CACHEABILITY STUDY FOR WEB CONTENT DELIVERY

ZHANG LUWEI

NATIONAL UNIVERSITY OF SINGAPORE

2003

CACHEABILITY STUDY FOR WEB CONTENT DELIVERY

ZHANG LUWEI

(B.Eng & B.Mgt, JNU)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2003

Name:	Zhang Luwei
Degree:	M.Sc.
Dept:	Computer Science
Thesis Title:	Cacheability Study for Web Content Delivery

Abstract

In this thesis, our main objective is to assist forward proxies to provide better content reusability and caching, as well as to enable reverse proxies to perform content delivery optimization. In both cases, it is hoped that the latency of web object retrieval can be improved through better reuse of content and the demand for network bandwidth can be reduced. We achieve this objective through a deeper understanding of the attributes for delivery. We analyze how objects' content settings affect the effectiveness of their cacheability from the perspectives of both the caching proxy and the origin server. We also propose a solution, called the TTL (Time-to-Live) Adaptation, to help origin servers to enhance the correctness of their content settings through the effective prediction of objects' TTL periods with respect to time. From the performance evaluation of our TTL adaptation, we show that our solution can effectively improve objects' cacheability, thus resulting in more efficient content delivery.

Keywords:

Proxy Servers, Effective Web Caching, Content Delivery Optimization, Time to Live (TTL), TTL Adaptation

Acknowledgement

In the entire pursuit of my Master degree, I have benefited greatly from my supervisor, Dr Chi Chi Hung, for his guidance and invaluable support. His sharp observations and creative thinking always provide me precious advice and ensure that I am on the right track in my research. I am grateful for his patience, friendliness and encouragement.

I sincerely thank Wang Hong Guang, for offering me lots of necessary assistance in both research inspiration on how to write this thesis.

I am grateful to Henry Novianus, Palit, whose enthusiasm in research has inspired me in many ways. He is always ready to help me, especially in the technical aspects of my research.

I would also like to thank Yuan Junli, who enlightened me whenever I encountered any problems in my research.

Special thanks also to my dear husband, Ng Wee Ngee, for giving me tremendous support and brightening my life constantly.

Finally, I would like to express my sincere gratitude to my loving and encouraging family.

Table of Contents

List of Tables

List of Figures

Summary

In this thesis, our objectives are to enable forward proxies to provide effective caching and better bandwidth utilization, as well as to enable reverse proxies to perform content delivery optimization for the purpose of improving the latency of web object retrieval. We achieve this objective through a deeper understanding of their attributes for delivery. We analyze how objects' content settings affect the effectiveness of their cacheability from the perspectives of both the caching proxy and the origin server. We also propose a solution, called the TTL (Time-to-Live) Adaptation, to help origin servers to enhance the correctness of their content settings through the effective prediction of objects' TTL periods with respect to time. From the performance evaluation of our TTL adaptation, we show that our solution can effectively improve objects' cacheability, thus resulting in more efficient content delivery.

We analyze the cacheability effectiveness of objects based on their content modification traces and delivery attributes. We further model all the factors affecting the object's cacheability as numeric values in order to provide a quantitative measurement and comparison. To ascertain the usefulness of these models, corresponding content monitoring and tracing experiments are conducted. These experiments illustrate the usefulness of our models in adjusting the policy of caching proxies, the design strategy of origin servers, and stimulate new directions for research in web caching.

Based on the monitoring and tracing experiments, we found that most objects' cacheability could be improved by proper settings of attributes related to content delivery (especially in the predicted time-to-live (TTL) parameter). Currently, Squid, an open source system for research, uses a heuristic policy to predict the TTL of accessed objects.

However, Squid generates a lot of stale objects because its heuristic algorithm simply relies on the object’s *Last-Modified* header field instead of predicting proper TTL based on the object’s change behavior. Thus, we proposed our TTL adaptation algorithm to aid origin servers in adjusting objects’ future TTLs with respect to time. Our algorithm is based on the Correlation Pattern Recognition Model to monitor and predict more accurate TTL for an object.

To demonstrate the potentials of our algorithm in providing accurate TTL adjustment, we present the result from accurate TTL monitoring and tracing of real objects on Internet. It shows the following benefits in terms of bandwidth requirement, content reusability and retrieval accuracy in sending the most updated content to clients. Firstly, it reduces a lot of unnecessary bandwidth usage, network traffic and server workload when compared to the original content server’s conservative directives and Squid's TTL estimation using its heuristic algorithm. Secondly, it provides more accurate TTL prediction through the adjustment of objects’ individual change behavior. This minimizes the possibility of stale objects’ generation when compared to the rough settings of origin servers and Squid’s unitary heuristic algorithm. As a whole, our TTL adaptation algorithm significantly improves the prediction correctness of an object’s TTL and this directly benefits web caching.

Chapter 1 Introduction

1.1 Background and Motivation

As the World Wide Web continues to grow in popularity, Internet has become one of the most important data dissemination mechanisms for a wide range of applications. In particular, web content, which is composed of basic components known as web objects (such as html file, image objects, …, etc.) is an important channel for worldwide communication between content provider and its potential clients. However, web clients want the retrieved content to be the most up-to-date and, at the same time, with lesser user-perceived latency and bandwidth usage. Therefore, optimizing web content delivery though maximum, accurate content reuse, is an important issue in reducing the user-perceived latency, while maintaining the attractiveness of the web content. (Note that since this thesis focuses on the discussion of web objects, the rest of the thesis might often refer web objects as objects, for simplicity reason.)

The control points along a typical network path are origin servers (where the desired web content is located), intermediate proxy servers, and clients' computer systems. Optimization can either be in the form of optimizing the retrieval of objects from the origin server, or be in the form of intermediate caching proxy. Caching proxy is capable of maintaining local copy of responses received in the past, thus reducing the waiting time of subsequent requests for these objects. However, due to the connectionless of the web, cached local copy of the data might be outdated. Hence, it is the challenge to content providers to design their delivery services such that both the freshness of the web content and lower user-perceived latency can be achieved. This is exactly what efficient content delivery service would like to target.

Improving the service of web content delivery can be classified into two situations:

- *For the first time delivery of web content to clients, or when cached web content in proxy servers has become stale.*

  The requested objects will have to be retrieved directly from the origin servers. Content design has major impact on the latency of this first time retrieval period. Multimedia content and frequent content updating result in more attractive web content. This is translated to embedded object retrieval and dynamically generated content in content design.

  Cumbersome multimedia is the main reason for the slowdown in content transfer. Dynamically generated content adds extra workload to origin servers as well as increases network traffic. It forces every request from clients to be delivered from origin servers. Typically research topics for faster transfer of the required embedded objects from origin servers include web data compression, parallel retrieval of objects in the same web page, and the bundling of embedded objects in the same web page into one single object for transfer [1].

- *Subsequent requests for the same object.*

  Reusability of objects in a forward caching proxy that stores them during their first time request can efficiently reduce user-perceived latency, server workload and redundant network traffic. It is because the distance of content transfer in the network can be shortened significantly. This area of work is called web caching. Substantial research efforts in this area are currently ongoing [2], and large number of papers have shown significant improvement in web performance through the caching of web objects [3,4,5,6,7,8]. Research also shows that 75% of web content can be cached, thus further maximizing its reusability potentials. Web caching has generally been agreed to play a major role in speeding up content delivery. Object's cacheability determines its reusability, which is defined by whether it is feasible to be stored in cache.

There are a lot of potentials in improving web content delivery through data reuse rather than just relying on the reduction of web multimedia content for the first time retrieval. This is an important task for caching proxy. Placing such proxy servers to cache data in front of LANs can reduce the access latency of end-user and lessen the workload of origin servers and network. Thus, bandwidth demand and latency bottlenecks are shifted, from narrow link between end-users (clients) and content providers (origin servers), to being between proxy caches and content providers [9]. With forward caching proxy, this can greatly reduce clients' waiting time for content downloading, through data reuse. This will attract potential clients when competing with others in the same field.

Despite the success of current research in improving the transfer speed of web content, their focuses are more on areas such as caching proxy architecture [10][11][12][13], replacement policy, and consistency problem of data inside [14][15][16][17][18]. Although there are research efforts that try to investigate the basic question of object cacheability – how cacheable are the requested objects, they are more towards the statistical analysis rather than to understand the reasons behind the observations. Not much work is found on delving into an object's attributes and understanding the interacting effects that will optimize their positive influence on the object's reusability and contribute to the optimization of web content delivery. Hence, in-depth understanding of an object's attributes in terms of how each affects object reusability, and quantifying each effect using a mathematical model into practical measurement, will directly benefit caching proxies and origin servers.

1.1.1 Benefits of cacheability quantification to caching proxy

From the view of a caching proxy, having a measurement that can quantify the effect of object's attributes on reusability can provide a more accurate estimate on the effectiveness of caching a web object. This can help to fine-tune the management of caching proxy, such as cache replacement policy, so as to optimize cache performance. Furthermore, web information changes rapidly and outdated information might be retrieved to clients if an object that is frequently updated is cached. Optimizing cache performance using a good cache policy is a key effort to minimize traffic and server workload, and at the same time, provide an acceptable service level to the users.

Therefore, quantitative model for object's cacheability is required which can reflect individual factors affecting: (1) whether an object can be cached, and (2) how effective the caching of this object is. This measurement should also be able to distinguish the effectiveness of caching different objects, so as the replacement policy can pick the best objects to be cached, and not blindly caching everything. By effectiveness, one implicit requirement is that during the time an object is in cache, its content is “fresh” or “properly validated without actual data transfer”. This is important because objects that have to be re-cached frequently increase network traffic and user-perceived network latency. Also, if the effectiveness of caching an object is too low, perhaps it should not be cached at all. This is to avoid replacing objects with higher effectiveness by those of lower effectiveness from cache. Analyzing the various factors that affect the effectiveness of caching an object is thus important. The name of this quantitative measurement used for caching proxy is called E-Cacheability Index.

1.1.2 Benefits of cacheability quantification to origin server

From the view of an origin server, the measurement can give content provider a reference to understand whether the content setting of their objects is effective for content delivery and caching reuse. It also suggests how these settings should be adjusted so as to increase the service competitiveness of their web content against other web sites in the same field.

