
World Wide Web: Internet and Web Information Systems, 7, 143–179, 2004

Predictive Prefetching on the Web and its Potential Impact in the Wide Area

CHRISTOS BOURAS and AGISILAOS KONIDARIS {bouras,konidari}@cti.gr
Computer Engineering and Informatics Department, University of Patras, GR-26500 Patras, Greece, and Computer Technology Institute-CTI, Riga Feraiou 61, GR-26221 Patras, Greece

DIONYSIOS KOSTOULAS kostoula@ceid.upatras.gr
Computer Engineering and Informatics Department, University of Patras, GR-26500 Patras, Greece

Abstract

The rapid increase of World Wide Web users and the development of services with high bandwidth requirements have caused the substantial increase of response times for users on the Internet. Web latency would be significantly reduced, if browser, proxy or Web server software could make predictions about the pages that a user is most likely to request next, while the user is viewing the current page, and prefetch their content.

In this paper we study Predictive Prefetching on a totally new Web system architecture. This is a system that provides two levels of caching before information reaches the clients. This work analyses prefetching on a Wide Area Network with the above mentioned characteristics. We first 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. Our experimental study includes the evaluation of two prediction algorithms: an "*n* most popular document" algorithm and a variation of the PPM (Prediction by Partial Matching) prediction algorithm. Our analysis clearly shows that Predictive prefetching can improve Web response times inside the GRNET WAN without substantial increase in network traffic due to prefetching.

Keywords: Web Prefetching, prediction algorithms, caching, trace based simulation

1. Introduction

Prefetching is a technique used to enhance several computer system functions. It has been used to enhance operating systems, file systems [22] 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 always introduces a very important (and dangerous in some cases) trade-off: system resources use versus prediction accuracy. If more resources are used and more pages are prefetched then it is more probable for some of them to be accessed. However, the accuracy of predictions would decrease. Prefetching systems always run the risk of misusing or even abusing system resources in order to execute their functions. At this point it is useful to present a general definition of the "perfect" prefetching system:

"A perfect prefetching system is the system that is able to accurately predict the next (or a number of next) user or system requests and pre-execute them (= execute them before they are placed) just by using idle system resources".

Prefetching is a large issue and it is not in the scope of this paper to analyze and evaluate all prefetching systems. We focus on Web object prefetching and specifically on an enhanced Client/Proxy/Web server architecture. Prefetching is not an issue that should be taken lightly on the WWW. Unlike caching, its potential positive effects may well turn into extremely negative effects, in terms of resource misuse, if it is not implemented correctly.

The ultimate goal of Web prefetching is to reduce what is called User Perceived Latency (UPL) on the Web. 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. A user perceives Web latency as the time between issuing a request for a resource and the time it is actually displayed in the browser window. The nature of the Web itself masks all the underlying network and protocol functions that are executed in order to satisfy a user request, so a user does not have to care when they are actually executed. It is clear that UPL must not be confused with request latency or network latency.

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. In order to be more formal we outline this separation with the use of a timeline (see Figure 1). In the first timeline (SCENARIO A), where prefetching is not performed, we see that a client request for resource R_i at t_1 causes the request for resource R_i to be placed on the network. The time that is needed for a response to reach the client, as an answer to its request, is equal to $t_2 - t_1$. In this case the client that issued the request experiences all of $t_2 - t_1$ as latency between its request and the reception of a response.

On the other hand, when we look at the second timeline (SCENARIO B), a client request is separated in time from the actual request to the same resource (R_i). Prefetching causes the placement of the actual request in advance (the significance of the $t_1 - t_0$ period is explained later). At this point it is useful to point out that $t_2 - t_1 = t_4 - t_0$. This is true since a request for resource R_i requires the same amount of time to be serviced under

Figure 1. Timeline representation of prefetching.

the same circumstances (network, server etc). In this case though, the user perceives the response as quicker than scenario A since $t_2 - t_1 > t_4 - t_1$.

The example presented in Figure 1 brings forward two of the basic issues related to Web prefetching. The first is the optimization of $t_1 - t_0$ in order for it to be:

- big enough in order for the resource R_i to be prefetched before the client requests it, and
- small enough in order for the resource R_i not to have expired before the client requests it.

The second important prefetching issue drawn from Figure 1 is the method used to foresee a client's request before it is placed since $t_0 < t_1$. The two issues mentioned here are common for all prefetching systems, but there are many more issues that must be looked into before prefetching can be deployed on any Web system. Many of these issues are directly related to the architecture of the Web system for which prefetching is designed. Prefetching does not provide any "clean-cut" or "one size fits all" solutions. Every prefetching system must be designed carefully and put to work only after an extensive trial period and after providing adequate answers to the following basic validation questions:

IF prefetching can and must be added to the specific system. Not all Web systems can be facilitated by prefetching. For example, a highly dynamic Web system may require the time tolerance ($t_1 - t_0$ in Figure 1) to be so small before a resource expires that no prefetching approach would be adequate to facilitate it.

WHO (referring to the Web system's components) will take part in the prefetching procedure. Prefetching can be facilitated by all basic components that a Web system consists of (clients, proxies, mediators and servers). One must answer to the basic WHO question before going to work, in order to produce prediction policies etc.

WHAT (referring to the different types of available resources) will be prefetched. As argued above, dynamic content is the most "difficult" candidate for prefetching. Before designing the prefetching system, this question must be answered and the file types that will take part in prefetching should be clearly depicted.

HOW (referring to the procedure) prefetching will be carried out. Prefetching can be considered as an application level protocol. This protocol must clearly state the procedures that will be followed in order to initiate, execute and terminate prefetching and also the communication procedures between the Web system components. For example, in some prefetching approaches proprietary text files containing most popular URLs are transferred between servers, clients and proxies. In order to implement the software modules that will handle these files on the appropriate Web system component the question of HOW prefetching will be applied must be answered.

WHEN (referring to the time of prefetching) a resource will be prefetched. In order to answer this question one must apply a suitable prediction algorithm that will receive a number of inputs and provide the answer to the WHEN question as the output.

It is most likely that the answers to the questions will be different for different Web system configurations. Prefetching has been explored in various Internet configurations, including the client/proxy/Web server architecture that performs prefetching at the client based on information related to the specific client only, the client/Web server architecture [18] that

is based on hinting between web system components and the client/mediator/Web server architecture which performs all prediction and prefetching activity on the mediator server based on information from many users connected to it.

In this paper we present a study on how prefetching can be performed on a Wide Area Network with two levels in its caching hierarchy: a Transparent cache on the edge of the WAN to the Internet and local Proxy servers on the edge of the backbone. There are two intuitive reasons that led us to study the benefits of prefetching in the Wide area. The first reason is based on the basic drawback of prefetching in all other configurations. Even if one adopts a perfect prediction system and applies it to one of the configurations mentioned above, the problem of competing original demand and prefetching requests will continue to exist. Keeping in mind that in all these configurations, only a very limited amount of bandwidth can be used by prefetching, we find that if we take prefetching to a higher level, the level of a Wide Area Access Point, this problem may be tackled more easily. This is based on the observation that even though Wide Area Networks (especially in Greece) have been upgraded a number of times in recent years, home users still access the Internet through use of the 56 kbps modem. The second reason for applying prefetching in the Wide Area is that a successful prefetching request may be useful to several users. This is so, due to the two levels of caching that follow prefetching. Both ICP and TCP requests are taken into consideration by our prefetching approach.

This work first presents a methodology for studying the prefetching problem and all the factors that influence a prefetching system during its design and during its appliance. This methodology stems from our experience gained from the study of prefetching on the GRNET WAN. We believe that the factors mentioned in the upcoming paragraphs must be taken into consideration very seriously during the design and implementation of every prefetching system, despite of the configuration of the Web system it aims to enhance.

This paper is constructed as follows. In Section 2 we present related work. In Section 3 we present all considerations that must be made in order to design a prefetching system. Section 4 presents the architecture of the GRNET WAN that we use for our experiments and Section 5 presents the basic characteristics of the log files used for our evaluation. Sections 6 and 7 provide the theoretic background of the two algorithms used for prediction. In Section 8 we present our experimental findings and we close this paper by presenting some possible future work in Section 9 and our conclusions in Section 10.

2. Related work

In this section we present other work that is related to all the techniques and procedures that make up a prefetching system. Due to its possible deployment variations, prefetching is closely related to several Web, Database, Network, Algorithmic and Data Mining procedures. This related work section refers to those procedures that were studied, and influenced our work in some way.

Prefetching as a general concept for speeding up several computer functions is not new. It has been proposed and used in the field of file prefetching and operating systems [22,42]. Under this approach programs inform operating systems of the files they will be accessing

or using next, and the operating system prefetches them into memory in order for them to be accessed quickly. In [21,27] prefetching is also presented under a file system approach in order to enhance file access performance.

The Web has proved to be a field where prefetching may have very positive effects. Prefetching in the Web has been mainly proposed as a complementary procedure to caching. Very useful overviews of caching and prefetching are presented in [45] and [46], respectively. Caching and prefetching are techniques that have been studied together and their combined results have been presented in many papers [20,30,34,39,45,48]. The limitations of caching [30] were the reason for the exploration of prefetching. But prefetching has not only been studied as a Web page (web object in general) handling procedure. Prefetching has also been studied as a procedure with applications to other Web related functions. In [8] the authors provide evidence that prefetching can be very useful to DNS queries and to the TCP warming period at the establishment of a new TCP connection. A technique for prefetching composite-multimedia rich pages by using partial prefetching is presented in [28].

Many different prefetching architectures have been studied. In [1,3,12,36–38] predictive prefetching is studied in a Client/Web Server architecture. In [5,9,24,30,32] predictive prefetching is studied in a Proxy/Web Server architecture. In this category we must also include Wcol [47] which is an independent proxy that parses HTML and prefetches Web objects. In [19,31] predictive prefetching is studied in a Client/Proxy architecture. The architecture that will be studied in this paper is a hierarchical caching architecture where prefetching is carried out at a WAN access point to the Internet. A similar caching architecture is presented in [20].

A very important factor in prefetching is the rate of resource change on the Web. As stated in previous paragraphs, prefetching must be carried out at a time between $t_1 - t_0$ (Figure 1) in order for it to be useful. Let us assume that prefetching takes place at a time t_5 where $t_0 < t_5 < t_1$. One has to make arrangements in order for the prefetched resource not to be changed during the period $t_1 - t_5$ since it would not be useful in this case. The Web's rate of change has been studied in [4,17]. These studies provide an insight of how the Web evolves and how one can model the change of a Web resource.

