Efficient Reduction of Web Latency through Predictive Prefetching on a WAN

Christos Bouras[1, 2], Agisilaos Konidaris[1, 2], and Dionysios Kostoulas[2]

[1] Research Academic Computer Technology Institute – CTI, Riga Feraiou 61, GR-26221, Patras, Greece
{bouras, konidari}@cti.gr
[2] Computer Engineering and Informatics Department, University of Patras, GR-26500, Patras, Greece
kostoula@ceid.upatras.gr

Abstract. This paper studies Predictive Prefetching on a Web system that provides two levels of caching before information reaches the clients. We analyze prefetching on a Wide Area Network with the above mentioned characteristics. First, we provide a structured overview of predictive prefetching and show its wide applicability to various computer systems. The WAN that we refer to is the GRNET academic network in Greece. We rely on log files collected at the network's Transparent cache (primary caching point), located at GRNET's edge connection to the Internet. We present the parameters that are most important for prefetching on GRNET's architecture and provide preliminary results of an experimental study, quantifying the benefits of prefetching on the WAN.

1 Introduction

Prefetching is a technique used to enhance several computer system functions [3, 4, 5, 6, 9, 11, 13]. It has been used to enhance operating systems, file systems and of course Web based systems. It is always useful to be able to foresee a request in such systems, in order to be able to service it before it is actually placed, since this would boost system performance. Prefetching systems always run the risk of misusing or even abusing system resources in order to execute their functions.

The ultimate goal of Web prefetching is to reduce what is called User Perceived Latency(UPL) on the Web [1, 2, 7, 8, 12]. UPL is the delay that an end user (client) actually experiences when requesting a Web resource. The reduction of UPL does not imply the reduction of actual network latency or the reduction of network traffic. On the contrary in most cases even when UPL is reduced, network traffic increases. The basic effect of prefetching on a Web system is to "separate" the time when a resource is actually requested by a client from the time that the client (user in general) chooses to see the resource.

G. Dong et al. (Eds.): WAIM 2003, LNCS 2762, pp. 25–36, 2003.

2 Caching in the Wide Area

Local Area Networks (LANs) usually include several clients configured to use a single Proxy/cache server, which in turn is connected and provides access to the Internet. In this work we look at the case of several LANs inter-connected with the use of a broadband backbone that provides access to the Internet through one main access point. This is the case of the Greek Research Network, GRNET [10]. In the case of the GRNET WAN we find 3 levels of caching. The first is the simple client browser caching mechanism, the second is the LAN Proxy server caching mechanism and the third is a Transparent caching mechanism implemented at the Athens edge router node, where the WAN is connected to the Internet. The simplified Prefetching architecture (without network equipment such as switches, routers etc) that we will study in this paper is presented in Figure 1. The Transparent cache initiates demand requests to Web servers represented by normal directed lines but also prefetching requests represented by the dotted directed lines between the Transparent cache and the Internet.

3 The n Most Popular Approach

In the following sections we present the two approaches that we followed for prefetching at the Transparent cache. First, we present the "Most popular" document approach and then the Prediction by Partial Matching (PPM) approach.

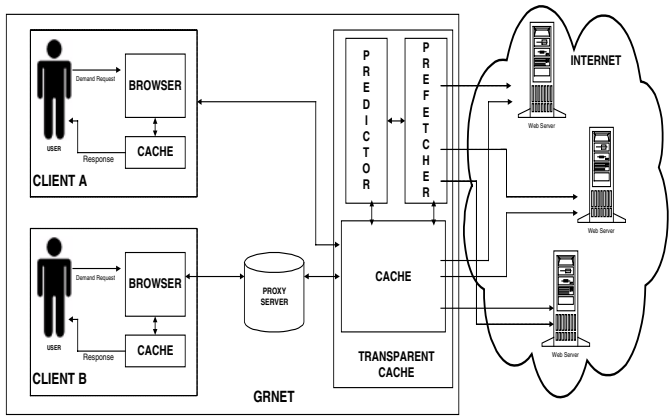

Fig. 1. The Prefetching architecture used in this paper

3.1 Popularity Ranking

Page popularity ranking is a process of log data analysis used to determine pages that are likely to be requested next. This process is applied for each user separately (in case predictions are based on user log data only) and overall (in case predictions are based on log data by all users). As mentioned in previous sections, user log data is

separated into session files. Therefore, page popularity ranking is first carried out at a session level. For any page in a user's session, all instances of it in that session are found and a popularity list is created, on the top of which exists the page that has been requested most by the user in the current session. Adding up page popularity data from all user sessions, we create a new popularity list that represents user's navigation habits in general (user-based page popularity ranking). In the same way, another page ranking is performed, which takes data from all users into account (overall page popularity ranking).