Web pages of similar content for the same targeted group of users normally perform differently, with some being more popular, and some less popular. One of the possible reasons for such a difference could be the way the content in a web page is presented or being set. For example, dynamic objects aim to increase the attractiveness of a web page, but typically at the expense of slowing down the access of the page. Inappropriate freshness settings of an object will cause unnecessary validation by the caching proxy with the origin server, thus increasing the client access latency and bandwidth demand. Even worse, it is also possible for stale content to be delivered to clients if the caching is too aggressive but not accurate.

Our quantitative measurement can aid content providers to gauge their web content in terms of delivery, and in turn understand, tune and enhance the effective content delivert. Our measurement for the origin server is called Effective Content Delivery Index (ECD).

1.1.3 Incorrect settings of an object's attributes for cacheability

Research on content delivery reveals that for both caching proxy and origin server, the most important attribute that affects an object's cacheability is the correctness of its freshness period, which is called time-to-live (TTL). This is one of the few most important content settings that, if not properly set, will directly affect the reusability of an object.

Recent studies have also suggested that other content settings of an object, such as the response header's timestamp values or cache control directives, are often not set carefully or accurately [19][20][21]. This affects the calculation of an object's freshness period and possibly results in a lot of unnecessary network traffic. In addition, such wrong settings will potentially result in cache objects with fresh content to be requested repeatedly from the origin server, thus increasing its workload.

We propose an algorithm in this thesis, TTL adaptation. It separately analyzes different characteristics of an object, and in turn adjusts the parameters for TTL prediction of web objects with respect to time. This algorithm is suitable to be implemented in the content web server or reverse proxy.

1.2 Measuring an Object's Attributes on Cacheability

Our measurement on effectiveness in terms of content delivery is based on the modeling of all content settings of an object that affect its cacheability to obtain a numeric value index. These factors can be grouped into three attributes: availability, freshness and validation. They are briefly described below:

- *Availability* of an object is an action used to indicate if the object can possibly be cached or not.

- *Freshness* of an object is a period during which the content of the cached copy of an object in proxy is valid (or the same as that in the original content server).
- *Validation* of an object is an action that indicates the probability of the staleness of an object, using the frequency of the need to revalidate the object with the origin server as a measure.

To the caching proxy, these three attributes of an object determine the object's E-Cacheability effectiveness measure – E-Cacheability Index. If the object is available to be cached, the longer the period of freshness and the lower the frequency of re-validation will result in a higher E-Cacheability Index value. The higher value of the E-Cacheability Index indicates higher effectiveness to cache this object. The higher the effectiveness is, the more useful it is to be cached in the caching proxy. On the other hand, objects with low effectiveness value can give hints on reasons why certain content settings have negative impact on the cacheability of an object. This will have impacts in other proxy caching research areas such as replacement.

Thus the overall objective of the measurement used in the caching proxy, based on the assumption of the correctness of all the content settings, is to provide an index to describe the combinational effects of the content's settings with regards to the effectiveness of caching this content.

To the origin server, these three attributes of the object determine its effective content distribution (ECD). However, its emphasis is different when compared to the caching proxy. The measurement used in the origin server is based on the assumption that the content settings of objects might be incorrect. And the purpose of ECD is to find ways of adjusting these settings so as to increase the chance of reusability of content. This is achieved by helping content providers to understand whether the content settings of their objects are effective for content delivery, and for cacheable objects, whether the freshness period of an object in the cache is set correctly to avoid either stale data or over-demand for server/network bandwidth.

For cacheable content, if validation always returns an un-changed copy of the object, it will take up a lot of unnecessary bandwidth on the network. For non-cacheable content, requests that retrieve the same unchanged copy of the content will also result in a lot of unwanted traffic. Dynamic and secure content are just several examples of non-cacheable content that return a lot of unchanged content. For instance, a secure page could include many decorative fixed graphics that cannot be cached because they are on a secure page.

The validation attributes are represented here as (1) change probability for cacheable contents, (2) change rate, and (3) change percentage for non-cacheable contents.

1.3 Proposed TTL-Adaptation Algorithm

Research has shown that carelessness in the origin server can cause the freshness content setting to be inaccurate. Too short a freshness period will generate lots of unnecessary validation, which will waste a lot of bandwidth and lengthen user-perceived latency. Cases of unnecessary validation (where the content validated is not changed) are found to be about 90-95% out of all validation requests with origin servers on the network [3]. Too long a freshness period will increase the possibility of providing outdated web content to users, thus decreasing the credibility of web service.

With the above consideration, we propose a TTL adaptation algorithm to adjust the freshness setting for web content with respect to time. In our algorithm, we use the traditional statistical technique, the Gamma Distribution Model, which was proven as a suitable model for live-time distribution, to determine whether an object has any potential to be predicted. And our algorithm uses the Correlation Pattern Recognition Model to monitor and adjust the object's future TTL accordingly.

The adaptation algorithm determines the object's prediction potential by capturing its change trends in the recent past period from the corresponding gamma distribution curve that fits to its change intervals distribution in that period. And the correlation coefficient, which is calculated between the recent past period and the near following future period, will be monitored and used for the replacement of regularity. It predicts TTL(s) in the near following period should be either similar to the one(s) in the recent past period if the regularity is similar or adaptively changes the prediction value(s) if the regularity is replaced. This continuous monitoring and adaptation enables the predicted object's TTL to be close to its actual TTL with respect to time. Thus it effectively increases the correctness of an object's freshness attribute, and in turn lessens the possibility of unnecessary validation as well as the credibility of web services.

1.4 Organization of the Thesis

The rest of this thesis is organized as follows. In Chapter 2, we outline related research work on web object's cacheability, i.e. investigating an object's attributes related to caching and their limitations. We also investigate several current possible solutions that study an object's TTL and briefly comment their pros and cons.

In Chapter 3, we outline the factors in content settings that affect an object's cacheability according to HTTP1.1. A cache decides if a particular response is cacheable by looking at different components of the request and response headers. In particular, it examines all of the followings: the request method, the response status codes and relevant request and response headers. In addition, because a cache can either be implemented in the proxy or the user's browser application, the proxy or browser preferences will also affect an object's cacheability to some extent. This thesis mainly focuses on the caching proxy, so we discuss the proxy preference as the 4th factor in our model.

In Chapter 4, we will discuss the measurement of cacheability effectiveness from the perspective of a caching proxy. We propose EC, a relative numerical index value calculated from a formal mathematical model, to measure an object's cacheability. Firstly, our mathematical model determines whether an object is cacheable, based on the effects of all factors that influence the cacheability of an object. Secondly, we expand the model to further determine a relative numerical index to measure the effectiveness of caching a cacheable object. Finally, we study the combinational effects of actual factors affecting an object's cacheability through monitoring and tracing experiments.

In Chapter 5, the measurement, Effective Content Delivery (ECD), is defined from the origin server's viewpoint. It aims to use a numeric form of measurement as an index to help webmasters gauge their content and maximize content's reusability. Our measurement takes into account: (1) for a cacheable object, its appropriate freshness period that allows it to be reused as much as possible for subsequent requests, (2) for a non-cacheable dynamic object, the percentage of the object that is modified, and (3) for a non-cacheable object with little or zero content modification, its non-cacheability is defined only because of the lack of some server-hinted information. Monitoring and tracing experiments were conducted in this research on selected web pages to further ascertain the usefulness of this model.

In Chapter 6, we propose our TTL adaptation algorithm to adjust an object's future TTL period. The algorithm first uses the Gamma Distribution Model to determine whether the object has any potential for TTL prediction. Following that, the Correlation Pattern Recognition Model is applied to decide how to predict/adjust the object's future TTL. We demonstrate the usefulness of our algorithm in terms of minimizing bandwidth usage, maximizing content reusability, and maximizing accuracy of sending the most updated content to clients through the monitoring of content modification in selected web pages. We show that our TTL adaptation algorithm can significantly improve the prediction accuracy of an object's TTL.

In Chapter 7, we conclude the work we have done and present some ideas for future work.

Chapter 2 Related Work

In this chapter, we will outline related work to our research on web object's cacheability. The focus here is to study the influence of an object's attributes to caching and analyze their limitations. We also investigate some current solutions that study an object's time to live (TTL) and briefly comment on their pros and cons.

2.1 Existing Research on Cacheability

Research on Cacheability is focused on the conditions required for a web object to be stored in a cache. Cacheability is an important concern for web caching systems as they cannot exploit the temporal locality of objects that are deemed uncacheable. In general, the determination of whether an object is cacheable is via multiple factors such as URL heuristics, caching-related HTTP header fields and client cookies.

One of the earliest studies on web caching is the Harvest system [22], which encountered difficulty in specifying uncacheable objects. It tried to solve this by scanning the URL name to detect CGI scripts, and discarded large cacheable objects because of size limitation. Their implementations were popular at the advent of the web [23].

Several trace based studies investigated the impacts of caching-related HTTP headers on cacheability decisions. One of the earliest studies was performed by University of California at Berkeley (UCB) in 1996 [24], in which they collected traces from their Home IP service at UCB for 45 consecutive days (including 24 million HTTP requests). They analyzed some of the header content settings with respect to caching, including "*Pragma: no-cache*", "*Cache-Control*", "*If- Modified-Since*", "*Expires*" and "*Last-*

Modified". They also analyzed the distribution of file type and size. However, they did not look at all HTTP response status codes, and HTTP methods. They also did not discuss cookies, which make an object non-cacheable in HTTP 1.1. Ignoring cookies, their results showed that the uncacheable results were quire low, similarly for the CGI response.

Feldmann et al. noticed the biasness of the results from [24] and considered cookies in their experiments [25]. They collected traces from both dialup modems to a commercial ISP and clients on a fast research LAN. They obtained more statistics on the reasons for uncacheability. These include whether a cookie was present, whether the URL had a '?', and on header content such as Client Cache-Control, Neither GET nor HEAD, Authorization present, Server Cache-Control. Their results showed that the uncacheable results due to cookies could be up to 30%. Later studies on different traces [26][27] showed that the overall rate of uncacheability was as high as 40%. However, they did not look at all HTTP response status codes. They also did not mention the *Last-Modified* header in the response, which is essential for browsers and caching proxies to verify an object's freshness.