Several prediction algorithms and techniques have been proposed for prefetching. These algorithms attempt to predict future client requests and prefetch them. The Prediction by Partial Match (PPM) algorithm which originates in the Data Compression community [7,25,44] has been explored in depth for Web prefetching. In [19,37,38] PPM is used to widely create branches from historical URLs. The PPM tree is given a fixed height. In [40] PPM is used and stores only long branches with frequently accessed URLs which are called LRS (Longest Repeated Sequence). In [36] the authors show that client latency can be reduced by 45% with the use of PPM and at the same time network traffic is increased by 50%. In [6] the authors use a variation of PPM called Popularity-Based PPM where the Markov prediction tree grows dynamically with a variable height in each branch. Cohen et al. [9], Kroeger et al. [30], Mogul [34], and Rosu et al. [41] use hint based prefetching. Kokku et al. [29] present Prefetch-Nice, an approach that does not treat demand and prefetch request equally guaranteeing they will not compete. In [15] the HTML content of a page is used to predict future Web requests, since authors observe that anchor text context

can be very beneficial to prediction when captured. Neural Net techniques have also been used for prediction [26]. Random walks and algorithms originated from Digital Signal Processing (DSP) are used to make prediction in [12]. Finally, Data Mining techniques and other algorithms with Markov models are used for predictions in [2,10,16,35,49].

Almost all prediction algorithms use historical access data found in Web access logs in order to predict future requests. Web log mining [13,27,48] is a very important aiding technique for prefetching. The information that can be found in Web access logs varies, according to the format of the logs and the log data selections that are usually made by administrators. Insufficient log data is a main source of inaccuracies for predictions.

In this paper two different prediction algorithms are proposed for prefetching in the Wide Area. Both approaches perform trace-driven simulation and HTML data of pages is not examined as in [15]. First we present an "n-next most popular" algorithm. This approach uses popularity ranking of user's past requests like the "Top-10 Approach" proposed in [32]. However, prediction is based only on the most popular documents found to have been requested after a given server's document, restricting predictions only to those pages that appear to be highly related to the currently displayed page. Furthermore, a page dependency threshold is used in a similar way with [36] to keep the amount of prefetched documents low in case of bandwidth limitations. Experimental results indicate that traffic increase because of prefetching is not as significant in our case as in [36]. Finally, our "most popular" algorithm uses a much more simple process for the construction of the prediction model than Data Mining [35] or Markov model [49] approaches, serving our primary goal of applying prefetching which is to reduce the response times on WWW.

Apart from a "most popular" approach our study includes a variation of the PPM prediction algorithm. A version of the partial match prediction technique is also presented in [19]. The authors state that 12% of the requests are prefetched with an increase in traffic equal to 18% and a reduction of latency less than 10%. However, a significant part of this reduction is attributed to the caching effect of the prefetch buffer. Both [19] and our PPM scheme use a number of past events to predict future ones. The maximum number of previous events taken into consideration defines the maximum order of the prefetching model and predictions from different orders are combined giving preference to the more accurate higher order ones. Unlike [19] where a constant threshold on probability of access is used for all candidates to select those that will actually be prefetched, we employ a more powerful, weighted threshold (confidence) that gets a different value for each order of the model. Higher order predictions are trusted more and are assigned a smaller threshold, whereas lower order candidates need to "prove" higher probability of access in order to be prefetched. As we show later in this paper the use of a weighted threshold improves the performance of the PPM algorithm. Moreover, the PPM approach presented in this paper predicts the future behavior of a client based on patterns from this specific client only. Although this approach may slightly reduce the applicability of prefetching, it provides well more accurate predictions than [19] where patterns from all clients' references are used to predict the behavior of a particular client. Results appear to be significantly better for our approach since 23% of the requests are predicted with an accuracy of 73% and with network traffic increased no more than 2%. Furthermore, reduction of UPL due to prefetching is greater in our case (13.6%).

3. Web prefetching considerations

There are several considerations that must be made in order to perform prefetching on the Web. In this section we present these considerations in a structured manner. If one decides to perform prefetching on a specific Web system there are several tasks that must be executed in order to evaluate the suitable prefetching policy for the specific system and then perform it efficiently. We first make the distinction between "Auxiliary prefetching" considerations and "Basic prefetching" considerations.

3.1. Auxiliary prefetching considerations

These are considerations that must mainly be made, before performing prefetching. They are issues that must be decided on before even beginning to design a prefetching system, since the efficiency of the whole prefetching system will rely on the decisions made at this stage.

3.1.1. Log analysis In order to perform predictive prefetching, researchers usually rely on traffic logs. Most prefetching policies rely on historical data to determine the most likely to be requested next pages and prefetch them. Several problems arise in the process of log analysis. Most of these problems have to do with incomplete information in the log files. As shown in [13] many traffic logs are imperfect and do not provide the information that is actually needed to evaluate prefetching. Furthermore, traffic logs used to evaluate a specific prefetching policy that will be applied on a specific system, should trace requests of the population to which prefetching will be applied. Thus, the intended target group, that will utilize prefetching, should be the same as the one that created the traffic logs used to evaluate prefetching in advance. Another issue that must be considered in traffic logs is the detail and the accuracy of the logged values. For example, M. Arlitt [1] reports that their analysis is incomplete or even flawed to some extent because the "request time" parameter included in the World Cup 1998 Web site logs did not include milliseconds but only seconds. In this paper we use log data from the primary caching point of the WAN. Traffic logs are processed before applying any prefetching policy in order to make them suitable for analysis.

3.1.2. Session estimation The notion of a session is very important in the evaluation of a prefetching policy. Web user sessions can formally be defined as "*A sequence of requests made by a single end-user during a visit to a particular site*" [33]. The previous definition can be elaborated much further. There are two points that need to be clarified in order for a session to be useful in the context of prefetching. These are:

- *The time a session begins and the time that a session ends.* This issue is very important and is thoroughly presented in [33]. Usually an idle time threshold is used in order to identify the end of a session. Of course the value of this time threshold plays an important role in log analysis and prediction algorithms, as shown is upcoming paragraphs.

- *The Web site or Web experience orientation of a Web session.* In the definition of [33] a Web session is characterized in the context of navigating a single Web site. This is obvious since the authors of [33] as in [1] examine single site Web traffic logs. In order to evaluate a prefetching policy for more than a single Web site it is obvious that the definition of the session must change. The alternative, more general definition of a Web session, as it will be used in this paper, is the following: "*A Web session is a sequence of requests made by a single end-user starting from the time that he/she visits a site and ending when he/she has not issued a request for a period of time equal to a specific time threshold*".
  This definition makes it easier to perform user specific prefetching for all the Web sites that he/she visits during a Web experience.

Web prefetching is highly related to how a system will define Web sessions. To be precise, the prediction algorithm used by a prefetching policy is highly dependent on the definition of a session. Thus, it is useful, in order to determine a suitable prediction algorithm, to experiment on the session length of the specific target user. It is shown in [1] that by changing the session length, different results on caching and prefetching may be extracted.

The results showing the effect of the session timeout threshold on the number of sessions and the number of requests in each session that we found in the log files used in this paper are shown in Figures 2 and 3.

Figure 2 clearly shows that the vast number of user sessions show a timeout value between 1 and 100 seconds. This means that most of the clients recorded in the log files that we used are very active. This is an expected result since many of the clients that are connected to the Transparent cache are Proxies which in turn service many clients. The mean value of the session timeout is close to 600 seconds or 10 minutes. This is the value that we selected in the evaluation of our Most Popular and PPM approaches in the following paragraphs. A graph like the one in Figure 2 is useful in any case because it is a direct indication of the types of clients connected to a Prefetching point.

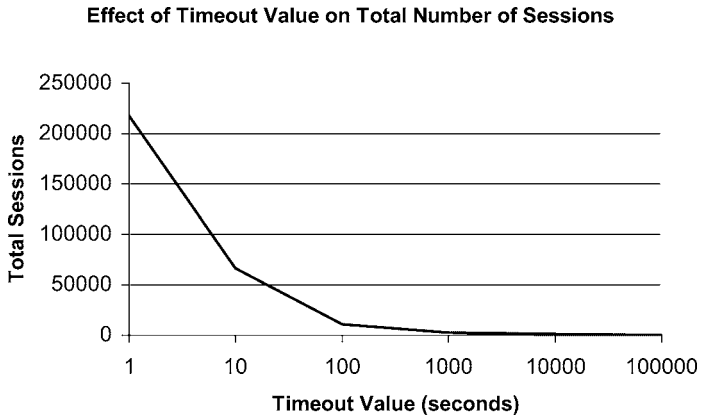

Figure 2. The effect of the timeout value on the number of sessions.

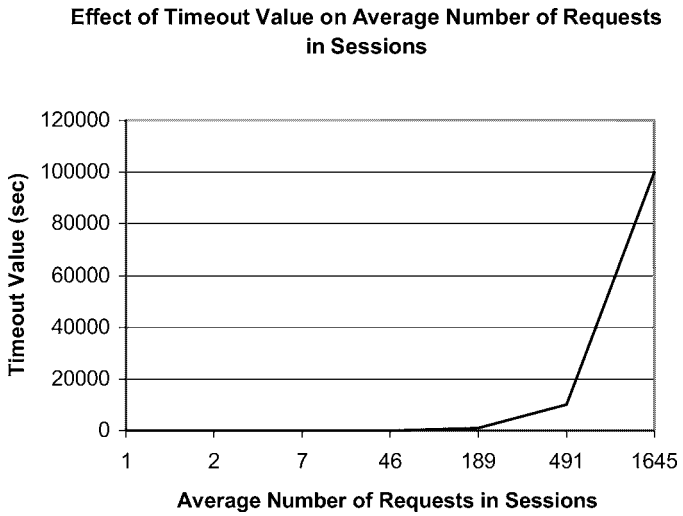

Figure 3. The effect of the timeout value on the number of requests per session.

Figure 3 also confirms the results depicted in Figure 2. It is a complementary graph that plots the effect of the session timeout value on the average number of requests found in a session. This graph is useful because it provides an idea of how many requests need to be serviced at a Prefetching point per client session. The number of requests is a clear indication of the "grace period", shown as $t_1 - t_0$ in Figure 1, which the prefetcher has, before a user places a request, in order to perform prefetching.

3.1.3. Evaluation parameters In order to determine a suitable prefetching policy for a specific Web system, it is very useful to specify the evaluation parameters that will be used to classify different policies and choose the best. The most popular parameters used for evaluating a prefetching system in the bibliography [11,30,32,36,43] are the prefetching hit ratio and the bandwidth overhead caused by prefetching. The prefetching hit ratio is usually defined as the fraction of useful prefetched objects to all prefetched objects. It is useful here to look at this fraction more closely. The result of the fraction is different if one uses a clearly numerical approach (for example 10 viewed prefetched objects out of 25 totally prefetched) and in the case that a more quantitative approach is followed (for example 2056 Kb representing prefetched objects actually viewed, out of 3876 Kb representing the total of prefetched objects). Clearly this is a choice directly related to other prefetching considerations such as available bandwidth. This also shows that even when a prediction algorithm may be very successful in predicting specific objects (e.g., jpg or gif files), overall it may cause bandwidth misuse.

3.2. Basic prefetching considerations

In the previous paragraph we presented some issues related to prefetching that must be taken into consideration before one even begins to plan a prefetching policy (prediction algorithms, etc). In this section we present the issues directly related to prefetching as a Web system (client–proxy–server) policy.