However, the basic process of log data analysis is that of finding for every web page those requested more frequently after it. We look for pages that were accessed within *n* accesses after a specified page. The parameter *n* is called lookahead window size. We choose the lookahead window size to be equal to 5. Any page requested within *n* accesses after a specified page was requested is considered to be a n-next page of it. For every page in the log data we find the frequency of visits within the lookahead window size, of all other pages found to be a n-next page to it.

Algorithm 1. Building the prediction model for the user-based "most popular" approach

Input: Prediction model constructed so far, user's training log data
Output: Updated prediction model

```
For every request in user's training log data:
    Set user's current request as active
        For each user's request within n requests after active:
            If request not in active request's n-next popularity list:
                Insert request in active request's n-next popularity list
                Request popularity = 1
            If request in active request's n-next popularity list:
                Request popularity++
Sort active request's n-next popularity list by popularity
```

3.2 Decision Algorithm

In a simple form of the decision algorithm, prefetching is decided for any page suggested by the prediction algorithm. We characterize this decision policy as an aggressive prefetching policy. However, when available bandwidth is limited we need to restrict our model and perform prefetching only for those predicted pages that appear to have a high chance of being requested. Those are the pages with high dependency to the currently displayed page or pages whose content does not seem to change frequently. This decision policy is called strict prefetching policy. The whole prediction process for the case of the user-based prediction is shown in Algorithm 2.

Algorithm 2. Predicting a user's next request given the prediction model, the current (last) request, the dependency threshold and the frequency of change threshold

Input: Prediction model, user's current request, dependency threshold, frequency of change threshold
Output: A set of predicted pages

```
For every request in user's simulation data:
  Set user's current request as active
    For each of m most popular requests in active request's n-next popularity
    list:
      Check request as prefetching candidate
      If policy == aggressive:
        Add request in set of predicted pages
      If (policy == strict) && (size > average size):
        If (dependency > dependency threshold) && (frequency of change <
        frequency of change threshold):
          Add request in set of predicted pages
```

4 The Prediction by Partial Matching Approach

Prediction by Partial Matching (PPM) is a context modeling algorithm which keeps track of the occurrence of events and their sequence. It then provides (and always keeps track of) a probability for the occurrence of the next event based on previous event occurrence and sequence. In our case an event is a request to a URL. We keep track of the URLs being requested by users (through traces) and build an m context trie, representing their occurrence and their sequence. Next, we describe all algorithms used to evaluate prefetching at the Transparent cache with the use of PPM as the prefetcher. It is not in the scope of this work to elaborate on PPM due to space limitations. The procedure we follow can be found in [14].

The algorithms used to evaluate PPM at the Transparent cache are presented next.

Algorithm 3. Building prediction model from users' access patterns (training process)

Input: Structure representing prediction model of order m constructed so far, user's training log data
Output: Updated prediction model

```
Current context [0] = root node of structure
For every request in user's training log data:
  For length m down to 0:
    If request not appearing as child node of current context [length] in struc-
    ture:
      Add child node for request to current context [length]
      request occurrence = 1
      current context [length+1] = node of request
    if request appearing as child node of current context [length] in structure:
      request occurrence ++
      current context [length+1] = node of request
    current context [0] = root node of structure
```

Algorithm 4. Prediction Process (predicting user's next request given the prediction model, the previous requests and the confidence threshold for each order of model)

Input: Structure representing prediction model of order m, previous k requests, confidence threshold for each order of prediction model
Output: A set of predicted pages