Other research studies are based on active monitoring [28]. Investigations are made on the cacheability of web objects after actively monitoring a set of web pages of popular websites. The study obtained a low proportion of uncacheable objects ([24]), even though cookies were included into the request headers in their experiment. The explanation of the result was that most of web content that required cookies actually returned the same content for following references if the cookies were set to the value of the "Set-Cookie" header of the first reference. However, their requests did not consider users' actions, and thus it is possible that the following references after the first reference may possibly cause different cookie value settings once users entered some information.

Such content customizations could not be detected under their data collection method. Their results also showed one important point in that dynamically generated web objects may not always contain content modifications.

Another research paper [29] investigated even more details about object non-cacheability such as dynamic URLs, non-cacheable HTTP methods, non-cacheable HTTP response status codes, and non-cacheable HTTP response headers. It also tried to find out the causes behind some of their observations, such as why the server does not put the *Last-modified* header with the file. However, it did not group reasons into complete entities and analyzed their combinational effects. Instead, it only focused on the discussion for each individual reason separately.

The research papers discussed above only focused on non-cacheable objects. They did not discuss on how cacheability affects cacheable objects, therefore not offering a balanced view.

The research by Koskela [30] presented a model-based approach to web cache optimization that predicts the cacheability value of an object using features extracted from the object itself. In this aspect, it is similar to our work. The features he used include a certain number of HTML tags existing in the document, header content such as *Expires* and *Last-modified*, content length, document length and content type.

However, it was mentioned by Koskela that building the model requires vast amount of data to be collected and estimating the parameters in the model can be a computationally intensive task. In addition, even though Koskela delves into an object's attributes, his focus on web settings is relatively narrow, only on a few header fields. His research is only valuable to the optimization of web caches, and those attributes he omits can potentially aid content providers to optimize their web content for delivery.

More complete analysis on content uncacheability can be found in [31][32]. [31] concluded that main reasons resulting in uncacheability included responses from server scripts, responses with cookies and responses without “Last-Modified” header. [32] proposed a complex method to classify content cacheability using neural networking.

From previous studies on cacheability of content, it has been discovered that a large portion of uncacheable objects are dynamically-generated or, have personalized content. This observation implies then of the potential benefits of caching dynamic web content.

2.2 Current Study on TTL Estimation

In traditional web caching, the reusability of a cached object is in proportion to its TTL value. The maximum value of the TTL is the interval between caching time and the next modification time. To improve on the reusability of a cached object, proxies are expected to perform, as accurate as possible, estimations of the TTL value of each cacheable object. Most of the rules of TTL estimation are derived from the statistical measures of object modification modeling. Rate of change (also known as average object lifespan) and time sequence of modification events for individual objects are the most popular subjects in object dynamics characterization.

Research on web information system has shown that the change intervals of web content can be predicted and localized. Several early studies investigated the characteristic of content change patterns. Douglis’ [33] study on the rate of change of content in the web was based on traces. He used the *Last-modified* header content to detect the changes in his experiment. Investigations focused on the dependencies between the rate of change and other content characteristics, such as access rate, content type and size. Craig [34], on the other hand, calculated the rate of change based on MD5 checksum. The research in [28] monitored daily the content changes on a selected group of popular websites, and noticed the change frequency of HTML objects tend to be higher in commercial sites than those in education sites. Yet another research [35] discovered that, based on monitoring on a weekly basis, web objects with a higher density of outgoing links to larger websites, tend to have a higher rate of change. All of the experiments (including later efforts in [27] and [36] confirming the results in [33]) showed that images and unpopular objects almost never change. They also showed that HTML objects were more dynamic than images.

Time sequence of modification events for a web object is another focus in the characterization of content dynamics. The lifespan of one version of an object is defined to be the interval between its last modification and its next modification. Therefore, the modification event sequence can also be viewed as the lifespan sequence. Research conducted in [37] noticed that the lifespan of a web object is variable. The study in [38] investigated the modification pattern of individual objects as a time series of lifespan samples and then applied the moving average model to predict future modification events. Both studies above pointed out that better modeling on object lifespan can improve TTL-based cache consistency.

Since then, researchers have put in considerable effort on modeling the whole web content because it is very important for information system to keep up with the growth and changes in the web. Brewington [35] modeled the web change as a renewal process based on two assumptions. One of the assumptions was that the change behavior of each page is according to an independent Poisson process. The other assumption was that every time a page renews its Poisson parameter, the parameter will follow a Weibull distribution across the whole population of web pages. He proposed an up-to-date measure for indexing a large set of web objects. However, as his interest was to reduce the bandwidth usage of web crawlers, the prediction of content change on individual objects, which is what web caching research is interested in, was not addressed.

Cho [39] proposed several improved frequency estimators for web page based on a simple estimator (number of detected changes/monitoring periods). Theoretical analysis for the precision of each estimator was based on the assumption that the change behavior of each page is according to an independent Poisson process. She also compared the accuracy of each estimator using data from both simulation and real monitoring. In his simulation, he generated synthetic samples from a series of gamma distributions and compared the effectiveness of multiple estimators. She pointed out that the purpose to choose a series of gamma distributions instead of exponential distributions was to consider the performance of each estimator under a "not quite Poisson" distribution for the page change occurrence. Both of them observed the change daily because they were interested in the update time of a web information system. It is a limitation to the study, as such a large time interval is too long to capture the essential modification patterns of web content for caching.

Squid [40], as an open source system for research, uses a heuristic policy known as the last-modified factor (LM-factor) [41] to predict every accessed object's TTL. The algorithm is based on the traditional caching standpoint that most of the objects are static, which means changes in older objects do not occur quickly. Therefore, its principle is that young objects are more likely to be changed soon because they have been created or changed recently. Similarly, old objects that have not been changed for a long time are less likely to be changed soon.

From the studies above, one common observation is that different objects have different patterns of modification. In the traditional TTL-based web-caching, accurate predictions is necessary to avoid redundant revalidations of objects whose next modification time has not arrived yet. However, it is more and more evident that current modification prediction heuristics cannot achieve acceptable levels of accuracy for web objects, all of which have different modification patterns. For instance, our real-life experience revealed that, contrary to the LM-factor algorithm, the longer the object does not change, the greater the possibility for it to change. Thus Squid either generates a lot of stale objects or causes unnecessary revalidation of object freshness.

The rate of change in today's web objects is very rapid, which inspires us to change the standpoint from the static perspective of an object to the dynamic perspective. In order to improve the above situation, there is a need to analyze individual object's change behavior separately and predict unique TTL for different objects according to each object's individual changing trend. Furthermore, to be as close to the actual TTL as possible, the prediction parameters should be continuously monitored and adaptively changed if required. Thus it is necessary to propose this kind of adaptive prediction algorithm – our TTL adaptation algorithm. Our algorithm is suitable to be implemented either in the reverse proxy or in the origin content server.

2.3 Conclusion

Previous research has focused on the statistical analysis on an object's attributes related to cacheability. Compared with our object's cacheability measurement, most of them do not delve into all attributes of an object attributes with regards to cacheability.

They discussed individual attributes separately, and have not studied the combinational effects of relevant attributes. They also only focused on non-cacheable objects and did not study how cacheability affects cacheable objects.

Except for Squid's LM-factor algorithm, existing studies on the object's Time-To-Live (TTL) mainly focus on getting an object's change frequency distribution for further web caching research. They did not use their distribution result to predict the value of object's future TTL. Compared with our algorithm that adjusts individual object's TTL based on the change of its own character, Squid's algorithm uses heuristic method to estimate that all objects that have not changed for a long time must have long future TTL and all recently changed objects must have short or zero future TTL. This argument, as we have shown in the later part of the thesis, might not hold.

Chapter 3 Content Settings' Effect on Cacheability

In this chapter, we outline the factors in content settings that affect an object's cacheability according to HTTP1.1[42]. A cache decides if a particular response is cacheable by looking at different components of the request and response headers. In particular, it examines all of the followings: (1) request method, (2) response status codes, and (3) relevant request and response headers. In addition, because a cache can either be implemented in the proxy or the user's browser application, the proxy or browser preferences will also affect an object's cacheability to some extent. This thesis mainly focuses on the caching proxy, so we discuss the proxy preferences as the 4th additional group of factors besides the three listed above.

3.1 Request Method

Request methods are significant factors to determine cacheability; they include GET, HEAD, POST, PUT, DELETE, OPTION and TRACE. Of these, there are only three kinds of methods that have potentially cacheable response contents: GET, HEAD, and POST. GET is the most popular request method, and responses to GET requests are by default cacheable. HEAD and POST methods are rare. The former response messages do not include bodies, so there is really nothing to cache, except using the response headers to update a previously cached response's metadata. The latter is cacheable only if the response includes an expiration time or one of the *Cache-Control* directives that overrides the default.

3.2 Response Status Codes

One of the most important factors in determining cacheability is the HTTP server response code. The three-digit status code, whose first digit value ranges from 1 to 5, indicates whether the request is successful or if some kind of errors occurs. Generally, they are divided into three categories: cacheable, negatively cacheable and non-cacheable. In particular, negatively cacheable means that, for a short amount of time, caching proxy can send the cached result (only the status code and header) to the client without fetching it from the origin server.

The most common status code is 200 (OK), which means that the request is successfully processed. The relevant response from this request is cacheable by default and there is a body attached. 203 (Non-Authoritative Information), 206 (Partial Content), 300 (Multiple Choices), 301(Moved Permanently), and 410 (Gone) are also cacheable. However, except for 206, they are only announcements without body.

204 (No Content), 305 (Use Proxy), 400 (Bad Request), 403 (Forbidden), 404 (Not Found), 405 (Method Not Allowed), 414 (Request-URI Too Long), 500 (Internal Server Error), 502 (Bad Gateway), 503 (Service Unavailable), 504 (Gateway Timeout) are negatively cacheable status codes.