3.2.1. Objects to be prefetched A prefetching policy pre-requests Web objects that are related to the currently viewed Web page, which is always an html document (referred to as the base object in the bibliography). The objects that can be prefetched by a prefetching policy are all types of Web documents: jpg, gif, png, html, htm, asp, php, pdf, zip and others. All files that can in some way be connected to a Web page (even via a link that initiates downloading) can be included in a prefetching policy. The common practice though, is to include only "real" Web objects in prefetching policies. These objects are all documents "viewable" through a Web browser without any add-ons or plug-ins. These are html, htm, asp, php, cgi, gif, jpg, and png files (others may also be included). But even after this first refinement of objects, the question of whether all of them are "prefetchable" still stands. A very interesting approach on this issue is presented in [14]. In this work the author defines the "prefetchability" of an object as: "*A Web resource (object) is prefetchable if and only if it is cacheable and its retrieval is safe*". This definition brings up the very important issue of object expiration. It is useless to prefetch and cache an object if it expires very frequently or if it is different every time it is requested. The objects that fall into this category are dynamically created pages (asp, php, cgi, jsp, etc). Should these pages be prefetched and cached? We refer to this issue in detail during our experimental study later in this paper.

A hypertext document is usually referred to as a base document containing links to other inline objects (images for example). Log dependent prefetching schemes can not analyze the links between base documents and inline objects in an online manner. They treat objects independently. A different approach is presented in [15]. The author argues that prediction would be much simpler and more efficient if a prediction system would be aware of the HTML content of the currently viewed page before deciding what to prefetch. In a following paragraph we explain how documents embedded in a base document are treated by our log based evaluation policy.

3.2.2. System architecture The determination of a prefetching system's architecture is very important in order to correctly evaluate data from log files and retrieve valid results. The bibliography has presented prefetching architectures that include client/server systems, client/proxy/server systems and client/mediator/server systems. Furthermore, the bandwidth of the network connections between all architectural components, have a large impact on prefetching. Low-bandwidth clients, high-bandwidth clients, backbone network connections and other types of connections (cable modem, satellite, etc) influence the prefetching procedure. In this paper predictive prefetching is studied in a WAN architecture.

3.2.3. Prediction algorithm The prediction algorithm [12,35,49] is at the heart of any prefetching policy. The prediction algorithm actually defines the objects that should be prefetched. There are two basic algorithms proposed in the bibliography that implement prediction for prefetching systems. The first is the well known Prediction by Partial Matching (PPM) algorithm [8,25]. PPM has been used and evaluated in a number of prediction systems in the bibliography. The accuracy of the algorithm ranges from 40–73% depending on its parameters, and generates 1–15% extra traffic. The second prediction algorithm used is the "most popular" document algorithm [32]. Several variations have been suggested. The basic idea is to prefetch documents that have been characterized as most popular after the current document. These two approaches are followed in our work in order to evaluate the performance of prefetching in the Wide Area.

4. 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 [23].

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 cache topology of GRNET is shown in Figure 4. As shown in Figure 4, individual clients may or may not be connected to Academic LAN Proxy servers. In this case there are only two levels of caching: Client browser caching and Transparent Proxy caching. The Academic Proxy servers forward client requests that they cannot fulfill to the Transparent cache with the use of the Internet Caching Protocol (ICP). Both academic Proxies and the Transparent cache use SQUID for cache management. SQUID is a high-performance proxy caching server for web clients, supporting FTP, gopher, and HTTP data objects. Unlike traditional caching software, Squid handles all requests in a single, non-blocking, I/O-driven process.

In view of the caching hierarchy of GRNET it is clear that the Transparent cache Log file that we analyze in this paper will consist of 3 categories of data: TCP Hits/misses, ICP hits/misses and Error code responses. The basic objective of this work is to quantify the success of a potential prefetching scheme applied at the Transparent cache using trace based simulations.

The simplified Prefetching architecture (without network equipment such as switches, routers, etc.) that we will study in this paper is presented in Figure 5. The prefetching point is set to be the Transparent cache which is the edge point of GRNET to the Internet. End users may be connected to the Transparent cache directly or through an academic Proxy server. The Transparent cache initiates demand requests to Web servers represented by

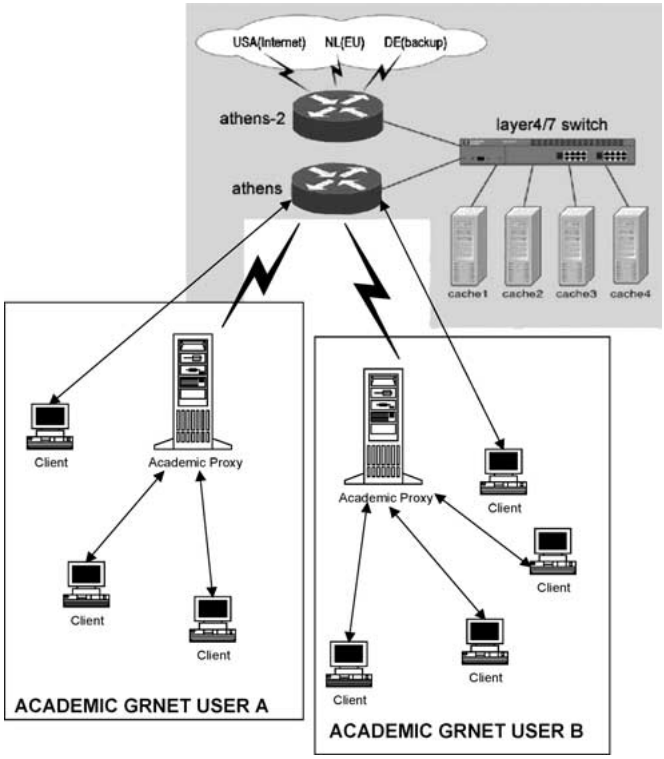

Figure 4. The GRNET topology.

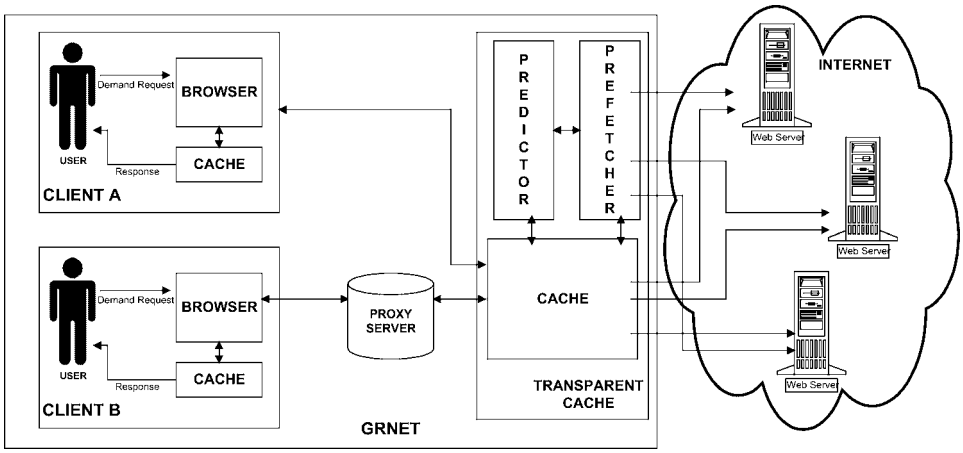

Figure 5. The Prefetching architecture used in this paper.

normal directed lines but also prefetching requests represented by the dotted directed lines between the Transparent cache and the Internet.

5. Trace log characteristics

In this first section of our experiments we provide a clear input on the workload of the Transparent cache, the types of requests that it is asked to fulfill and statistical data of the trace logs that we analyze. The traffic logs that we use consist of web page requests made at the primary caching point over a 7-day period. The Transparent cache receives requests from 300 different clients, both Proxy servers and single users, during that time. As we mentioned earlier in this paper the session timeout value that we selected for our study after examining different timeout values was 10 minutes. The total number of sessions found in the log data for this value of session timeout is 3398. The average number of requests per session is 147.

5.1. Request categories (TCP, ICP, Error)

Here we compare the request types of the Transparent cache. It is useful to make this comparison because its result will be a clear indication of the user categories of the Transparent cache. Figure 6 shows the number of requests logged for each request category.

The number of requests per request category indicates the amount of requests received by each of the different types of clients that the Transparent cache services. The cache receives requests by both "big" clients (ICP requests by Proxy servers) and "small" clients (TCP requests by individual clients). It is obvious that both request categories must be taken into consideration by a prefetching approach at the Transparent cache in order for it to be useful.

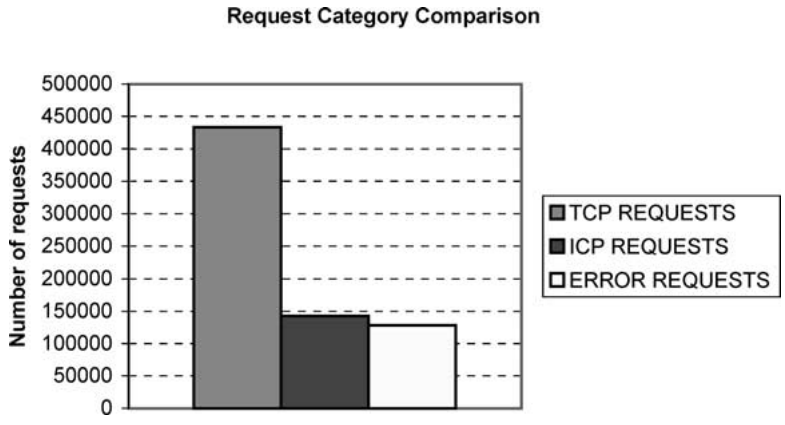

Figure 6. A comparison of request categories in the log files used.

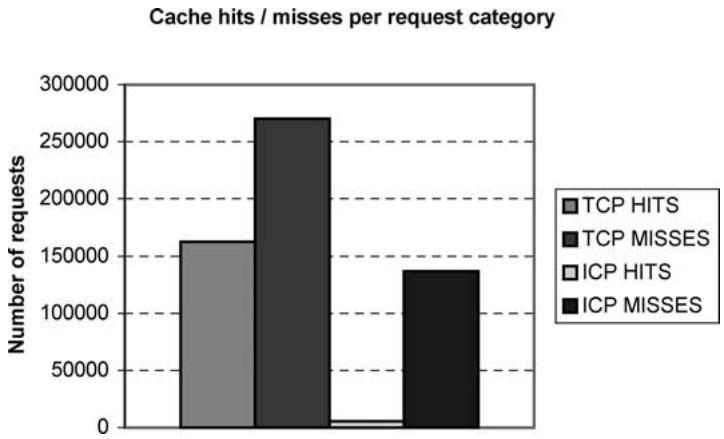

Figure 7. Cache hits and misses per request category without use of prefetching.

5.2. *Cache hits/misses per request category*

The success of a prefetching scheme on a cache includes the reduction of cache misses (both ICP and TCP). In this section we quantify the caches hits and misses per request category before any prefetching scheme is applied. The number of cache hits and misses for each request type is indicated by Figure 7.