For length 1 to k:
Current context [length] = node representing access sequence of previous length requests
For length k down to 1:
For each child node of current context [length]:
If (request occurrence for child node / request occurrence for parent) >= confidence threshold for order length:
Add request of child node in set of predicted pages
Remove duplicates from set of predicted pages

5 Experimental Study

In order to evaluate the performance benefits of the two prefetching schemes introduced in the previous paragraphs, we use trace-driven simulation. As mentioned access logs of GRNET Transparent cache are used to drive the simulations. The results presented in this paper are based on logs of web page requests recorded over a 7-day period. In each execution, simulation is driven on requests of a different group of users. In all experiments, 80% of the log data is used for training (training data) and 20% for testing (testing data) to evaluate predictions. Furthermore, all traces are preprocessed, following the five filtering steps described in a previous section.

The basic performance metrics used in our experimental study are *Prefetching Hit Ratio, Usefulness of Predictions* and *Network Traffic Increase*.

The three evaluation metrics mentioned above are used for both the "Most popular" approach and the PPM approach. However, two additional metrics are used to evaluate the performance of the "Most popular" prefetching model. These are *Average Rank* and "*Most popular" prefetch effectiveness.*

5.1 Most Popular Approach Evaluation

If user-based prediction is performed and all pages suggested by the prediction algorithm are prefetched, the prefetching hit ratio appears to be equal to 48%, for prefetching window size equal to 5. Usability of predictions is equal to 27,5% and traffic increase per request is found to be 6%. Moreover, the average rank of successfully prefetched documents is 2 and the prefetch effectiveness equals 52%. If we use n-next popularity data obtained from the general population in our predictions, instead of user log data, prefetch effectiveness is a bit higher (54%). However, this requires an 18% traffic increase. This is expected, since in the case of overall prediction there is greater availability of n-next popularity data. Therefore, prefetching is performed for more requests and more documents are prefetched. As a result, the cost in bandwidth is greater, but prefetch effectiveness is higher. If traffic increase is limited to 8%, then

prefetch effectiveness will be equal to 50%. It is clear that for the same traffic increase the performance results of user-based prediction are better than those of overall prediction since user data implies more accurately the user's future web behavior.

It is obvious that a small value of the prefetching window size provides better accuracy of predictions. Actually, the less documents a client is allowed to prefetch, the higher its prefetching hit ratio will be, as only highly probable objects are going to be prefetched. Practicing simulation for a prefetching window size equal to 3 and user-based prediction, we experience a significant increase of hit ratio (58%) compared to the case of a prefetching window size equal to 5 (48%). In addition, less bandwidth is required to apply the prefetching method (4% traffic increase). However, usability of predictions is lower (25%) compared to the case of a prefetching window size equal to 5 (27,5%), as less prefetching actions are performed. The results of overall prediction for m equal to 3 are similar to those taken for m equal to 5.

Generally, results of applying prefetching are good when: (i) a lot of prefetching is being done and (ii) prefetching is successful. In the first case, substantial traffic increase is required. Therefore, if we want to keep bandwidth cost low, we need to improve the prediction ability of our prefetching model. This can be done either by improving the prediction algorithm, so as to increase its accuracy of predictions, or by applying prefetching only for those cases that predictions seem to be secure. Since no prediction process can ensure success, we try prefetching those documents that appear as prefetching candidates and have good chances of being requested. A web page has a greater chance of being requested next, if its dependency to the currently displayed page is high. Furthermore, we need to avoid prefetching for pages that change regularly. Therefore, security thresholds are used. In our experiments, if a document has a greater size than the average, it is prefetched only if its dependency is greater than 0.5 and its frequency of change is not greater than 0.5. In this case, the prefetching window size refers to the maximum number of prefetched pages.

For user-based prediction and a prefetching window size equal to 5, the prefetch effectiveness of a strict policy is almost as high as that of the aggressive policy (51%), while traffic increase is limited to 4%. It is clear that by using such a policy we manage high performance with lower cost in bandwidth, as the number of prefetched objects is limited. Actually, comparing these results (m = 5) to those of the aggressive policy (m = 5 and m = 3), we see that we experience prefetch effectiveness almost equal to that of the aggressive algorithm for m = 5, but with traffic increase equal to that of the m = 3 case. Moreover, hit ratio is higher (51%) for the same prefetching window size, with little cost in usefulness of predictions. Table 1 shows results taken for all three cases of user-based prediction. Figure 5 compares performance results of user-based and overall prefetching.