3.3 HTTP Headers

It is not sufficient to use only the request method and response code to determine if a response is cacheable or not. The final cacheability decision should be determined together with the directives in HTTP headers, to show the combinational effects on an object’s cacheability.

Although the directives in both request and response headers affect an object's cacheability, our discussion in this section focuses only on the directives that appear in a response. With one exceptional request directive (*"Cache-control: no-store"* in request) that we will discuss below, request directives don't affect object cacheability.

- Cache-control

  It is used to instruct caches how to handle requests and responses. Its value is one or more directive keywords that we will mention later. This directive can override the default of most status codes and request methods when determining cacheability. There are several keywords as detailed below:

  – "*Cache-control: no-store*" directive keyword, appearing either in request or response, is a relatively strong keyword to cause any response to become non-cacheable. It is a way for content providers to decrease the probability that sensitive information is inadvertently discovered or made public.

  – "*Cache-control: no-cache*" and "*Pragma: no-cache*" don't affect whether a response is available to be cached or not. It instructs that the response can be stored but may not be reused without validation. In other words, a cache should validate the response for every request if the content of the request has been cached. The latter is the backward compatibility with HTTP1.0. Both HTTP versions have the same meaning for this.

  – "*Cache-control: private*" makes a response to be non-cacheable for a share cache, like caching proxy, but cacheable for a nonshared cache, such as browser. It is useful if the response contains content customized for just one person, thus the origin server can use it to track individuals.

  - "*Cache-control: public*" makes a response to be cacheable by all caches.
  - "*Cache-control: max-age*" and "*Cache-control: s-maxage*" directives hint the object is cacheable. They are alternate ways to specify the expiration time of an object. Furthermore, they have the first priority over all other expiration directives. The slight difference is that the latter only applies to shared caches.
  - "*Cache-control: must-revalidate*" and "*Cache-control: proxy-revalidate*" hint the object is cacheable. They force the response to do validation when expired. Similarly, the latter only applies to shared caches.

- "*Last-Modified*"

  It makes a response cacheable for a caching proxy that uses the LM factor to calculate an object's freshness period, such as that in Squid. And it is one of the most important headers to be used for validation.

- "*Etag*"

  It doesn't affect whether a response is available to be cached. But if other factors cause an object to be cached, the header hints that the cache should perform validation on the object after its expiration time.

- "*Expires*"

  It indicates that a response is cacheable. It specifies the expiration time of an object. However, its priority is lower than those of "*Cache-control: max-age*" and "*Cache-control: s-maxage*".

- "*Set-cookie*"

  It indicates that the response is non-cacheable. A cookie is a device that allows an origin server to maintain session information for an individual user among his requests [43]. However, if it is placed in "*Cache-control: no-cache = Set-cooki*e", it only means that this header may not be cached but this will not affect the whole object's cacheability.

3.4 Proxy Preference

A cache is implemented in the caching proxy, so proxy preference also determines an object's cacheability. In this thesis, we will use the Squid proxy as an example for caching proxy because it is the open proxy system for research purposes and is the world's most popular caching proxy being deployed today. For Squid, except for the protocol's rules discussed above, its preferences that determine a response to be non-cacheable (when the request method is GET and response code is 200) include:

- 'Miss public when request includes authorization'

  It means that without "*Cache-control: public*", response directive including "*WWW-Authenticate*" means that the server can determine who is allowed to access its resources. Since a caching proxy does not know which users are authorized, it cannot give out invalidated hints. So caching it may be meaningless.

- "*Vary*"

  It is used to list a set of request headers that should be used to select the appropriate variant from a cached set [44]. It determines which resource the server returns in its response. Squid still has not implement it yet, and this makes an object to be non-cacheable.

- "*Content-type: type-multipart /x-mixed-replaced*"

It is used for continuous push replies, which are generally dynamic and probably should be non-cacheable.

- "*Content-Length > 1Mbytes*"

  It indicates that it is less valuable to cache a response with large body size because such an object occupies too much space in the cache, and may result in more useful smaller objects being replaced from cache.

- "From peer proxy, without Date, Last-Modified and Expires"

  It seems non-beneficial to cache a reply from peer without any *Date* information, since it cannot be judge whether the object should be forwarded or not.

3.5 Conclusion

Whether an object can be cached in an intermediate proxy is determined by its cacheability content settings. These settings include request method, response status codes and its relevant headers. Proxy preference also plays an important role in deciding cacheability. According to all these factors, we will propose two measurement models in Chapter 4 and Chapter 5 to measure how effective an object's content settings on cacheability is, from the aspect of the caching proxy and the origin server respectively.

Chapter 4 Effective Cacheability Measure

In this chapter, we will discuss our effective measurement from the perspective of the caching proxy. We propose Effective Cacheability measure, also call E-Cacheability Index, a relative numerical measurement calculated from a formal mathematical model, to measure an object's cacheability quantitatively. In particular, the followings will be discussed:

- the cacheability of information that passes through a proxy cache,
- define an objective, quantitative measure and its associated model to quantify the cacheability potentials of web objects from the view point of a proxy cache,
- evaluate the importance of cacheability meansure to its deployment in proxy cache. The larger the value is, the higher will be the potential for an object to be kept in proxy cache for possible reuse without contacting the original server, and
- evaluate different factors affecting the cacheability of web objects.

4.1 Mathematical Model - E-Cacheability Index

The final decision on the cacheability of an object is actually made in the caching proxies. Apart from obeying the HTTP protocol's directives, caching proxies also have their own preferences to determine whether they should cache the object according to their own architecture and policies. In other words, even though a response is cacheable by protocol rules, a cache might choose not to store it.

Many caching proxy software include heuristics and rules that are defined by the administrator to avoid caching certain responses. As such, caching some objects is more valuable than caching others. An object that gets requests frequently (and results in higher cache hits) is more valuable than an object that is requested only once. If the cache can identify non-frequently used responses, it will save resources and increase performance by not caching them.

Thus, to better understand an object's cacheability, we should first analyze the combinational effects of relevant content settings on the effectiveness of caching an object. For this purpose, our method employs an index, called the *E-Cacheability Index* (Effective Cacheability Index), which is a relative numerical value derived from our proposed formal mathematical model of object cacheability. This *E-Cacheability Index* is based on its three properties – object availability to be cached, its freshness and its validation value.

4.1.1. Basic concept

From basic proxy concept, we understand that three attributes determine an object's *E-Cacheability Index*. They are object availability to be cached, data freshness and validation frequency. Their relationship is shown in the equation below.

$$\textit{E-Cacheability Index} = \textit{Availability\_Ind} * (\textit{Freshness\_Ind} + \textit{Validation\_Ind}) \qquad (4.1)$$

Unlike normal study on object cacheability, which just determines if an object can be cached), *E-Cacheability Index* goes one step further. It also measures the effectiveness of caching an object by studying the combinational effect of the three factors of caching availability, data freshness, and validation frequency.

In the equation above, the *Availability_Ind* of an object is used to indicate if the object is available for caching or not. If the object is not available, the *E-Cacheability Index* of the object is zero. Thus, all non-cacheable objects have an *E-Cacheability Index* of zero, and under this case, the meaning of the other terms (*Freshness_Ind* and *Validation_Ind*) is undefined. Hence, *Availability_Ind* is in the most dominant position in our measurement.

After the indication of whether the object is cacheable from the *Availability_Ind* attribute, the *Freshness_Ind* and *Validation_Ind* attributes are then important to measure how effective the caching of this object is.

Freshness_Ind is a period that indicates the duration of the data freshness of the object, and *Validation_Ind* is an index that indicates the probability of the staleness of an object, using the frequency of the need to revalidate the object.

It seems at the first glance that the validation effect should be included in the freshness definition. However, we separate these two factors because not all objects need to perform validation after its freshness period. For example, an object that has no validation header directives, such as "*Last-Modified", "Etag", "Cache-Control: must-revalidate"* will be evicted from the caching proxy. In addition, the caching proxy has a maximum cache period. So, even if an object has no validation information, it will be evicted.

Thus, the *E-Cacheability* Index is defined by these two attributes once an object is determined to be available for caching. The longer the period of data freshness and the lower the frequency of re-validation result in a higher *E-Cacheability* Index. Larger value of the *E-Cacheability Index* indicates higher potentials to cache this object. The higher the effectiveness, the more useful the object is in this aspect of consideration to be cached in the proxy.

Furthermore, for objects with smaller E-Cacheability Index, detailed analysis can give hints on which content settings have larger influence on the effective cacheability of an object. This can help to optimize the content settings for better caching.

In Equation (4.1), the "*" operator is used to handle the situation when an object is non-cacheable. As being seen in later sections, it will enforce the resulting index to be zero for non-cacheable objects. The "+" operator is used to separate the two situations of reusing the cached content by shifting the index to two exclusive regions – the region of negative values to indicate the need for revalidation each time an object is used, and the region of value greater than or equal to one to give a quantitative measure of the caching effectiveness.

In the next section, we will describe, based on the actual request methods, response codes, header fields, and proxy preferences that were discussed in Chapter 3, the detailed composition of each term in the equation above. We will use *I* in the equations to indicate request information, and *O* to indicate response information.

4.1.2. Availability_Ind

In this section we will discuss in detail on the term *Availability_Ind* in Equation (4.1). This term is defined as the overall composition of all factors that will possibly affect the caching availability of an object. The possible value of this term is 0 (non-cacheable) or 1 (cacheable).

The *Availability_Ind* of an object to be cached is dependent on several factors:

- The request method must be a method that allows its response to be cached.
- The status code of the response must be one that indicates that the object is cacheable.
- All header fields within the response that influence the availability of the object to be cached are considered.
- Proxy preferences within the response that influence the availability of the object to be cached are considered.
- If the relevant header fields in the request exist, it will mean that the object is cacheable and the response should act according to these information.