Figure 7 shows that the percentage of cache hits for TCP requests before using a prefetching policy is 37.6%. For ICP requests only 3.85% of them are cache hits. This reveals the need for applying prefetching on the highest level of caching.

5.3. *The requested file types*

Figure 8 shows some overall statistics on the trace log files used, concerning the types of files requested.

Image files and HTML files account for the vast majority of requests to the Transparent cache. Figure 8 is an indication of the objects that must be considered in the prefetching policy that we will present in following paragraphs. The successful prefetching of images and HTML files would increase the effectiveness of the Transparent cache.

6. 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.

The prefetching model used in each of the two approaches is built on log data stored at the Transparent cache of GRNET. Every record of the Transparent cache log file consists of the time the request was made, the size of the requested document, the IP address of

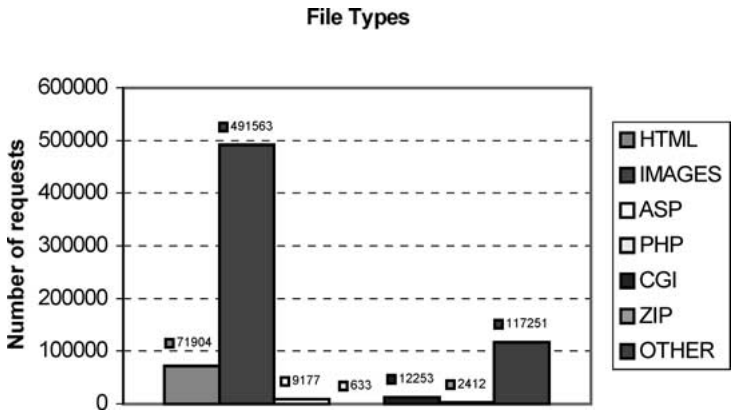

Figure 8. The requested file types.

the client and the requested URL. However, the initial form (classic SQUID Proxy server format) of the log file was not suitable for examination. Therefore, it was preprocessed to organize its content. The filtering steps that were followed are:

1. Requests other than GET and ICP requests (e.g., POST requests) were removed, as we are interested in retrieval latency observed on the client side.
2. Requests for dynamic web pages, such as cgi pages, or for web pages that represent data generated by search engines were not also taken into account, because prefetching them would have no sense since they can possibly change on every request. This is similar to the approach followed in [37,38].
3. Instances of self-referring documents were taken away, as the requested document would already be in the client's cache after the first request.
4. Documents embedded in a base document were taken into account. These documents are automatically requested when the main document is requested. Embedded documents were identified and separated from other requests. We assumed that these documents are pre-sent when the page that contains them is pre-sent.
5. The file taken after performing steps 1–4 on the initial Transparent cache log file was separated into smaller files, each of which represented a session. The access log file consists of a set of sessions. Each session is a sequence of requests by a client, where each request corresponds to a visit to a web page. The new files, which replace the existence of the Transparent cache log file in our model, are indexed by the number of each client and the client's session number. Sessions are extracted from the initial file as follows: *For any two adjacent visits, if the time interval between them is greater than a time threshold, then these visits are considered to belong to two different sessions. The threshold is set to ten minutes.*

The "most popular" approach proposed in this section bases prediction for client's future requests on popularity ranking of previous requests. Popularity ranking can be applied either on behavior patterns of the client only (client-based approach) or on behavior patterns of the general population (overall approach).

As mentioned earlier in this paper, clients connected to the Transparent cache, where prefetching takes place, can be either individual clients or academic Proxy servers. In the first case, the client is a single user. A client-based approach for this type of client performs popularity ranking on this user's passed requests only. In the second case, a LAN Proxy server plays the role of a client for the Transparent cache. A client-based approach in this case relies actually on behavior patterns of more than one single user who are all connected to the LAN Proxy server. However, all these users are treated as a single client by the prefetching algorithm that runs on the Transparent cache, as all requests made by them pass through the LAN proxy server before being forwarded to the Transparent cache. In other words, a general behavior pattern is built for all single users connected to a LAN Proxy, which is then used by the prefetching algorithm that is applied at the Wide Area access point to perform client-based ranking for this proxy client. Actually, in order to carry out client-based popularity ranking on the third level of caching hierarchy, we perform first an overall popularity ranking approach on the second level of caching hierarchy for each LAN Proxy server, as behavior patterns of the general population of the academic Proxy server are used for ranking.

To predict a future request made by a client we first need to build the access profile for this client. The Transparent cache log data is used for that reason. Analysis of log data focuses on the frequency of visits and the sequence of requests. These specify the preferences of users and imply their future behavior. Log data processing includes popularity ranking of requested pages, frequency of content change estimation and page dependencies examination. All these procedures are carried out for every client separately and result in the construction of different popularity lists for each client. These popularity lists are then used by a "most popular" prediction algorithm to compute which pages are most likely to be requested next and by an additional decision algorithm that decides whether prefetching will be applied or not, and how many pages are going to be prefetched, based on bandwidth limitations determined by available resources. If an overall approach is followed, both the prediction algorithm and the decision algorithm use general popularity lists extracted by adding up popularity data from all the separate client-based popularity lists. The whole process of log data analysis for both client-based and overall ranking is shown in Figure 9. All independent procedures are described in the following paragraphs.

6.1. Popularity ranking

Page popularity ranking is a process of log data analysis used to determine the pages that are most likely to be requested next. This process is applied for each client connected to the Transparent cache separately (in case predictions are based on client log data only) and overall (in case predictions are based on log data by all clients).

As mentioned before, client log data in case of a LAN Proxy server may consist of log data from many different users. Therefore, page popularity ranking in this case needs to be carried out first for each user connected to the proxy server. For any page that appears in a user's log data, all instances of it are found and a popularity list is created, on the top of which exists the page that has been requested most by this specific user. Adding up page

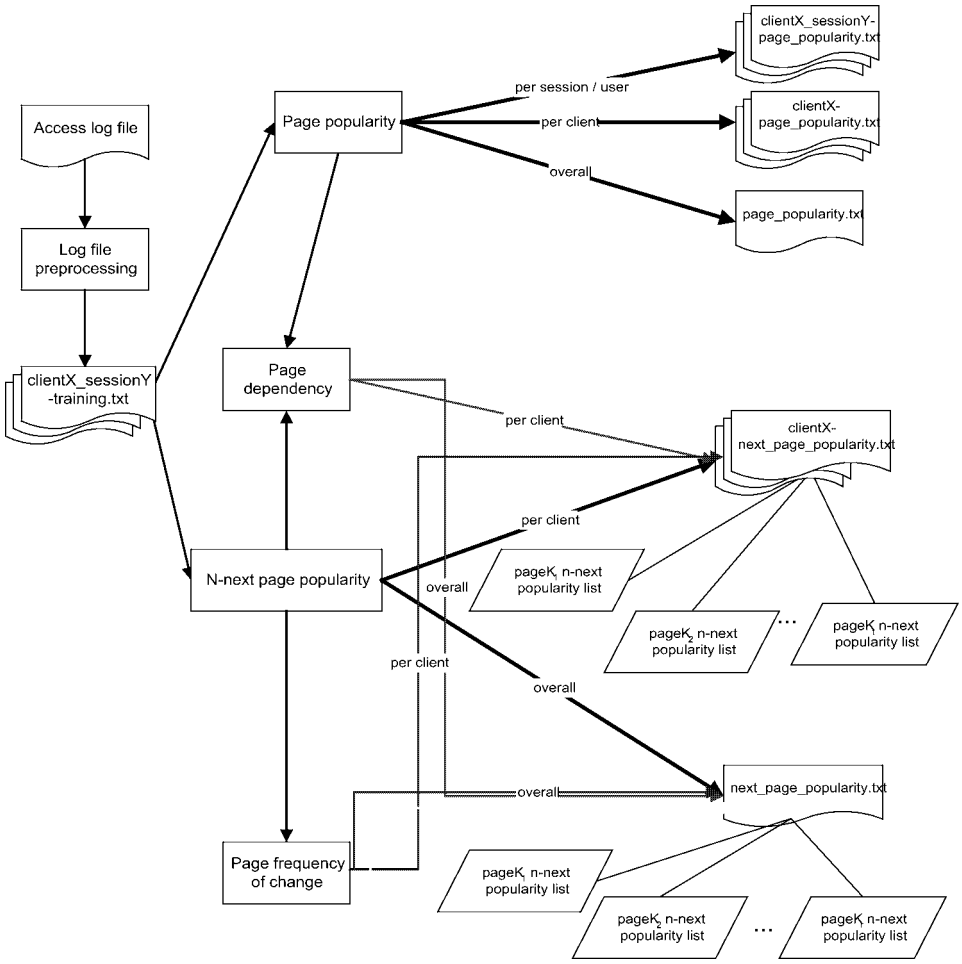

Figure 9. Processes of log data analysis and output files used by the "most popular" prefetching model.

popularity data from all users that are connected to the same academic server, we create a popularity list representative of the general browsing habits of users using this LAN Proxy server (client-based page popularity ranking).

Popularity lists for separate users that are connected to the Transparent cache through a LAN Proxy server are not maintained in memory after performing popularity ranking for all requests directed through the proxy server. The storage of a separate popularity list for each user would increase the space cost of our "most popular" algorithm dramatically as a large table would need to be stored for every user to keep this statistic. The computational cost for updating or searching in all these lists would also be very high. This is the reason why client-based popularity ranking performed at the Transparent cache is not user-specific for each user in the first level of caching hierarchy (client browser) but rather client-specific

for each client in the second level of caching hierarchy (LAN Proxy servers), for the cases that three levels of caching exist. In this way we take advantage of the intermediate caching point of our architecture. If a different approach was used the caching level that consists of LAN Proxy servers would not add anything to the prefetching scheme as prefetching would be performed exclusively between the first and the third level of caching hierarchy. Our approach may lose in accuracy of predictions if users from diverse groups connect to the same LAN Proxy server. However, we assume that the browsing profiles of users connecting to the same academic server are somewhat similar and that such an approach adds more to the computational simplicity of the "most popular" algorithm than affects the personalization of predictions.

For single users connected directly to the Wide Area access point page popularity ranking is first carried out at a session level. A popularity list is created first for pages requested 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 (client(user)-based page popularity ranking).

Client-based page popularity ranking is performed for all clients (both individual and proxy clients) connected to the edge point of GRNET. Then data from all clients is taken into account to perform an additional page ranking on the Transparent cache (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. Any page requested within n accesses after a specified page was requested is considered to be an n-next page of it. In order for a page to be counted as an n-next page of an examined page it also needs to have been requested within the same client session that the examined page has been requested. 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 an n-next page to it.

The value of the lookahead window size needs to be large enough to extend the applicability of the prefetching algorithm and small enough to avoid abusing the system and network resources available. The study of Transparent cache log data in terms of dependency between pages reveals that pages accessed more than 5 accesses after a specified page are not highly related to this page. Therefore, we choose the lookahead window size value to be equal to 5.