Table 1. Performance results for user-based "most popular" prediction

	Hit Ratio	**Usefulness of predictions**	**Prefetch Effectiveness**	**Network Traffic Increase**	**Average Rank**
Aggressive policy, m = 5	48%	27,5%	52%	6%	2
Aggressive policy, m = 3	58%	25%	42%	4%	1
Strict policy, m =5	51%	27%	51%	4%	2

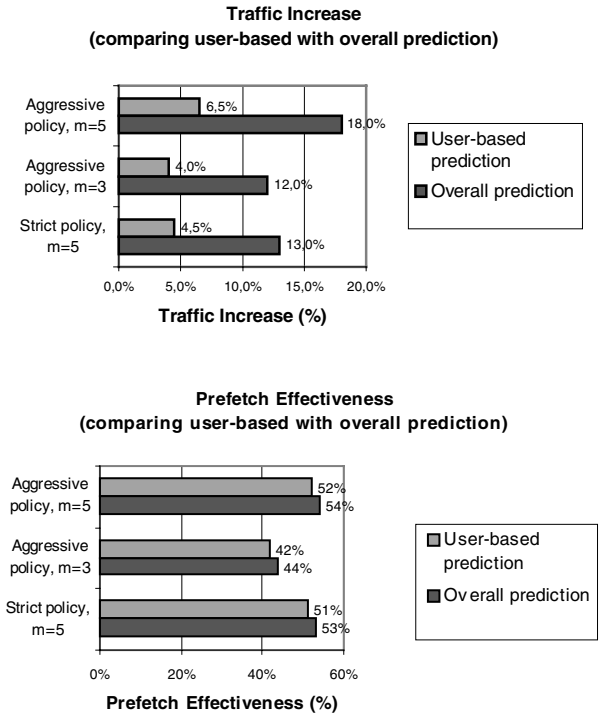

Fig. 2. Comparison of user-based and overall prediction scenarios for all policies (graphs should be read in pairs of bar charts)

5.2 Prediction by Partial Matching Approach Evaluation

Large values of confidence cause fewer predictions to be made, thus, reducing the network traffic. Additionally, the accuracy of predictions increases since only the highly probable pages are prefetched. However, the usefulness of predictions decreases due to the limited number of prefetched pages.

Figure 6 shows that for values of confidence between 0 and 0.3 any slight increase in the confidence's value results in significant increase of the hit ratio with less significant cost in the usefulness of predictions. Additionally, values greater than 0.7 have the opposite effect on performance parameters. Therefore, values of confidence between 0.3 and 0.7 are preferable as they provide better performance results.

If higher order contexts are trusted more than low order contexts and are assigned a smaller confidence, we have the case of weighted confidence. Figure 7 compares the cases of constant and weighted confidence. The horizontal lines represent the performance of the algorithm with weighted confidence which takes the values of 0.8 to 0.6 (reducing in steps of 0.1) for orders 1 to 3. This algorithm is compared with three

others that use constant confidence from 0.6 up to 0.8. Diagrams show that weighted confidence performs above the average of the three constant-confidence algorithms.