The *Availability_Ind* equation to consider all the above factors is shown below:

$$Availability\_Ind = I_{RM(A)} * O_{SC(A)} * O_{HD(A)} * O_{pp(A)} * I_{HD(A)} \tag{4.2}$$

where $I_{RM(A)}$ refers to the request method sent, $O_{SC(A)}$ refers to the response code related to object availability, $O_{HD(A)}$ refers to the header fields in the response that influence the availability of cacheability of the object, $O_{pp(A)}$ refers to the proxy preference in the response, and $I_{HD(A)}$ refers to the relevant header fields that influence availability in the request. The value of *Availability_Ind* is either zero (non-cacheable) or one (cacheable).

Equation (4.2) uses the associative operator (*), signifying that an object is non-cacheable (not available for caching) if there exists at least one factor that suggests the non-availability of the object in cache.

4.1.3. Freshness_Ind

The term *Freshness_Ind* in Equation (4.1) is defined as the overall composition of all factors that will possibly affect the data freshness of an object. The possible value of this term is zero for non-cacheable object to value greater than zero for cacheable objects.

The *Freshness_Ind* of an object can be determined by several factors:

- The request method must be one that allows its response to be cached.
- The status code of the response must be one that indicates that the object is cacheable.
- The header fields in the response that influences the freshness of an object will determine the freshness period of an object

The *Freshness_Ind* equation that considers all the above factors is shown below:

$$Freshness\_Ind = I_{RM(F)} * O_{SC(F)} * O_{HD(F)} \tag{4.3}$$

where $I_{RM(F)}$ refers to the request method sent, $O_{SC(F)}$ refers to the response code related to data freshness, and $O_{HD(F)}$ refers to the relevant header fields that influence the data freshness in the response.

Equation (4.3) associative operator (*) indicates that a non-cacheable response will result in the entire equation to be zero ($I_{RM(F)}$ and $O_{SC(F)}$). Otherwise, the *Freshness_Ind* value of the object will be determined by the relevant header fields in the response ($O_{HD(F)}$).

4.1.4. Validation_Ind

The term *Validation_Ind* in Equation (4.1) is defined as the overall composition of all factors that will possibly affect how valuable an object is in terms of its validation requirement. The possible values of this term is 0 (non-cacheable), -1 (if object must be revalidated each time even though it is cacheable), and greater than 1 (if object is cacheable).

The *Validation_Ind* of an object is determined by various factors:

- The request method must be one that allows its response to be cached.
- The status code of the response must be one that indicates that the object is cacheable.
- There are 3 terms to determine the length of the validity of an object:
  - All header fields in the request that influence the validity of an object.
  - All header fields in the response that influence the validity of an object.
  - All proxy preferences in the response that influence the validity of an object.

The Validation_Ind equation that considers all the above factors is shown below:

$$Validation\_Ind = I_{RM(V)} * O_{SC(V)} * OR\_val\text{-}op\ (I_{HD(V)}, I_{pp(V)}, O_{HD(V)}) \qquad (4.4)$$

where $I_{RM(V)}$ refers to the request method, $O_{SC(V)}$ refers to the status code, $I_{HD(V)}$ refers to the relevant header fields in the request that influence validation, $O_{HD(V)}$ refers to the relevant header fields in the response that influence validation, and $I_{pp(V)}$ refers to the proxy preferences in the request that influence validation. For function *OR_val-op* $(a_1, \ldots a_n)$, where $a_i \in \{-1, 0, 1\}$, its value is as follows:

$$OR\_val\text{-}op(a_1, \ldots a_n) = \begin{cases} -1 & \text{there exists at least one } a_i \text{ with value -1} \\ 1 & \text{there exists at least one } a_i \text{ with value 1 and no } a_i \text{ with value -1} \\ 0 & \text{all } a_i \text{ with value 0} \end{cases}$$

Equation (4.4) indicates that a non-cacheable response will result in the equation being zero ($I_{RM(V)}$, $O_{SC(V)}$). Otherwise, the value in the equation will either be 1 or -1 depending on the input parameters of the *OR_val-op* operator.

4.1.5. E-Cacheability index

Based on Equation (4.1), substituting the terms of all factor equations in (4.2), (4.3) and (4.4) into the equation, we have (in the rest of the chapter, we will use "*EC*" to shorten "*E-Cacheability Index*"):

$$\begin{aligned} EC = \; & I_{RM(A)} * O_{SC(A)} * O_{HD(A)} * O_{pp(A)} * I_{HD(A)} * (I_{RM(F)} * O_{SC(F)} * O_{HD(F)} + \\ & I_{RM(V)} * O_{SC(V)} * OR\_val\text{-}op\,(I_{HD(V)}, I_{pp(V)}, O_{HD(V)})) \end{aligned}$$

Since the request term in *Availability_Ind*, *Freshness_Ind* and *Validation_Ind* equations must be the same and is defined for the same object, let $I_{RM(A)} = I_{RM(F)} = I_{RM(V)} = I_{RM}$. Following the same argument, the status code of the response is the same response since all three factors are defined for the same object. Hence, let $O_{SC(A)} = O_{SC(F)} = O_{SC(V)} = O_{SC}$. Then,

$$EC = I_{RM} * O_{SC} * O_{HD(A)} * O_{pp(A)} * I_{HD(A)} * (I_{RM} * O_{SC} * O_{HD(F)} + I_{RM} * O_{SC} * OR\_val\text{-}op\,(I_{HD(V)}, I_{pp(V)}, O_{HD(V)}))$$

$$= I_{RM}^{2} * O_{SC}^{2} * O_{HD(A)} * O_{pp(A)} * I_{HD(A)} * (O_{HD(F)} + OR\_val\text{-}op(I_{HD(V)}, I_{pp(V)}, O_{HD(V)})) \quad (4.5)$$

The value of I_{RM} and O_{SC} can be easily determined:

$$I_{RM} = \begin{cases} 1, & \text{if method is GET, POST or HEAD} \\ 0, & \text{otherwise} \end{cases}$$

$$O_{SC} = \begin{cases} 1, & \text{if status code is 200, 203, 206, 300, 301, 410} \\ 0, & \text{otherwise} \end{cases}$$

Given the values of I_{RM} and O_{SC} above, Equation (4.5) can be simplified as:

$$EC = I_{RM} * O_{SC} * O_{HD(A)} * O_{pp(A)} * I_{HD(A)} * (O_{HD(F)} + OR\_val\text{-}op\,(I_{HD(V)}, I_{pp(V)}, O_{HD(V)})) \quad (4.6)$$

Equation (4.6) is the final mathematical formula to compute the effectiveness of caching an object. For the remaining terms, their corresponding header fields and proxy preferences, together with the value for each field and preference indicating their existence, are grouped in Table 4.1 below (we use groups C1-C7 to represent each of these terms). We use $x_{i(j)}$ to represent their values, and their details will be discussed. (*i* represents factor C1-C7, *j* represents the sub-term (either a header field or a proxy preference)).

Term in Equation	Relevant Header Fields/ Proxy Preferences	Existent Factor	Non-exist
C1: $O_{HD(A)}$	(1) Set-cookie (2) Cache-Control: private (3) Cache-Control: no-store	0 0 0	1 1 1
C2: $O_{pp(A)}$	(1) Miss public when request includes Authorization (2) Vary (3) Content-Type: multipart/x-mixed-replace (4) Content-Length = 0 (5) Content-Length >1Mbytes (6) From peer proxy, without Date, Last-Modified and Expires	0 0 0 0 0 0	1 1 1 1 1 1
C3: $I_{HD(A)}$	Cache-Control: no-store	0	1
C4: $O_{HD(F)}$	(1) Cache-control: max-age (2) Expires (3) Last-Modified (LM-Factor (an algorithm)) where priority of (1) > (2) > (3)	Seconds Seconds Seconds	0 0 0
C5: $I_{HD(V)}$	(1) Cache-Control: must-revalidate or Cache-Control: proxy-revalidate	-1	0
C6: $O_{HD(V)}$	(1) Cache-Control: no-cache, Pragma: no-cache (2) Cache-Control: must-revalidate or Cache-Control: proxy-revalidate (3) Last-Modified	-1 -1 1	0 0 0

Table 4.1 Terms and Their Relevant Header Fields

From Table 4.1, according to HTTP 1.1 (C1 represents $O_{HD}(A)$, C3 represents $I_{HD}(F)$) and Squid proxy preference (C2 represents $O_{pp}(A)$), the existence of any of the header fields of C1, C2 and C3, would cause the response object to be non-cacheable. Thus, we propose the value to C1, C2 and C3 $x_{i(j)}$ as 1 if it exists in the header of the object, or 0 if it does not exist. The *Availability_Ind* is defined as follows:

$$C1 * C2 * C3 = \prod_{i=C1}^{C3} x_i$$

$$= x_{C1(1)} * x_{C1(2)} * x_{C1(3)} * x_{C2(1)} * x_{C2(2)} * x_{C2(3)} * x_{C2(4)} * x_{C2(5)} * x_{C2(6)} * x_{C3(1)}$$

Only after the determination of I_{RM}, O_{SC}, and $O_{HD(A)}$ will the *Freshness_Ind* and the *Validation_Ind* be computed to obtain the effective cacheability measure of the object.