The process of n-next popularity ranking is carried out initially for every client separately. For each web page that has been requested by the client, the pages that had been requested within n accesses after it are stored. The instances of any of these pages as n-next of the examined page are calculated and the pages are ranked according to their popularity as n-next of the examined page. Thus, an n-next popularity list is created for every page requested by the client (client-based n-next popularity ranking). Algorithm 1 shows the process for building the n-next popularity prediction model from access patterns of the client only.

Putting the results from all clients together, we build a general n-next popularity list for every page logged, which maps the web behavior of the general population for that page

Algorithm 1. Building the prediction model for the client-based "most popular" approach
Input: Prediction model constructed so far, client's training log data
Output: Updated prediction model
for every request in client's training log data:
  set client's current request as active
  for each client'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

(overall n-next popularity ranking). In this case, a prediction model from access patterns of all clients is being constructed.

6.2. *Frequency of change*

Web content changes regularly. Therefore, the frequency of a page occurrence as n-next of another web page cannot by itself be a secure criterion for prefetching. If a page has a high probability of being requested after a specified page but its content appears to change continuously, its prefetching would have no sense. For that reason, for every page found as n-next of the currently examined page (active page), we check how often its content changes for the times it has been requested within n accesses after the active page. This is done as follows: every time the page is found as n-next of the examined page, its size is compared with the size it had when it was last requested as n-next of the active page. The frequency of change for such a page is defined as the ratio of the times the size of the page is found to have changed to the total number of times the page was requested as n-next of the examined page.

As in the case of n-next popularity ranking, the process of finding a page's frequency of change as n-next of a specified web page is carried out both for every client separately and overall. In the first case, frequency of change is estimated only for those times the page has been requested by the specific client within n accesses after the examined page. In the second case, all occurrences of the page as n-next of the examined page are taken into account. Frequency of change values are kept for every page in the proportional field of the appropriate n-next popularity list of the specified page. This is the client-based n-next popularity list for the first case and the overall n-next popularity list for the case that all occurrences of page are taken into consideration.

6.3. *Page dependency*

Accuracy of prediction for client's next request is affected by the extent of relation between pages. If a page that is a candidate for prefetching, is highly dependent of the currently

displayed page, then the prediction that this would be the client's next request, has a high probability of being correct. Dependency is defined as the ratio of the number of requests for a page as n-next of a specified page (stored in an n-next popularity list), to the total number of requests for the specified page (stored in a page popularity list). In every n-next popularity list that is created, there exists a field at each record including an integer (with value between 0 and 1) that stands for the dependency value of the web page stored as n-next there. To calculate the page dependency value based only on client navigation history, we use data from client popularity lists. Otherwise, general lists are used.

The values of frequency of change and page dependency are both used by the decision algorithm of our "most popular" prefetching scheme to limit the number of prefetching actions being performed, in case of low availability in resources.

6.4. Prediction algorithm

To predict a future request of a client, we use a simple algorithm that is based on n-next popularity data. Suppose a client is currently displaying a page. To predict his next request we rely on the client's request history for the currently displayed page. The pages that have the best chance of being requested after the currently visited page are those that were most frequently visited as n-next of it in the past. Thus, the pages, which are at the top of the n-next popularity list of the currently displayed page, appear to be candidates for prefetching. Prediction can be based either on n-next popularity lists that are created from client log data (client-based prediction) or on lists of the general popularity (overall prediction).

The number of pages that are going to be predicted as candidates for prefetching is specified by the parameter m, which is called prefetching window size. The prediction algorithm suggests the m first web pages of the n-next popularity list of the currently displayed page as the most probable pages to be accessed next. The prefetching window size is a parameter of the prefetching scheme. A large value of m results in many pages being prefetched, which increases the number of successful prefetching actions. However, more bandwidth is required to perform prefetching then, resulting in a considerable increase of network traffic.

6.5. Decision algorithm

The "most popular" prefetching model that is proposed in this paper uses a decision process, which determines whether or not prefetching will be applied and how many pages will be prefetched. In any case, the size of prefetching cannot be greater than the prefetching window size determined by the prediction algorithm.

In a simple form of the decision algorithm, prefetching is decided for any page suggested by the prediction algorithm. We characterize this form of decision policy as an *aggressive prefetching policy*. In this case, we first check if the page that is a candidate for prefetching is already in the cache of the client. If not, the page is pre-sent to the client.

Algorithm 2. Predicting a client's next request given the prediction model, the current (last) request, the dependency threshold and the frequency of change threshold
Input: Prediction model, client's current request, dependency threshold, frequency of change threshold
Output: A set of predicted pages

```
for every request in client's simulation data:
  set client'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
```

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. Furthermore, prefetching of pages whose content seems to change frequently should be avoided, since such a page could be obsolete before actually being viewed. This decision policy is called *strict prefetching policy*. In the case of the strict prefetching policy, the decision algorithm checks the following parameters for every page proposed by the prediction algorithm: its size, its dependency to the current page and its rate of content change and decides, regarding bandwidth limitations, whether prefetching of that page would be advantageous or not. If the size of the page is larger than the average size of all visited pages, then a possible unsuccessful prefetching action performed for this page would affect network traffic dramatically. So prefetching is decided for such a page only if the dependency and the frequency of change values estimated for it satisfy the thresholds used to assure that prefetching of this page is very probable to be successful. The whole prediction process for the case of client-based prediction is shown in Algorithm 2.

7. The Prediction by Partial Matching approach

Prediction by Partial Matching (PPM) is an algorithm that originates in the data compression field. It was first proposed as a data compression scheme but is also suitable as a general purpose prediction algorithm, provided that predictions are based on a number of recent previous events (context modeling). The length of a context is its order. If m previous events are used for prediction then this is an *order-m* model. One of the best features of PPM is that it is online, meaning that it can theoretically be applied even with no training period before it starts predictions. Of course a training period makes the algorithm more efficient.

Algorithm 3. Building prediction model from client' access patterns (training process)
Input: Structure representing prediction model of order *m* constructed so far, client's training log data
Output: Updated prediction model

```
current context [0] = root node of structure
for every request in client's training log data:
   for length m down to 0:
      if request not appearing as child node of current context [length] in structure:
         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 client'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
```

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 a m context trie, representing their occurrence and their sequence. Here 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 [37,38].

Algorithm 3 describes the process of building the PPM prediction model. At the beginning the only available context is the order-0 context. The first request found in client's

training log data becomes child of the root node of structure and at the same time the new order-1 context. For each context occurrences are kept. When the next request is accessed all contexts are updated in turn. If the request is accessed for the first time as next request of the current context then a new node is added in the model as child of the current context. Otherwise the occurrences' field is updated. This process continues until the entire sequence of requests examined is exhausted.

Algorithm 4 describes the prediction process of our PPM approach. A different value of confidence is used for each context of the prediction model constructed by Algorithm 3. Each context produces independent predictions based on the value of confidence used for it to decide whether a page should be prefetched or not. Duplicate pages that are predicted for more than one context are removed.

8. Experimental study

In order to evaluate the performance benefits of the two prefetching schemes introduced in the previous paragraph, we use trace-driven simulation. As mentioned in Section 4 access logs of GRNET Transparent cache are used to drive the simulations. In each execution, simulation is driven on requests of a different group of clients. 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*, *Latency Reduction* and *Network Traffic Increase*.

Prefetching Hit Ratio is the ratio of prefetched pages that the users requested (useful prefetched pages) to all prefetched pages. It represents the accuracy of predictions. If hit ratio is high, then retrieval latency on the client side, in case prefetching has been performed, will be low, as almost all requests are served by the client's cache.

Usefulness of Predictions is the ratio of prefetched pages that the users requested (useful prefetched pages) to all requested pages. It represents the coverage (recall) of predictions.

Latency Reduction is the reduction in user perceived latency due to prefetching. As we saw earlier, prefetching causes the placement of the actual request for some resource at some time t_0, in advance of the time t_1 that the client actually chooses to see the resource. Therefore, the $t_1 - t_0$ part of the total request latency is not perceived by the user. We assume, based on the definition of the "perfect" prefetching system that was presented in the introductory section, that the actual requests are prioritized against the prefetching requests. Thus, prefetching can only take place when no actual requests are being serviced. In other words, the $t_1 - t_0$ time can only be part of the idle time between actual requests. We estimate the idle time by calculating the difference between the estimated end-of-transmission of one request and the start of the next request from the same client in the same session. For each useful prefetched page user perceived latency is reduced by the amount of the idle time used for pre-sending the page between the request for the page and the previous request. The latency reduction is calculated as the percentage of network latency without prefetching being hidden from the user because of prefetching. The increase

of network traffic due to unsuccessful prefetching is taken into account when estimating latency reduction.

Network Traffic Increase is the increase in network traffic due to unsuccessful prefetching. It represents the bandwidth overhead added, when prefetching is employed, to the network traffic of the non-prefetching case. It is obvious that future behavior of users cannot be predicted precisely by their access history. Therefore, some of the prefetched documents will not be actually requested. These documents add undesired network traffic, as they should not have been prefetched in the first place.

Any prefetching policy aims at balancing the conflicting factors of Prefetching Hit Ratio (or Usefulness of Predictions) and Network Traffic Increase when prediction about future requests is made and pre-sending of documents is performed. A strict policy that almost never prefetches any documents will manage to keep traffic increase low, but will not profit much from the application of prefetching. Accordingly, an aggressive policy, which prefetches a large number of web pages every time trying to "catch" user's next request, will definitely have high usefulness of predictions, but to do so it will need great bandwidth cost.

Latency reduction is related to the usefulness of predictions. Having a large number of requests being served by prefetching means that a large number of requests are served with a $t_1 - t_0$ "gain" in response time perceived by the user. Therefore response times are significantly improved. However, as we will see in the following sections, the percentage of latency reduction is always smaller than the percentage of pages served by prefetching. The reason is that the size of useful prefetched pages is most of the times smaller than that of non-prefetched pages. Since prefetching is only performed during idle times between actual requests, there is usually not enough time to prefetch large objects. Additionally, in cases of large idle times between requests, no prefetching or unsuccessful prefetching usually takes place. This happens because a request issued a long time after the previous request was issued is most likely to be unrelated to the previous one. Therefore, it is more difficult to be predicted based on the previous request. Even if a prediction is managed to be made there is a high chance for it to be unsuccessful.

The 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*.

Average Rank is the average rank of prefetch hits (in cases of successful prefetching action) in the set of predicted pages (or in the n-next popularity list of the active request).

"*Most popular*" *prefetch effectiveness* is the ratio of requests that are serviced from prefetched documents to the total number of requests for which prefetching is performed. This value is different from that of Usefulness of Predictions as only requests for which prefetching is applied are taken into account. This metric is used mainly to compare the performance of the client-based and the overall prefetching. Client-based prefetching results in more accurate predictions, because of the use of more accurate log data. On the other hand, overall prefetching has higher usefulness of predictions as it can be performed for more requests, but inadequate log data used for some of these cases causes low accu-

racy of predictions. For this reason, a new performance metric is used in order to make comparison between these two cases of "most popular" prefetching more clear.