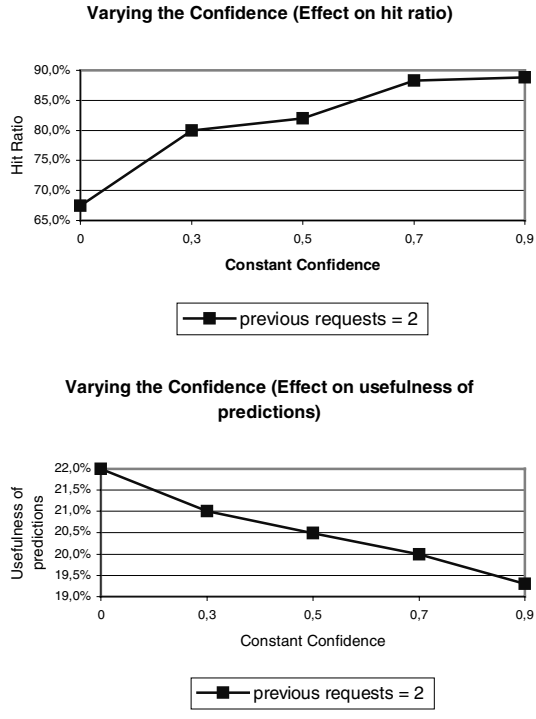

Fig. 3. Varying confidence. Effect on performance of PPM approach

5.3 Comparing "Most Popular" and PPM Performance

Figure 9 shows that in any case the PPM approach has significantly better accuracy of predictions and less traffic increase than the "Most popular" approach. On the other hand, the "Most popular" algorithm has higher usefulness of predictions. It is necessary to mention here that the complexity of the "most popular" case is much less than that of the PPM case, since a simple algorithm, with no special operational or storage requirements, is used for the construction of the "n-next most popular" prediction model.

It is obvious that the performance of the Prediction by Partial Matching approach on any of the specified metrics varies depending on the values of the parameters. If an aggressive policy that allows many pages to be predicted is used, usefulness of predictions will be high, causing, however, high network overhead and low hit ratio. On the other hand, a strict policy will result in accurate predictions and reduced traffic increase but its coverage will be limited. Two such cases are shown in Table 2. The

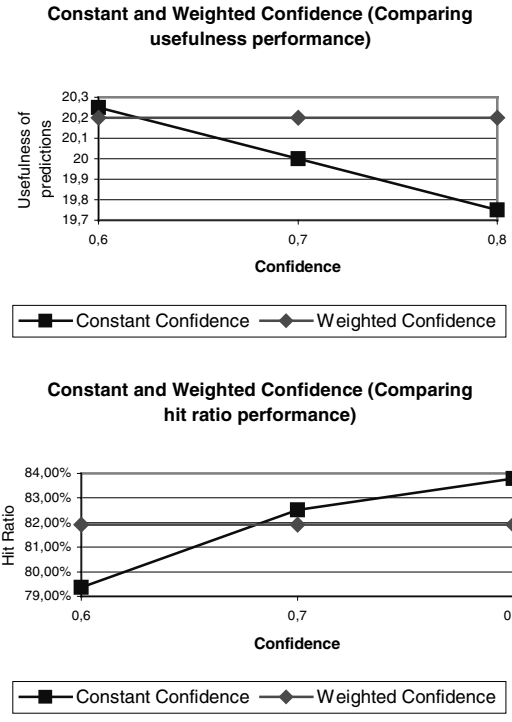

Fig. 4. Comparing constant and weighted confidence

first predicts almost a quarter of the total number of requests with hit ratio equal to 73%. The second one has higher accuracy (85%) but usefulness shrinks to 18,25%.

To have a clearer picture of performance differences of the two prefetching approaches we try to find proportional cases of them. For the case of the "Most popular" aggressive policy with prefetching window size equal to 5 we choose the case of PPM with previous requests equal to 1 and confidence equal to 0. We choose these values for the PPM parameters as the "Most popular" case uses also one previous request in predictions (the last request) and has no confidence limitations applied. For the case of the "Most popular" strict policy with a prefetching window size equal to 5, we choose the case of PPM with previous requests equal to 1 and confidence equal to 0.5. These values of PPM parameters are selected since the strict "Most popular" case uses only the last request in predictions and the threshold of the decision algorithm is also equal to 0.5. Figure 10 shows results taken for comparing these proportional cases.