The freshness information is obtainable through any of C4(1) or C4(2) or C4(3). The unit of measure is delta-second (but notice that any value will do) and the value is obtained according to the calculation method of TTL (Time to Live) in rfc2616. Meanwhile, according to rfc 2616, the existence of C4(1) will override both C4(2) and C4(3), and the existence of C4(2) will override C4(3). Using $x_{C4(1)}$, $x_{C4(2)}$ and $x_{C4(3)}$ to represent C4(1), C4(2) and C4(3), the *Freshness_Ind* is defined by the value of the *OR_fresh_op* function:

$$\mathit{OR\_fresh\text{-}op}(\, x_{C4(1)}, x_{C4(2)}, x_{C4(3)}\,) = \begin{cases} x_{C4(1)} & \textit{(if } x_{C4(1)} \textit{ exists)} \\ x_{C4(2)} & \textit{(if } x_{C4(1)} \textit{ does not exist)} \\ x_{C4(3)} & \textit{(if both } x_{C4(1)} \textit{ and } x_{C4(2)} \textit{ do not exit)} \end{cases}$$

Similarly, we use $x_{i(j)}$ to represent the validation-related header fields in C5 and C6. The existence of C5(1) is indicated with -1, and is 0 otherwise. Such case is for C6(1) and C6(2). The reason to include the case of value being -1 is that according to rfc2616, the cache MUST perform validation each time a subsequent request for this object arrives, even if there is other freshness information. For term C6(3), its value is 1 if it exists and is 0 otherwise. Therefore, the *Validation_Ind* is given as follows:

$$\mathit{OR\_val\text{-}op}(\, x_{C5(1)}, x_{C6(1)}, x_{C6(2)}, x_{C6(3)}\,) = \begin{cases} -1 & \textit{one of } x_{C5(1)}, x_{C7(1)}, x_{C7(2)} \textit{ exists} \\ 1 & x_{C5(1)}, x_{C7(1)}, x_{C7(2)} \textit{ not exist, } x_{C6(3)} \text{ exists} \\ 0 & \textit{none of validation-related header exists} \end{cases}$$

Since the existence of C5(1), C6(1) and C6(2) will override all other header fields that might exist at the same time, the formula "*Freshness_Ind* + *Validation_Ind*" will be given as follows:

$$OR\_fresh-op(x_{C4(1)},x_{C4(2)},x_{C4(3)})+OR\_val-op(x_{C5(1)},x_{C6(1)},x_{C6(2)},x_{C6(3)})$$

$$\begin{cases} = OR\_val-op(x_{C5(1)},x_{C6(1)},x_{C6(2)},x_{C6(3)}) & \text{if } \textit{one of } x_{C5(1)},x_{C7(1)},x_{C7(2)} \textit{ exists} \\ = OR\_fresh-op(x_{C4(1)},x_{C4(2)},x_{C4(3)})+1 & \text{if } \textit{none of } x_{C5(1)},x_{C7(1)},x_{C7(2)} \textit{ exists,} \\ & \textit{while } x_{C6(3)} \textit{ exists} \end{cases}$$

Finally, our mathematical model will be represented as follows:

$$\textbf{EC} = \prod_{i=C1}^{C3} x_i * (OR\_fresh-op+OR\_val-op)$$

$$= x_{C1(1)} * x_{C1(2)} * x_{C1(3)} * x_{C2(1)} * x_{C2(2)} * x_{C2(3)} * x_{C2(4)} * x_{C2(5)} * x_{C2(6)} * x_{C3(1)}$$

$$* (OR\_fresh-op(x_{C4(1)},x_{C4(2)},x_{C4(3)})+OR\_val-op(x_{C5(1)},x_{C6(1)},x_{C6(2)},x_{C6(3)}))$$

From the analysis above, we can deduce that the possible values of *E-Cacheability Index* is as follows:

$$EC = \begin{cases} 0 & \textit{non-cacheable} \\ -1 & \textit{cacheable, but should validate in every request} \\ \geq 1 & \textit{cacheable} \end{cases}$$

When *EC* = 0, it is non-cacheable.

When *EC* = -1, it is cacheable, according to HTTP1.1. However, since the object has to be validated every time it is requested, and it may have insufficient freshness or validation information, the benefit of caching it will not be much. Many caching proxies, such as Squid, treat this kind of objects as non-cacheable. In our experiment, we will discuss this in the non-cacheable section, in accordance with the Squid's preference.

For $EC \geq 1$, $EC = 1$ means that the freshness period of the object is 0, which results in the need for revalidation each time a request for the object arrives at the caching proxy. However it is different from EC = -1 because of the sufficient information for validation. The larger EC is, the longer will be the period that the object can be cached, until it finally expires.

4.1.6. Extended model and analysis for cacheable objects

After the *E-Cacheability Index* classifies an object to be cacheable, we can further analyze the condition of $EC \geq 1$.

An object that is expired does not mean that it is useless for caching. If it has validation information, this might be able to lengthen its stay in cache by re-calculating its freshness period one more time. And this can potentially lead to more effective caching. In other words, an object with shorter freshness period in the first retrieval doesn't mean that it will be less effectively cached than an object with longer freshness period in the first retrieval. According to the model discussed in Section 4.1.5, if the value $EC \geq 1$, we can perform more precise calculation to further measure its relative EC. Here we will propose an extended mathematical model to obtain the value for such cases. It is very useful especially when the validation equation (mentioned in Section 4.1.5) is equal to 1.

From the model analyzed in Section 4.1.5, we have already classified objects as cacheable or non-cacheable. For objects with $EC \geq 1$, their *E-Cacheability Index* can be further considered as the benefit gained from caching over the cost of caching them.

Benefit and cost are defined separately by both the first retrieval effect and the revalidation effect. So the *E-Cacheability Index* can be re-defined as follows:

$$EC = \frac{benefit}{\cos t} = \frac{w*[(t_0 T_r) + \sum_{n=1}^{\infty} t_n * (T_r - T_v) * p^n]}{T_r + \sum_{n=1}^{\infty} T_v * p^n} \qquad (4.7)$$

w --- the weight of the object retrieval latency. If two objects' information are the same except that they have different object retrieval latencies, it is obvious that the longer the latency, the larger is its effective cacheability. *Since the object retrieval latency value takes an important part of EC*, it can be assigned to: $w = T_r$.

t_0 --- the cache time indicated by server after the first retrieval.

T_r --- the retrieval time spent on transfer. It is usually greater than the validation transfer time without content.

t_n --- the caching time after revalidation

T_v --- the validation time spent on transfer

p --- the probability for the content to be unchanged. In other words, the validation result is "304---Not modified".

n --- validation times

Assuming that $t_1=t_2=\ldots=t_n=t$ and because $\sum_{n=1}^{\infty} p^n = \frac{p}{1-p}$, Formula (4.7) can be simplified to:

$$EC = \frac{(t_0 T_r + t * (T_r - T_v) * \frac{p}{1-p}) T_r}{T_r + T_v * \frac{p}{1-p}} \qquad (4.8)$$

Here we use an example to illustrate the model clearly. Given two objects. Object A has no validation information, whereas object B has validation information. Their retrieval time is the same at 60 seconds. Object A can be cached for 2 hours before expiring. Object B can be cached for 1.5 hours, and after a 10 seconds validation delay, it can be cached for another 1.5 hours and then its content will be changed. We deduce that the probability of content of A being unchanged at its origin is 0, while the probability for that of B is 0.5.

Using the model, we can obtain the relative value: EC_A = 432,000 and EC_B = 509,142.86. This case demonstrates that validation may aid an object with a shorter cached period in its first retrieval time to be more effectively cached than an object that has no validation information but has a longer cached period in its first retrieval time.

4.2 Experimental Result

In this section, we performed trace simulation and analyzed objects' effective cacheability using the *E-Cacheability Index* (Equation (4.6) and Equation (4.7)) as given in Section 4.1.5 and Section 4.1.6. We obtained the raw trace data from National Laboratory for Applied Network Research (NLANR) [45]. We picked one day's sv trace (Oct, 17, 2001) that contains 86,718 total requests and 4.88 MB of data. We also modified Squid to record header information of HTTP requests that can be used as the input to our model. Then we repeated these 86,718 requests through the modified Squid and the corresponding information was recorded.

After we got the trace result, we first classified the objects' cacheability. Next, we proceeded to analyze the factors contributing to objects being classified as non-cacheable or cacheable according to our mathematical model in Section 4.6. Finally, for objects that are classified as cacheable, we perform further analysis using our model in Section 4.7.

The result of the various request methods that are from I_{RM} of our model is shown in Table 4.2. The table shows that the GET request method is used much more frequently (99.83% of all request methods) than all other request methods.

Method	GET	POST	HEAD
Percentage	99.83%	1.32%	0.32%

Table 4.2 Request Methods of Monitored Data

Of all requests that use the GET method, 93.78% of the replies return the status code 200 (Through the trace, 86.21% of these replies with status code 200 are cacheable, while the remaining 13.79% are non-cacheable). The distribution of status codes of replies of the GET method, as represented by O_{SC}, can be seen below in Table 4.3:

Status Code	**Percentage**
200 OK	93.78%
Other codes: 203 Non-Authoritative Information, 300 Multiple Choices, 301 Moved Permanently, 410 Gone	0.46%
Non-cacheable codes: 303 See Other, 304 Not Modified, 400 Unauthorized, 406 Not Acceptable, 407 Proxy Authenticate Required	0.91%
Negatively cacheable codes: 204 No Content, 305 Use Proxy, 400 Bad Request, 403 Forbidden, 404 Not Found, 405 Method Not Allowed, 414 Request-URI Too Large, 500 Internal Server Error, 501 Not Implement, 502 Bad Gateway, 503 Service Unavailable, 504 Gateway Time-out	5.74%

Table 4.3 Distribution of Status Codes of Monitored Data

In Section 4.2.2, 4.2.3, and 4.2.4, we will discuss experimental results for cacheable object. We will re-scale the percentage of cacheable replies (86.21% as was mentioned above) and detail all percentage calculations with respect to 100%. Then in

Section 4.2.5, 4.2.6, and 4.2.7, we will discuss experimental results for non-cacheable objects. We will also do the same re-scaling (rescale non-cacheable percentage 23.79% to 100%) in our analysis of the data.

4.2.1. EC distribution

Firstly, we summarize all the simulation results that used our proposed mathematical model (Equation 4.6) in Figure 4.1. The extension model (Equation 4.7) will not be used here. It will be used to perform further analysis on objects' relative *E-Cacheability Index* in Section 4.2.3. According to our model, the possible values for an object's cacheability are as follows (refer to Table 4.1): -1 means that the object is cacheable but its cacheability will be decided by proxy preference (Squid treats it as non-cacheable); 0 means non-cacheable; 1 means that the object's cached period is 0 but there is validation information; and value greater than 1 means that the object is cacheable. Figure 4.1 and Table 4.4 show the distribution of these cases. In Figure 4.1, the x-axis, except for -1,0,1 which indicates an object is cacheable or not, represents the cached periods of objects when the EC measure is greater than 1.

