búzios-RJ.

Calsavara, A., dos Santos, R. G., and Jamhour, E. (2003). The least semantically related cache replacement algorithm. In *Proceedings of the IFIP/ACM Latin American Network Conference*, pages 21–34. Nova York: ACM.

Calsavara, A. and Schmidt, G. (2004). Semantic search engines. In Ramos, F. F., Unger, H., and Larios, V., editors, *Advanced Distributed Systems: Third International School and Symposium, ISSADS 2004, Guadalajara, Mexico, January 24-30, 2004. Revised Selected Papers*, volume 3061 of *Lecture Notes in Computer Science*, pages 145–157. Springer.

Cao, P. and Irani, S. (1997). Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, Monterey, CA.

Cheng, K. and Kambayashi, Y. (2000). Advanced replacement policies for WWW caching. In *Web-Age Information Management*, pages 239–244.

Chidlovskii, B., Roncancio, C., and Schneider, M.-L. (1999). Semantic cache mechanism for heterogeneous web querying. *WWW8 / Computer Networks*, 31(11-16):1347–1360.

Dar, S., Franklin, M. J., Jónsson, B. T., Srivastava, D., and Tan, M. (1996). Semantic data caching and replacement. In *Proc. VLDB Conf.*, pages 330–341, Bombay, India.

dos Santos, R. G. (2001). Internet object cache replacement based on semantics. Master's thesis, Pontificia Universidade Catolica do Parana, Curitiba, Brazil.

Keller, A. M. and Basu, J. (1996). A predicate-based caching scheme for client-server database architectures. *VLDB Journal*, 5(1):35–47.

Liu, J. S. (2001). *Monte Carlo Strategies in Scientific Computing*. Springer Verlag. 360 pages.

Murta, C. D., Almeida, V. A. F., and Meira Jr., W. (1998). Analyzing performance of partitioned caches for the WWW. In *Proceedings of the 3rd International WWW Caching Workshop*.

O'Neil, E. J., O'Neil, P. E., and Weikum, G. (1993). The LRU-K page replacement algorithm for database disk buffering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 297–306.

Pinheiro, J. C. (2001). Avaliação de políticas de substituição de objetos em cache na web. Master's thesis, Instituto de Ciências Matemáticas e de Computação de São Carlos – USP.

Podlipnig, S. and Boszormenyi, L. (2003). A survey of Web cache replacement strategies. *ACM Computing Surveys*, 35(4):374–398.

Ren, Q., Dunham, M. H., and Kumar, V. (2003). Semantic caching and query processing. *IEEE Transactions on Knowledge and Data Engineering*, 15(1):192–210.

Rizzo, L. and Vicisano, L. (2000). Replacement policies for a proxy cache. *IEEE/ACM Transactions on Networking*, 8(2):158–170.

Shmidt, G. (2002). Semantics server for internet resources: an approach based on rdf. Master's thesis, Pontificia Universidade Catolica do Parana, Curitiba, Brazil.

Silberschatz, A. and Galvin, P. B. (1994). *Operating Systems Concepts*. Addison-Wesley, Reading, Mass., fourth edition.

Vakali, A. (2001). Proxy cache replacement algorithms: A history-based approach. *World Wide Web*, 4:277–298.

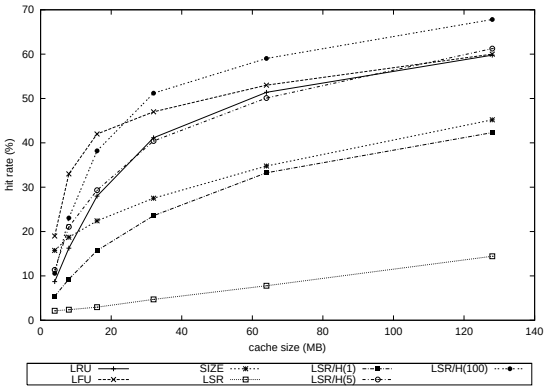
Wang, J. (1999). A survey of web caching schemes for the internet. *ACM Computer Communication Review*, 29(5):36–46.

Williams, S., Abrams, M., Standridge, C. R., Abdulla, G., and Fox, E. A. (1996). Removal policies in network caches for World-Wide Web documents. In *Proceedings of the ACM SIGCOMM '96 Conference*, Stanford University, CA, USA.