All results studied in the above paragraphs clearly show that the application of prefetching in the Wide Area can be quite beneficial. Even with the use of a simple prediction algorithm, as the "n-next most popular" algorithm proposed in this paper, the

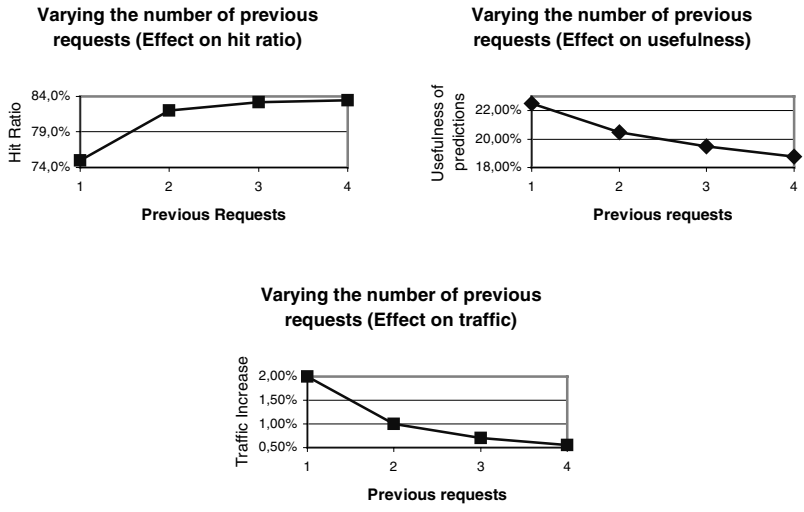

Fig. 5. Varying previous requests number. Effect on performance of PPM approach

accuracy of predictions can reach 58% (case of aggressive, user-based policy with prefetching window size equal to 3) with an insignificant for a WAN increase of network traffic equal to 4%. If a more complex algorithm is used, which is a variation ofthe Prediction by Partial Matching algorithm in this paper, we show that a high fraction of user requests (18,25%–23%) can be predicted with an accuracy of 73%–85%, while bandwidth overhead added is not higher than 2%. In fact, performance results are better, if we take into account that many of the prefetched requests will be used by more than a single end user, as prefetching in that case is performed for ICP requests made by Proxy servers.

Table 2. Efficiency of PPM approach

	Hit Ratio	**Usefulness of predictions**	**Network Traffic Increase**
Aggressive policy, previous requests = 1, confidence = 0,2-0,3	73%	23%	2%
Strict policy, previous requests = 4, confidence = 0,7-0,9	85%	18,25%	< 0,5%

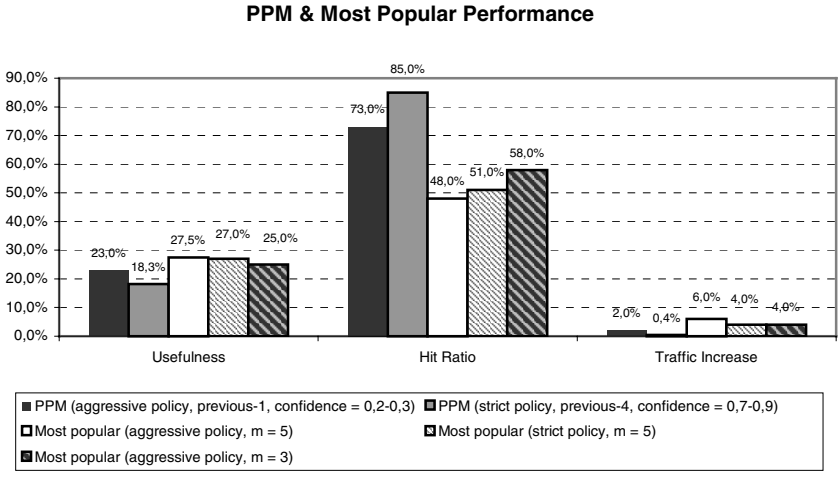

Fig. 6. "Most popular" and PPM performance for strict and aggressive prefetching policy