Status of Object	***EC* Value**	**Percentage (%)**
Cacheable but decided by proxy preference	-1	1.98
Non-cacheable	0	11.81
Cached period = 0, But object has validation information	1	7.07
Cacheable	> 1	79.14

Table 4.4 Object Status and the Corresponding EC Value versus their Percentage

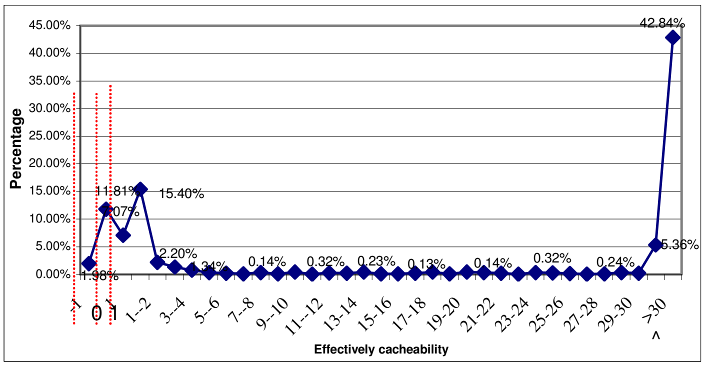


Figure 4.1 EC Distribution of all Objects

(X-axis represents the day of cached period (day), except first 3 values –1,0,1; y-axis represents the percentage of the objects)

Figure 4.1 and Table 4.4 shows that there are about 1.98% (EC = -1) non-cacheable objects due to Squid’s preference, making up a total of 13.79% (EC ≤ 0) non-cacheable objects. With respect to the remaining 86.21% cacheable objects, besides 7.07% objects with cached period of 0 and validation information, the distribution of most object’s cached periods is either in the period of ≤ 3 days, or > 1 month, as is shown in the figure.

The above result highlights that, based on our sample data, a high percentage of objects in the web are cacheable (86.21%). Using our *EC* model, it can further be broken down that a large portion of these cacheable objects (42.84% of all objects considered) have high *EC* values. It can be seen therefore that knowing whether an object is cacheable or not may not be sufficient enough for a caching proxy to be effective, as the space of a caching proxy is finite. Our *EC* model can sift out those more effective cacheable objects that should be cached, and this can improve the effectiveness of caching proxies.

4.2.2. Distribution of change in content for cacheable objects

In this section, we focus on cacheable data and discuss why we should further modify the *E-Cacheability Index* by using our extended mathematical model (Equation 4.7 in Section 4.1.6) to precisely measure their *E-Cacheability Index*.

In all the cacheable objects, about 7.07% of them have a freshness period of 0 but with revalidation information, thus resulting an *EC* value of 1. Among them, C4 (1) (*Cache-Control: max-age = 0)* made up 0.07%; C4 (2) *Expires* equaling *Date* made up about 5.13%, while the remaining 1.87% of objects has this *EC* value calculated through the C4 (3) LM-algorithm.

As discussed in our extended mathematical model in Section 4.1.6, since these objects have validation information, they can be revalidated freely. Even if their freshness period in the first time calculation is 0, it does not mean that they cannot be cached effectively. Since validation might lengthen their freshness periods, consequent requests can still get the fresh copy from the origin with less bandwidth and faster speed.

Figure 4.2 shows our experimental result on the effectiveness of caching such objects. We monitored those objects with *EC* = 1 in 5 minutes interval for a duration of about 90 minutes. The graph shows only those objects that are unchanged in the interval of 90 minutes. In particular, the moment when an object is changed, it will no longer be considered in the graph. It is because we are only interested in collecting data on how long an object that has freshness period of 0 can continue to be valid in the cache (i.e. until it is updated). Thus, the graph only shows those objects that did not change in the 5th minute. Observations continued to be made only on this group of remaining objects, to see which of them would be changed in the next revalidation, and so on.

For example, to obtain Figure 4.2, our program will perform the followings to achieve the result we mentioned above. Suppose there are 40 objects with *EC* = 1 in our monitoring list. Initially, we get these objects' bodies and keep a local copy of them. After 5 minutes, we retrieve the bodies of these 40 objects again. Comparing with the previously saved copies, we discover that there are 35 objects that remain unchanged. We will then note down in the graph, that 87.5% of objects remain unchanged. Next, we will remove the 5 modified objects from our monitoring list. After another 5 minutes, we will continue to monitor these 35 remaining objects. This procedure will last for 90 minutes.

From Figure 4.2, we notice that about 4.3% out of all these objects have real freshness period that is more than 1½ hours, and not 0, which means that this percentage of objects are actually quite cacheable. The graph also shows that even though the cached periods of such objects are 0, a large percentage of them remains unchanged for a certain period of time. There are several reasons why these objects have cached periods of 0, ranging from the origin server being set not to allow its objects to be cached, to the origin server not setting the header contents correctly. With our EC measure, we can sift out these objects, and perhaps investigate as to why they have a cached period of 0.

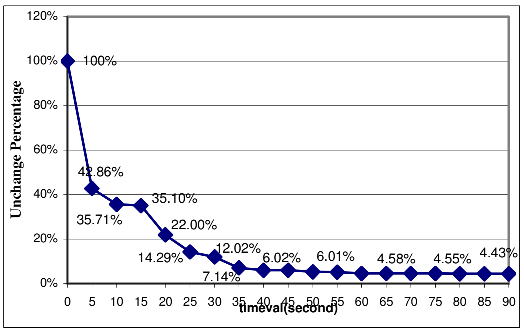


Figure 4.2 Every 5 Minutes Content Change Monitoring for Objects with Original Cached period of 0

(X-axis represents minutes, y-aixs repres- -ents percentage of unchanged objects)

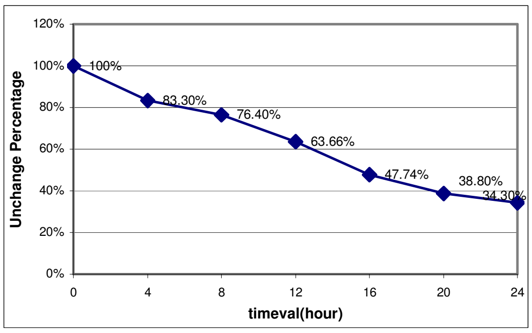


Figure 4.3 Every 4 Hours Content Change Monitoring for Objects with Original Cached period of 4 hrs

(X-axis represents hours, y-axis represents percentage of unchanged objects)

To further analyze the relationship between the real freshness period (cached period defined by the first retrieval time) and effective cacheability, we choose objects whose explicit freshness period is about 4 hours with validation information (this takes up 11.2% out of the total objects of EC > 1 in our experiment). The analysis method is the same as that for the objects with EC = 1. We perform validation in 4 hours interval for about 1 day. The percentage of objects that remains unchanged is shown in Figure 4.3. Comparing Figure 4.3 with Figure 4.2, it seems that the longer the explicit freshness period, the higher the possibility to lengthen freshness periods through performing validation. This is observed from the graphs that the percentage of objects that have changed is at a much slower rate in Figure 4.3 than those in Figure 4.2.

4.2.3. Relationship between EC and content type for cacheable objects

E-Cacheability Index can reveal the effectiveness cacheability in different content types. The legend for the numbers along the x-axis in Figure 4.4 and Figure 4.5 is shown in Table 4.5; the percentage of each legend taken out of all cacheable data is also included in Table 4.5:

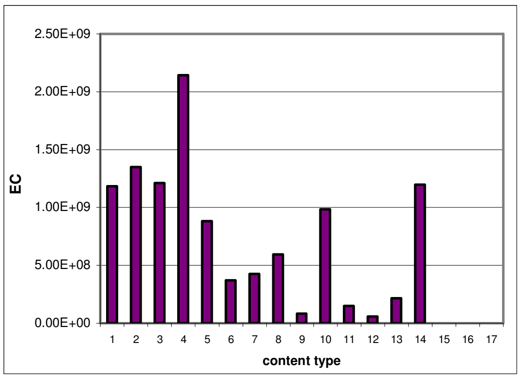


Figure 4.4 Relationship Between *EC* and Object's Content Type

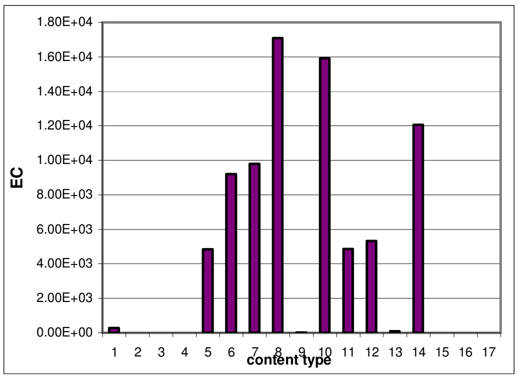


Figure 4.5 Relationship Between *EC* per Byte and Object's Content Type

Legend	1	2	3	4	5	6	7	8	9
Content type	audio/ mpeg	text/html	image/ jpeg	image/gif	application / octet-stream	video/ quicktime	application/ - xshockwave-flash	text/ plain	video/ mpeg
Perc-entage(%)	0.76	4.75	23.07	64.28	0.84	0.03	0.61	1.12	0.04
Legend	10	11	12	13	14	15	16	17	
Content type	application/x-javascript	application/zip	audio/ x-pn-real audio	Application /pdf	text/ css	application/x-zip-compressed	audio/x-mpeg	others	
Perc - entage(%)	1.89	0.62	0.01	0.07	0.93	0	0	0.99	

Table 4.5 Legend for the Numbers Along Category X-axis in Figure 4.4 and Figure 4.5

Figure 4.4 and Figure 4.5 show certain relationship between the *E-Cacheability Index* and content type. It is commonly agreed that image files do not change so often, so their *E-Cacheability Index* is expected to be much larger than those of other content types. They are thus the most effectively cached candidates; they make up the largest portion of cached objects.

HTML framework objects are usually changed at a very slow rate, as web masters often make only slight changes in web pages. The file type contents, such as the templates for HTML, javascript application and audio mpeg files also have quite effective cacheability.

Thus, from Figure 4.4, it can be seen that those content types that are most effectively cacheable logically have high EC values. Since the content size of the text file and some executive application files are comparatively smaller than that of other content types, their *E-Cacheability Index* per byte is larger than others' correspondingly.