In the theoretic analysis of the two prefetching approaches we saw that a set of predicted pages is suggested for prefetching by the prediction process. However, only one of the prefetched pages will serve client's next request, in case the prediction is successful. Presending the rest of prefetching candidates is not necessarily a waste of bandwidth, as they may serve some other future requests. If such a pre-sent document is actually requested in the same session, we assume that the request is served from the client's cache and we do not consider the size of this document when estimating traffic increase. In other words, client's cache in our experiments is considered to hold no more pages than those requested during a client's session. If we take into account the average size of requests in GRNET and the average number of requests in a session, for session timeout value equal to 10 minutes, the average cache size of a client considered by our simulation is found to be 1 MB. This size of cache stands only for a small part of the web client cache that is actually used for requested documents (especially in the case of academic Proxies). However, a small value of cache size enables us to study the actual benefits of prefetching for each of the two approaches proposed. Additionally, Transparent cache log data analysis indicates that the frequency of content change for prefetchable objects within a user session is very low. Less than 1% of the requests were found to change content within a session. Therefore, the results presented in the following paragraphs are not affected by large $t_1 - t_0$ times that cause cached pages to flush out, since documents are considered to be replaced in cache after the current session ends and therefore prefetched again if requested in a different session.

8.1. Most popular approach evaluation

As presented earlier in this paper, prediction of future requests can be based either on client-specific web log data (client-based prediction) or on data from all clients (overall prediction). Moreover, different decision policies can be applied, depending on the availability of resources. When availability of bandwidth is high, prefetching is applied for every page suggested by the prediction algorithm (aggressive policy). Otherwise, a more restricted policy is followed, which takes into account the size of the document, the dependency value and the frequency of change (strict policy).

The prefetching window size, m, is also a parameter that varies in "most popular" approach experiments. For every request m documents are predicted as prefetching candidates. In this section, we present results taken for $m = 3$ and $m = 5$.

The experimental scenarios for the evaluation of the "most popular" prefetching scheme's performance, as these derive from the alternative values of the parameters mentioned above, are:

(i) aggressive policy, $m = 5$, client-based prediction;
(ii) aggressive policy, $m = 5$, overall prediction;
(iii) aggressive policy, $m = 3$, client-based prediction;
(iv) aggressive policy, $m = 3$, overall prediction;

Algorithm 5. Most Popular Algorithm (Performance Evaluation)
for each request in set of predicted pages concerning prediction of client's next request:
  if request ! = *client's actual next request*
    prefetch miss ++
    traffic overhead + = *request size*
  else if request == *client's actual next request*
    prefetch hit ++
    prefetch hit rank = *rank of request in set of predicted pages*

(v) strict policy, $m = 5$, client-based prediction;
(vi) strict policy, $m = 5$, overall prediction.

Algorithm 5 shows the evaluation procedure followed for the case of the "most popular" model.

8.1.1. Results The results obtained by simulation experiments show the benefits and the costs of our "most popular" prefetching approach. Here we first present and compare the performance results of our method for two different values of the prefetching window size ($m = 3$ and $m = 5$), in the case of the simple, aggressive decision algorithm. Then, we investigate how the use of a strict decision process affects the performance of the prefetching scheme. Furthermore, we check the results of prefetching for both cases of client-based prediction and overall prediction.

If client-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% for this case, user perceived latency reduces by 15.71% and traffic increase per request is found to be 6%. Moreover, the average rank of successfully prefetched documents in the n-next popularity list is 2. Finally, the prefetch effectiveness for this scenario (scenario (i)) is 52%. This means that almost one out of two requests, for which prefetching is performed, is serviced by prefetched documents that are already in the cache.

If we use n-next popularity data obtained from the general population in our predictions, instead of client 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 we try to limit network traffic increase to 8%, then prefetch effectiveness will be equal to 50%. It is clear that for the same traffic increase the performance results of client-based prediction are better than those of overall prediction. This is because client data implies more accurately the client's future web behavior than overall data.

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 client-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, so traffic increase for this case is only 4%. 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. As a result, latency reduction is less significant (14.5%). The average rank of successfully prefetched documents for this experimental scenario (scenario (iii)) is 1.

The results of overall prediction for m equal to 3 are similar to those taken for m equal to 5. Prefetch effectiveness is a bit higher (44%) compared to the value found for client-based prediction (42%), but traffic increase reaches 12%. For traffic increase equal to 5%, prefetch effectiveness cannot be higher than 41%.

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 experiencing at the same time high performance, 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. So we use a dependency threshold to decide if we are going to prefetch or not a prefetching candidate. Furthermore, we need to avoid prefetching for pages that change regularly. Therefore, a threshold is used for frequency of change. Next, we present performance results for using a strict decision process (experimental scenarios (v) and (vi)) with dependency and frequency of change thresholds. 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 client-based prediction and a prefetching window size equal to 5, prefetch effectiveness is almost as high as in the case of the aggressive policy (51%), while traffic increase is limited to 4%. Similarly, for overall prediction, prefetch effectiveness is equal to 53% and traffic increase is not greater than 13%. 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%) compared to the aggressive approach (48%), for the same prefetching window size, with little cost in usefulness of predictions. Table 1 shows results taken for all three cases of client-based prediction. Figure 10 compares performance results of client-based and overall prefetching for the different prefetching policies and the different values of prefetching window size.

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

	Hit ratio	Usefulness of predictions	Prefetch effectiveness	Latency reduction	Network traffic increase	Average rank
Aggressive policy, $m = 5$	48%	27.5%	52%	15.71%	6%	2
Aggressive policy, $m = 3$	58%	25%	42%	14.5%	4%	1
Strict policy, $m = 5$	51%	27%	51%	15.68%	4%	2

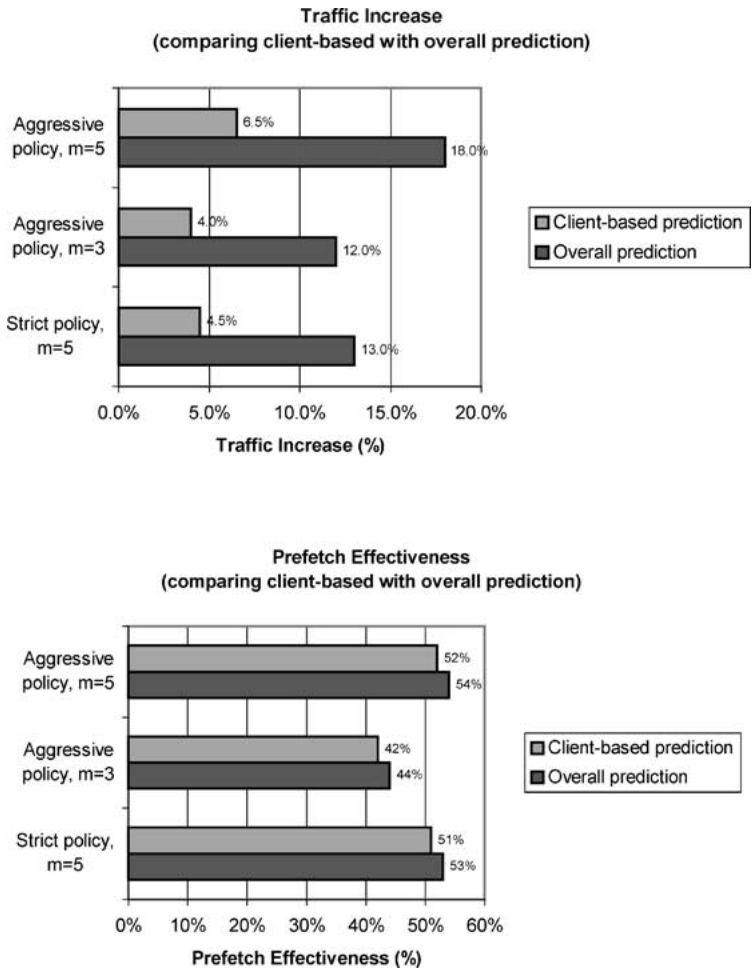

Figure 10. Comparison of the client-based and overall prediction scenarios for all policies (the graphs should be read in pairs of bar charts).

Algorithm 6. PPM algorithm (performance evaluation)
for each request in set of predicted pages concerning prediction of client's next request:
  if request ! = *client's actual next request*
    prefetch miss ++
    traffic overhead + = *request size*
  else if request == *client's actual next request*
    prefetch hit ++

8.2. Prediction by Partial Matching approach evaluation

In the evaluation process of the Prediction by Partial Matching prefetching scheme, a prediction model that stores multiple context orders in the same structure is being used. This approach offers the opportunity to combine predictions from different orders. The value of the maximum order for the model used in our experiments is 4. Furthermore, the maximum number of predictions allowed is 5, to match the case of the "most popular" approach with prefetching window size equal to 5.

The additional evaluation parameters that we use in our simulation [37,38] are *Confidence* and *Previous Requests*.

Confidence is defined as the number of occurrences of the current node (Web object) divided by the number of occurrences of the parent node.

Previous requests is defined as the number of requests that a particular client has issued to a server so far.

Confidence and *previous requests* parameters vary in our experimental study. The effect of these parameters on the performance of prefetching is investigated later in this paper.

The evaluation process followed for the case of the PPM model is shown in Algorithm 6.

8.2.1. Results In this paragraph we examine the influence of PPM parameters on the performance of the method.

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 11 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 offer 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 12 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 which use constant confidence from 0.6 up

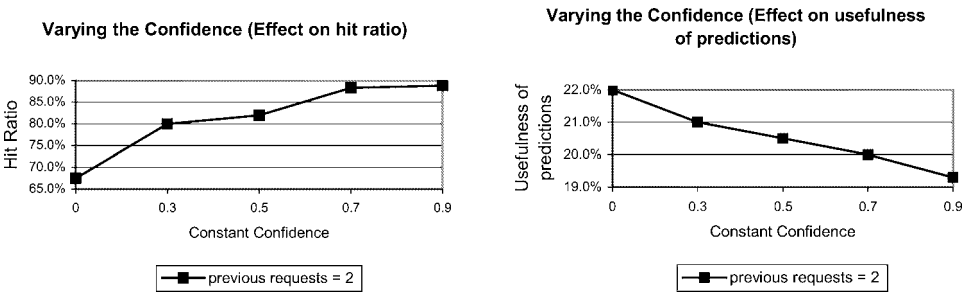

Figure 11. Varying confidence – effect on performance of PPM approach.