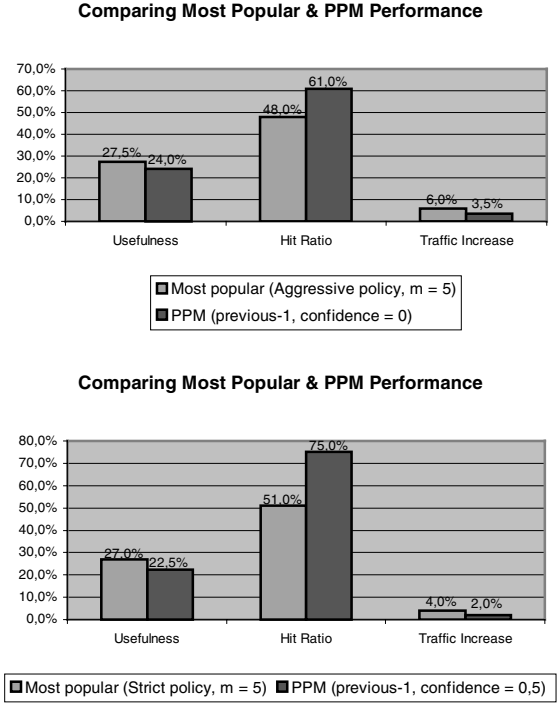

Fig. 7. Comparing "Most popular" and PPM performance (using proportional cases)

6 Future Work and Conclusions

In order to evaluate prefetching on the edge connection of a WAN to the Internet we use the GRNET WAN edge connection to the Internet, which is a Transparent cache. We first study the system and present its characteristics. After employing two different algorithms for prefetching, a “n Most Popular” approach and a PPM approach, we find that prefetching can be potentially beneficial to the GRNET WAN. Of course many further issues must be explored, before deploying prefetching on the edge of GRNET. Preliminary results provide a clear indication that UPL would be significantly reduced in GRNET if a version of prefetching was performed at the Transparent cache.

References

1. M. Arlitt, "Characterizing Web User Sessions", ACM SIGMETRICS Performance Evaluation Review, Volume 28, Issue 2 pp.50–63, 2000.
2. B. Brewington and G. Cybenko, “Keeping up with the changing Web”, IEEE Computer, 33(5) pp. 52–58, 2000.
3. X. Chen and X. Zhang, “Coordinated data prefetching by utilizing reference information at both proxy and Web servers”, ACM SIGMETRICS Performance Evaluation Review, Volume 29, Issue 2 pp. 32–38, 2001.
4. X. Chen, X. Zhang, “Popularity-Based PPM: An Effective Web Prefetching Technique for High Accuracy and Low Storage”, in Proc. of the 2002 International Conference on Parallel Processing (ICPP'02), Vancouver, B.C., Canada, 2002, pp. 296–304.
5. E. Cohen and H. Kaplan, “Prefetching the means for document transfer: A new approach for reducing Web latency”, in Proc. of IEEE INFOCOM, Tel Aviv, Israel, 2000, pp. 854–863.
6. B. D. Davison, "Assertion: Prefetching with GET is Not Good" in Proc. of The Sixth International Workshop Web Caching and Content Distribution, Elsevier, Boston, 2001, pp. 203–215.
7. B. D. Davison, “Predicting Web actions from HTML content” in Proc. of the The Thirteenth ACM Conference on Hypertext and Hypermedia (HT'02), College Park, MD, 2002, pp. 159–168.
8. D. Foygel and D. Strelow, "Reducing Web Latency with Hierarchical Cache based Prefetching", in Proc. of the International Workshop on Scalable Web Services, (in conjunction with ICPP'00), Toronto, Ontario, Canada , 2000, pp. 103.
9. E. Gelenbe and Qi Zhu, “Adaptive control of pre-fetching”, Performance Evaluation 46 pp. 177–192, Elsevier, 2001.
10. GRNET, Web Site: http://www.grnet.gr/.
11. T. I. Ibrahim and C. Z. Xu, “Neural Nets based Predictive Prefetching to Tolerate WWW Latency”, in Proc. of the 20th International Conference on Distributed Computing Systems, Taipei, Taiwan, 2000, pp. 636–643.
12. Joshi and R. Krishnapuram, "On Mining Web Access Logs", in Proc. of ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Dallas, Texas, 2000, p.p. 63–69.
13. J. I. Khan and Q. Tao, “Partial Prefetch for Faster Surfing in Composite Hypermedia”, in Proc. of USITS 2001, San Francisco, California, USA, 2001.
14. T. Palpanas and A. Mendelzon, “Web Prefetching Using Partial Match Prediction”. in Proc. of the Web Caching Workshop, San Diego, CA, USA, 1999.