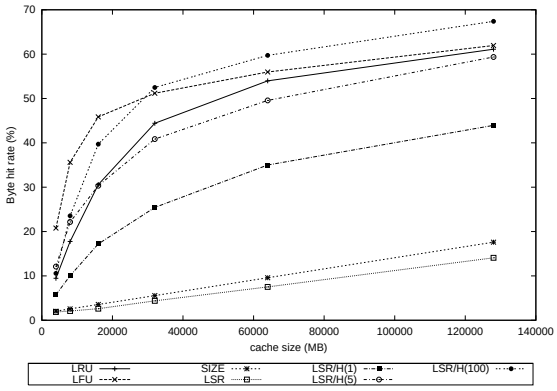
Wolman, A., Voelker, G. M., Sharma, N., Cardwell, N., Karlin, A. R., and Levy, H. M. (1999). On the scale and performance of cooperative web proxy caching. In *Symposium on Operating System Principles*, pages 16–31.

Wooster, R. P. and Abrams, M. (1997). Proxy caching that estimates page load delays. *Computer Networks*, 29(8-13):977–986.

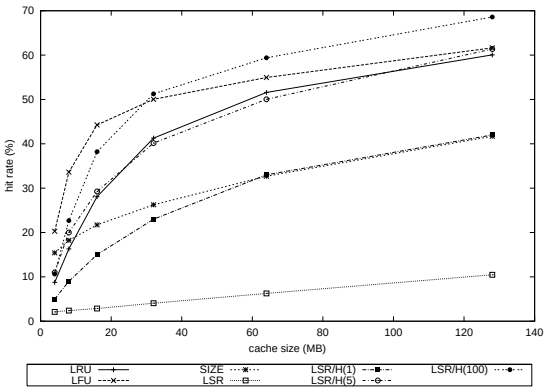
Zheng, B., Xu, J., and Lee, D. L. (2002). Cache invalidation and replacement strategies for location-dependent data in mobile environments. *IEEE Transactions on Computers*, 51(10):1141–1153.



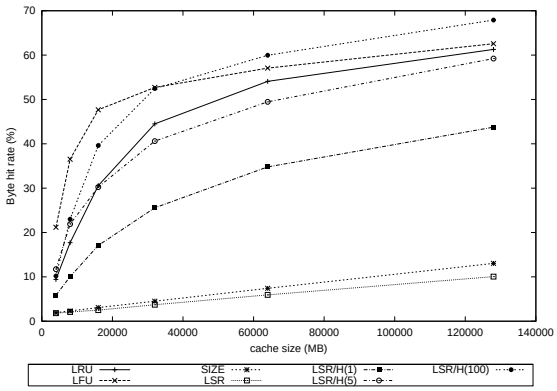
(a) hit rate after 0.5 million requests



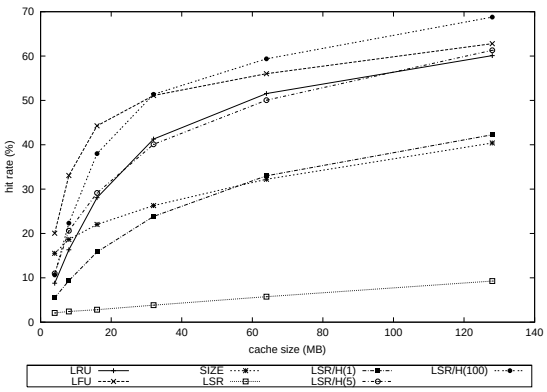
(b) byte hit rate after 0.5 million requests



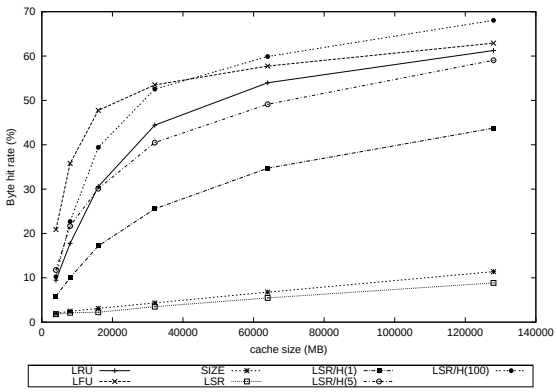
(c) hit rate after 1.0 million requests



(d) byte hit rate after 1.0 million requests



(e) hit rate after 1.5 million requests



(f) byte hit rate after 1.5 million requests

Figure 5. Performance of object cache replacement strategies