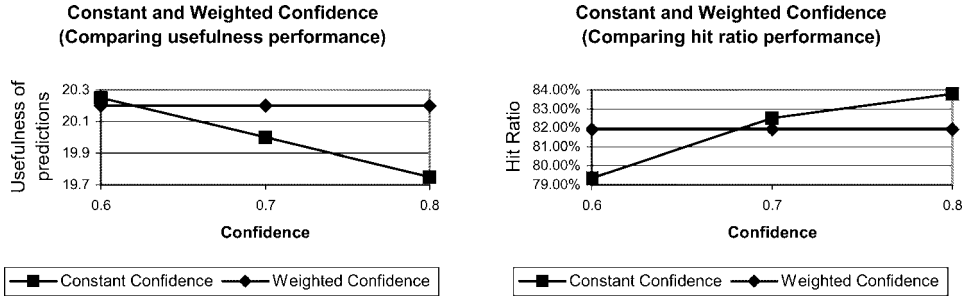

Figure 12. Comparing constant and weighted confidence.

to 0.8. Diagrams show that weighted confidence performs above the average of the three constant-confidence algorithms.

As shown in Figure 13 increase of the previous requests parameter results in decrease of the number of prefetched pages (represented by usefulness of predictions metric) and in less latency reduction. However, predictions become more accurate since the prefetcher makes predictions only for clients that do not leave after the first requests.

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 resulting in an insignificant reduction of user perceived latency. Two such cases are shown in Table 2. The 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%.

8.3. Comparing "Most popular" and PPM performance

For both prefetching approaches studied in this paper an aggressive prefetching policy, that maximizes the usefulness of predictions metric but at the same time minimizes the hit ratio

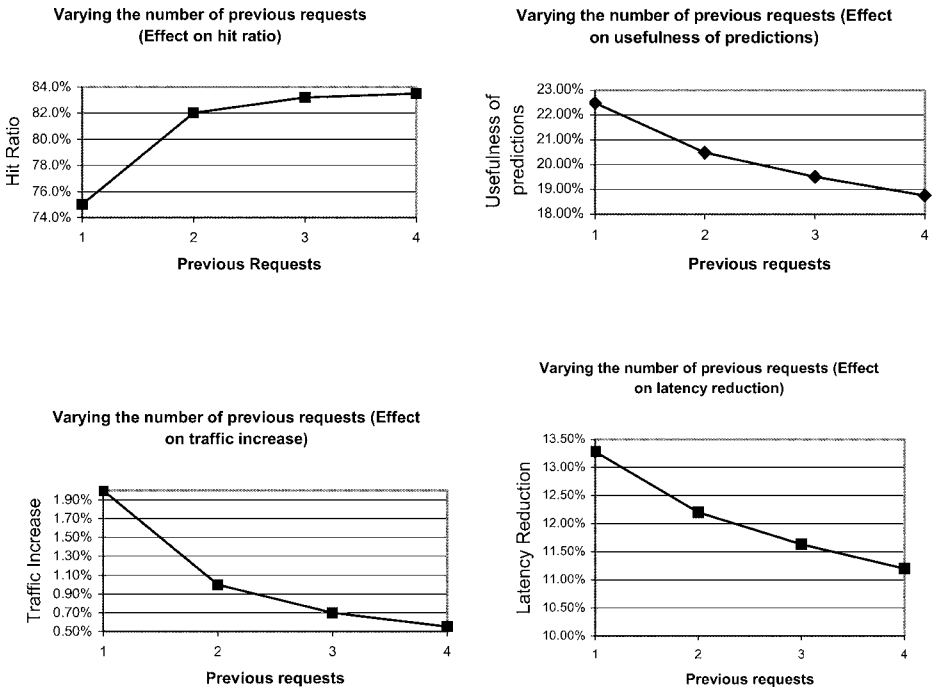

Figure 13. Varying previous requests number – effect on performance of PPM approach.

Table 2. Efficiency of PPM approach.

	Hit ratio	Usefulness of predictions	Latency reduction	Network traffic increase
Aggressive policy, previous requests = 1, confidence = 0.2–0.3	73%	23%	13.6%	2%
Strict policy, previous requests = 4, confidence = 0.7–0.9	85%	18.25%	11%	<0.5%

metric adding traffic to network, and a strict policy, that minimizes predictions made, were introduced. In this paragraph, we compare results taken for the "Most popular" and the PPM algorithms for both prefetching policies. We use performance results of the client-based "Most popular" approach (shown in Table 1) and of the PPM approach presented in previous paragraph (Table 2). The two approaches are compared for Hit Ratio, Usefulness of Predictions, Latency Reduction and Traffic Increase metrics (Figure 14). We choose the client-based prediction approach of the "Most popular" scheme to compare with the PPM algorithm, as PPM is a client-based prefetching model, as well. Furthermore, client-based prefetching for the "most popular" approach appears to have better overall performance results than prefetching based on the general population (as shown in Section 8.1.1).

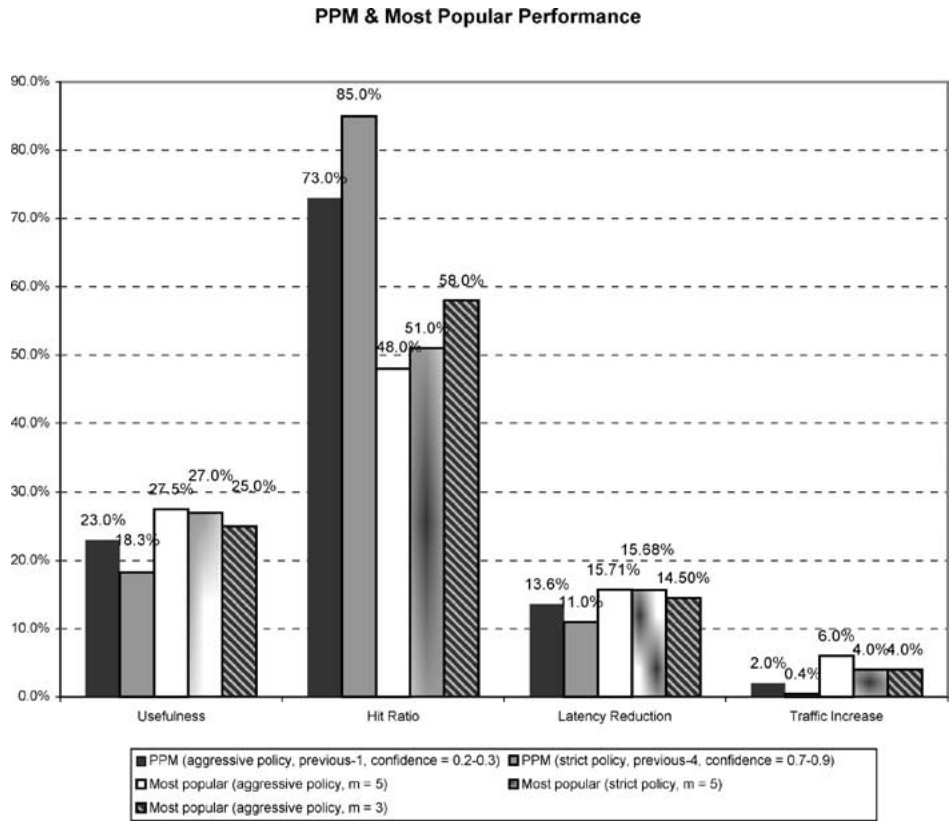

Figure 14. "Most popular" and PPM performance for strict and aggressive prefetching policy.

Figure 14 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 and greater reduction of user perceived latency. 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.

To have a clearer picture of the 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. Figure 15 shows results taken for comparing these proportional cases of "Most popular" and PPM algorithms.

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

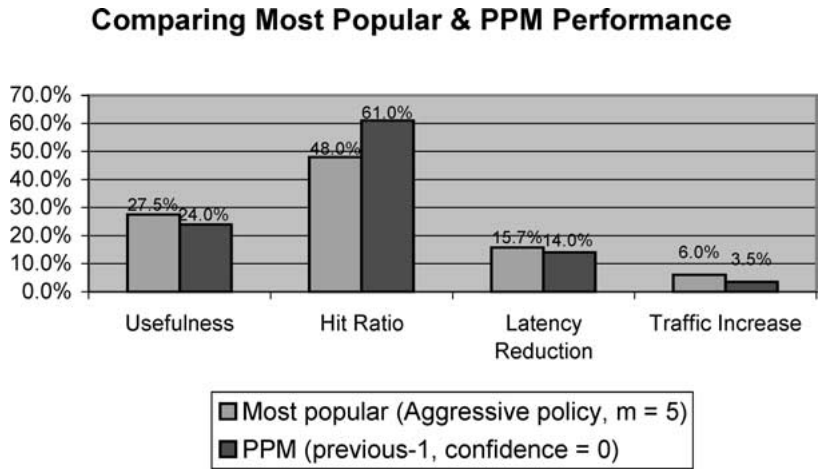

Figure 15. Comparing "Most popular" and PPM performance (case of aggressive "most popular" policy).

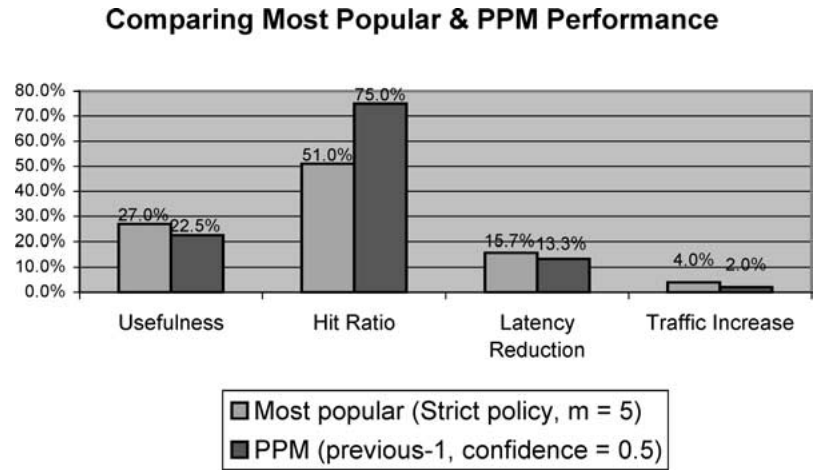

Figure 16. Comparing "Most popular" and PPM performance (case of strict "most popular" policy).

to 0.5. These values of PPM parameters are selected because the strict "Most popular" case uses one previous request in predictions (the last request) and the threshold used by the decision algorithm is also equal to 0.5. Figure 16 shows results taken for comparing these proportional cases of "Most popular" and PPM algorithms.

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 accuracy of predictions can reach 58% (case of aggressive, client-based policy with prefetching window size equal to 3) and latency on the client side can be reduced by 14.5% with an

insignificant for a WAN increase of network traffic equal to 4%. If a more complex algorithm is used, which is a variation of the 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%. Furthermore, cache hits appear to be at least 2 times more in the first case of the "most popular" algorithm and at least 3 times more in the case of the PPM approach, compared to the non-prefetching results presented in Section 5. 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. Moreover, results presented in this section are based on the consideration that a small client cache, which holds only requests of the current session, is used. Benefits of prefetching would be more significant in case of a larger cache size as many of the requested pages considered to have been prefetched more than one times by our simulation would have actually been served automatically by the client's cache after the first time they were prefetched. As a result, either traffic increase would be less in that case or for the same network bandwidth used, additional pages could have been prefetched increasing the number of useful prefetching actions and therefore the cache hit ratio since more requests would be served by the client's cache. Furthermore, the reduction of the user perceived latency would be greater in this case as more requests would be placed in advance hiding $t_1 - t_0$ time from the latency experienced by the user.

9. Future work

As stated in the opening sections of this paper, prefetching is a technique that involves many trade-offs, most important of which is the "resource vs. prediction precision" trade-off. During the preparation of this work many issues arose, that were not foreseen. Some of these issues remain unanswered in the context of this work. Some of them are not even directly related to prefetching, but emerged from the study of related problems and techniques.

In this work we did not study the prefetching of dynamically constructed resources such as search engine results or parameterized pages. The study of whether dynamic content may be included in prefetching is very "attractive". This study must include the Web resource frequency of change problem in order to provide adequate results. Another important open issue is the creation of an algorithm that would prioritize ICP requests to the Transparent cache over direct TCP requests. This idea is based on the observation that ICP requests originate from Proxy servers and TCP requests originate from single users. The prioritization of ICP requests in prefetching intuitively means that the resulting prefetched resource could potentially be useful to more than one clients since it would reside on a proxy server. We also intend to study the usefulness of prefetched objects in the case of ICP requests. If we combine the analysis of Transparent log files with the analysis of its client Proxy server log files, recording the same period of time, we will surely be able to find the positive effects of reuse of the prefetched objects at the Transparent cache.

An issue that came up during the extraction of request categories at the Transparent cache (Figure 6) is the effect of error propagation to Network latency inside a WAN. The Error requests that were measured are almost equal to ICP requests. All these requests were propagated to the Transparent cache, utilizing network resources all the way from clients to the edge of the WAN. It would be useful to be able to contain error requests close to clients in a WAN in order for them not to use network resources all the way to the edge of the WAN. An error request/response procedure may not be important to a high bandwidth link but it could use valuable resources over a low-bandwidth link.

10. Conclusions

In this paper we initially presented a methodology for studying prefetching on any Web system. We have shown that prefetching is highly dependent on the architecture that prefetching is intended to serve. The parameters that influence prefetching have been pointed out and elaborated on. We show that extensive study of a system must precede the design and implementation of prefetching, since the parameters that play a role in prefetching are closely related to the system itself.

Prefetching can be highly beneficial to the reduction of response times on the Web. It is not a Web transfer optimization technique aiming to reduce Network traffic. It must be viewed as a latency "masking" technique and can be considered harmful to network resources if not applied correctly. In this paper we argue that prefetching can be more efficient if it is applied at the edge network connection of a WAN. This approach can be more efficient than applying client initiated prefetching because of the more efficient use of available bandwidth and because prefetching at this point may be useful to many clients.

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, an "*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 Web latency 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* 28(2), 2000, 50–63.

[2] A. Bestavros, "Using speculation to reduce server load and service time on the WWW," in *Proc. of the 4th ACM International Conference on Information and Knowledge Management*, Baltimore, MD, 1995, pp. 403–410.

[3] A. Bestavros and C. Cunha, "Server-initiated document dissemination for the WWW," *IEEE Data Engineering Bulletin* 19, 1996, 3–11.

[4] B. Brewington and G. Cybenko, "Keeping up with the changing Web," *IEEE Computer* 33(5), 2000, 52–58.

[5] X. Chen and X. Zhang, "Coordinated data prefetching by utilizing reference information at both proxy and Web servers," *ACM SIGMETRICS Performance Evaluation Review* 29(2), 2001, 32–38.
[6] X. Chen and 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, BC, Canada, 2002, pp. 296–304.
[7] G. Cleary, W. J. Teahan, and I. H. Witten, "Unbounded length contexts for PPM," *The Computer Journal*, 40(2/3), 1997, 67.
[8] 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.
[9] E. Cohen, B. Krishnamurthy, and J. Rexford, "Improving end-to-end performance of the Web using server volumes and proxy filters," in *Proc. of SIGCOMM'98*, Vancouver, BC, 1998, pp. 241–253.
[10] E. Cohen, B. Krishnamurthy, and J. Rexford, "Efficient algorithms for predicting requests to web servers," in *Proc. of INFOCOM'99*, Vancouver, British Columbia, Canada, 1999, pp. 241–253.
[11] M. Crovella and P. Barford, "The network effects of pre-fetching," in *Proc. of IEEE INFOCOM 1998*, pp. 1232–1240.
[12] C. R. Cunha and C. F. B. Jaccoud, "Determining WWW user's next access and its application to prefetching," in *Proc. of the Second IEEE Symposium on Computers and Communications*, Alexandria, Egypt, 1997, pp. 6–11.
[13] B. D. Davison, "Web Traffic logs: An imperfect resource for evaluation," in *Proc. of the Ninth Annual Conference of the Internet Society (INET'99)*, San Jose, 1999.
[14] 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.
[15] B. D. Davison, "Predicting Web actions from HTML content," in *Proc. of the thirteenth ACM Conference on Hypertext and Hypermedia (HT'02)*, College Park, MD, 2002, pp. 159–168.
[16] J. Dean and M. R. Henzinger, "Finding related pages in the World Wide Web," in *Proc. of WWW-8, the Eighth International World Wide Web Conference*, Toronto, Canada, 1999, pp. 389–401.
[17] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. Mogul, "Rate of change and other metrics: A live study of the World Wide Web," in *Proc. of the USENIX Symp. on Internet Technologies and Systems*, Monterey, CA, 1997, pp. 147–158.
[18] D. Duchamp, "Prefetching hyperlinks," in *Proc. of the 2nd USENIX Symposium on Internet Technologies and Systems (USITS'99)*, Boulder, CO, 1999.
[19] L. Fan, P. Cao, and Q. Jacobson, "Web prefetching between low-bandwidth clients and proxies: Potential and performance," in *Proc. of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'99)*, Atlanta, GA, 1999, pp. 178–187.
[20] 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, p. 103.
[21] E. Gelenbe and Q. Zhu, "Adaptive control of pre-fetching," *Performance Evaluation* 46, 2001, 177–192.
[22] J. Griffioen and R. Appleton, "Reducing file system latency using a predictive approach," in *Proc. of the 1994 USENIX Summer*, Boston, MA, 1994, pp. 197–207.
[23] GRNET, `http://www.grnet.gr/`
[24] J. Gwetzman and M. Seltzer, "The case for geographical pushing-caching," in *Proc. of HotOS Conference*, 1994.
[25] D. T. Hoang, P. M. Long, and J. S. Vitter, "Dictionary selection using partial matching," *Information Sciences* 119(1–2) 1999, 57–72.
[26] 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.
[27] A. 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, TX, 2000, pp. 63–69.
[28] J. I. Khan and Q. Tao, "Partial prefetch for faster surfing in composite hypermedia," in *Proc. of USITS 2001*, San Fransisco, CA, 2001.

[29] R. Kokku, P. Yalagandula, A. Venkataramani, and M. Dahlin, "A non-interfering deployable web prefetching system," Technical Report TR-02-51, Computer Sciences, Austin, UT, 2002.

[30] T. M. Kroeger, D. D. E. Long, and J. C. Mogul, "Exploring the bounds of web latency reduction from caching and pre-fetching," in *Proc. of the USENIX Symposium on Internet Technologies and Systems (USITS)*, Monterey, CA, 1997, pp. 13–22.

[31] T. S. Loon and V. Bharghavan, "Alleviating the latency and bandwidth problems in WWW browsing," in *Proc. of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, Monterey, CA, 1997, pp. 219–230.

[32] E. P. Markatos and C. E. Chronaki, "A top-10 approach to pre-fetching the Web," in *Proc. of INET'98 (The Internet Summit)*, Geneva, Switzerland, 1998.

[33] D. Menasce, V. Almeida, R. Fonseca, and M. Mendes, "A methodology for workload characterization of e-commerce sites," in *Proc. of ACM Conference on Electronic Commerce (EC-99)*, Denver, CO, 1999, pp. 119–128.

[34] J. C. Mogul, "Hinted caching in the Web," in *Proc. of the Seventh ACM SIGOPS European Workshop*, Connemara, Ireland, 1996, pp. 129–140.

[35] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos, "Effective prediction of Web-user accesses: A data mining approach," in *Proc. of the Workshop WEBKDD*, San Francisco, CA, 2001.

[36] V. Padmanabhan and J. Mogul, "Using predictive prefetching to improve World Wide Web latency," *Computer Communication Review* 26(3), 1996, 22–36.

[37] T. Palpanas, "Web prefetching using partial match prediction," M.Sc. Thesis at the Computer Science department of the University of Toronto, Canada, 1998.

[38] T. Palpanas and A. Mendelzon, "Web prefetching using partial match prediction," in *Proc. of the Web Caching Workshop*, San Diego, CA, 1999.

[39] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, "Informed prefetching and caching," in *Proc. of the Fifteenth ACM Symposium on Operating Systems Principles*, Copper Mountain, CO, 1995, pp. 79–95.

[40] J. Pitkow and P. Pirolli, "Mining longest repeating subsequences to predict World Wide Web surfing," in *Proc. of the Second USENIX Symposium on Internet Technologies and Systems*, Boulder, CO, 1999.

[41] D. Rosu, A. Iyengar, and D. Dias, "Hint-based acceleration of Web proxy caches," in *Proc. of the 19th IEEE International Performance, Computing, and Communications Conference (IPCCC 2000)*, Phoenix, AZ, 2000.

[42] V. Soloviev, "Prefetching in segmented disk cache for multi-disk systems," in *Proc. of the ACM-IEEE Annual Workshop IOPADS*, Philadelphia, PA, 1996, pp. 69–82.

[43] Z. Su, Q. Yang, and H. J. Zhang, "A prediction system for multimedia pre-fetching in Internet," in *Proc. of the ACM Multimedia Conference 2000*, ACM, Los Angeles, CA, 2000.

[44] W. J. Teahan, "Probability estimation for PPM," in *Proc. of the N.Z. Comp. Science Research Students' Conference*, Univ. of Waikato, Hamilton, New Zealand, 1995.

[45] J. Wang, "A survey of Web caching schemes for the Internet," *ACM Computer Communication Review* 29(5), 1999, 36–46.

[46] Z. Wang and J. Crowcroft, "Prefetching in World Wide Web," in *Proceedings of the IEEE Global Internet 96*, London, 1996, pp. 28–32.

[47] Wcol Group, "WWW collector: the prefetching proxy server for WWW," `http://shika.aist-nara.ac.jp/products/wcol/wcol.html`

[48] Q. Yang, H. H. Zhang, and I. T. Y. Li, "Mining Web logs for prediction models in WWW caching and prefetching," in *Proc. of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001, pp. 473–478.

[49] I. Zukerman, D. W. Albrecht, and A. E. Nicholson, "Predicting users' requests on the WWW," in *Proc. of the 7th International Conference on User Modeling*, Banff, Canada, 1999, pp. 275–284.