4.2.4. EC for cacheable objects acting as a hint to replacement policy

In this section, we discuss how to use *E-Cacheability Index* as a hint for cache management such as web cache replacement policy.

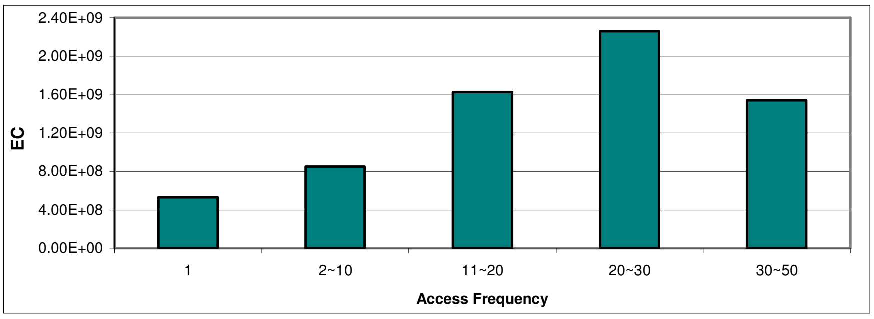


Figure 4.6 Relationship Between *EC* and Object's Access Frequency

Several approaches for replacement are widely used in web caching. One well-known approach is the LFU (Least Frequently Used) approach – a simple algorithm that ranks the objects in terms of frequency of access and removes the object that is the least frequently used [33]. Here we want to see whether there is relationship between our *E-Cacheability Index* and object's access frequency.

The largest access frequency was less than 50 times in our monitoring experiment. Lots of image files were accessed more than 10 times in our study. In the access frequency range of 20 to 30 times, most objects are JPEG files. Since the cached period for this kind of files is longer and their changing rate is lower (or the probability for contents to remain unchanged is higher), their E-Cacheability Index is higher, according to Equation 4.7 in Section 4.1.6.

Lots of text files, application files (like javascript file), and some image files are congregated in the access frequency range of 30-50 times. Though their access frequency is quiet high, the cached period of many text files and application files may be shorter than image files and their *E-Cacheability Index* may be lessened relatively. Refer to those objects that are accessed only once, some are still in the classification of *EC* = 1, which lessen the average *E-Cacheability Index* of this kind of objects.

From Figure 4.6, it seems that when the access frequency is less than 30 times in our experiment, the *E-Cacheability Index* is quite suitable in aiding the LFU replacement approach. Objects with higher EC value imply that they have a higher chance to be accessed, and hence the object will be a good candidate for caching. This can potentially improve the cache performance. In addition, the EC can also be viewed to a certain extent as the server hints for proxy cache replacement policy as the origin server can set the fields in such a way so as to hint to the proxy cache if an object will be cached effectively.

4.2.5. Description of factors influencing objects to be non-cacheable

As shown in Figure 4.1, under the status code 200, there are 13.79% objects that are non-cacheable. We use our mathematical model proposed in Section 4.1.5 and 4.1.6, and concentrate on the factors listed in Table 4.1 to analyze the cases. The factors consist of the existence or absence of various header fields affecting availability and freshness and validation of an object. More importantly, they might exist simultaneously instead of exclusively. Understanding the existence relationship among these factors is important because fixing one factor might or might not help in the overall object cacheability. This is what we will focus on: factors' combinational effects and their simultaneous existence relationship. Referring to Table 4.1, C1, C2, C3 are relevant to availability, C4, C5, C6 and C7 are relevant to freshness and validation. To simplify our discussion, we use numbers to represent these factors that were mentioned in Table 4.1. The representation is shown in Table 4.5.

Factor number	1	2	3	4	5	6	7	8	9	10
Factor contents	C1(1)	C1(2)	C1(3)	C2(1)	C2(2)	C2(3)	C2(4)	C2(6)	C6(1)	C6(3)

Table 4.6 Main Factors that Make Objects to be Non-cacheable

- Factor 1 C1 (1), *Set Cookie* header is used by servers to initiate HTTP state management with a client. Sever often traces some designated clients and this often makes it non-cacheable in the public caching proxy.
- Factor 2 C1 (2), *Cache-Control: private* indicates that the response is intended strictly for a specific user.
- Factor 3 C1 (3), *Cache-Control: no-store* identifies sensitive information, which tells cache servers not to store the messages locally, particularly if its contents may be retained after the exchange.
- Factor 4 C2 (1), the public caching proxy is of no use to cache a reply to a request containing an Authorization field without *Cache-Control: public*. Since the reply can only be served to the designated client, such client will often have a local copy in its local cache.
- Factor 5 C2 (2), The *Vary* header lists other HTTP headers that, in addition to the URI, determine which resource the server returns in its response. Squid still has not implemented it yet, which makes object non-cacheable.
- Factor 6 C2 (3), *Content-type: type-multipart /x-mixed-replaced* is used for continuous push replies, which are generally dynamic and probably should not be cacheable.
- Factor 7 C2 (4), the reply *Content-Length* is 0, thus there is no point in caching.

- Factor 8 C2 (6), it seems that there is no benefit to cache a reply from peer without any *Date* information, since it cannot be judged whether the object should be forwarded or not.
- Factor 9 C6 (1), when the header includes *Cache-Control: no-cache* or *Pragma: no-cache*, it instructs the cache servers not to use the response for subsequent requests without revalidating it. Whether the object is cacheable or not depends on the proxy's preference.
- Factor 10 C6(3), missing all the freshness information, especially *Last-Modified,* would result in the inability to calculate freshness or perform validation.

4.2.6. All factors distribution for non-cacheable objects

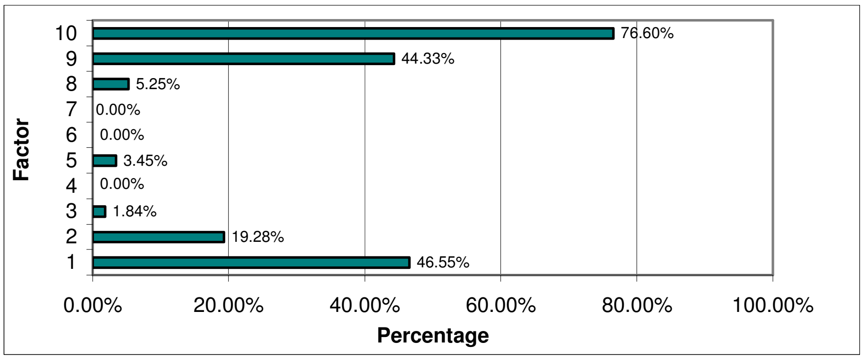


Figure 4.7 All Factors Distribution

From the discussion in Section 4.1.5, according to our mathematical model, we observed that factors 1-8 are relevant to availability and factors 9-10 are relevant to freshness and validation. Figure 4.7 (*y-axis is the factor number; x-axis is the percentage taken up by the indicated factor*) shows that factor 1 has the highest occurrence frequency among all factors affecting availability and factor 10 is the most important factor affecting freshness and validation. Factor 2 is another important reason that decides an object's cacheability and availability. Factor 8 also has significant impact as it prevents this category of objects from bouncing back-and-forth between siblings forever because they do not have the *Date* field, the most popular and required field to describe objects. In our study, 5.25% of objects do not have the *Date* field, which directly contributes to this factor and hence renders the objects to be non-cacheable.

With regards to validation, there are only 7.99% non-cacheable objects with C8 (4) (*Etag*), 0.81% with C8 (2)(*Cache-Control: must-revalidate)* and 0.03% with C8 (2) (*Cache-Control: proxy-revalidate)*. This shows that they are not so important in affecting an object's cacheability if an object is already non-cacheable. So we did not indicate these factors in Figure 4.7.

4.2.7. Non-cacheable objects affected by combination of factors

The ultimate purpose of our study on the factors that affect an object's cacheability is to find ways to improve caching. To a non-cacheable object, we can easily find out the factors that cause an object to be non-cacheable using our mathematical model proposed in Section 4.1.5. However, from our experiment, we realize that these factors often occur together. If we only concentrate on single factor impact without analyzing the relationship among them, we might not be able to have solution for caching improvement. On the contrary, if we can summarize the effects of the various combinations of factors, it will serve as good hints on which factors should be fixed first/together so as to improve the overall object cacheability.

As shown in Figure 4.8, Figure 4.9, and Figure 4.10, factor 10 is the most important reason causing objects to be non-cacheable. In other words, the missing header field *Last-Modified* is the major reason of objects being non-cacheable. Since very few objects include *Cache-Control: max-age* and *Expires*, this will enhance the role of the freshness guidance – *Last-Modified*. This header field also acts as the validation checksum. HTTP1.1 suggests that all objects should include this header field. Still, there are several reasons for missing the *Last-Modified* Data:

- The object is dynamically generated.
- The origin server asks the browser to fetch the object directly from it; it uses this approach to calculate the actual accesses or log user behavior.
- There is some mis-configuration problem with the web server.

Figure 4.8 shows that the 35.9% of all data being non-cacheable is caused by this single factor. Thus if this factor occurs without any combination with other factors, we may have the chance to fix this factor and improve the object's effective cacheability.

With regards to the combination of factors causing objects to be non-cacheable, we analyze the reasons of such combinations shown in Figure 4.9 and Figure 4.10. Factor 1 occurs most frequently, followed by factors 10, 9 and 2.

The combination of factors 1 and 10 affecting cacheability is probably due to the objects being generated dynamically and the server uses a state connection with the client.

The combination of factors 1 and 9 is probably due to the server having a tight connection with the client for user behavior tracing.

In Figure 4.10, the simultaneous occurrence of factors 1,9,10 indicates that servers emphasize the dynamic nature of these objects. As a result, there is no benefit to cache these objects at all. The case of factor 1 occurring with factor 2 is quite normal as it explicitly informs others that the server only cares for the designated client and others cannot share any information in this communication.

Factor 9 is one of the important factors that make objects to be non-cacheable. This is the proxy's preference. HTTP1.1 does not use "MUST NOT" to define this rule. It doesn't exactly prohibit cache servers from caching the response; it merely forces them to revalidate a locally cached copy. We may make these objects cached if they do not have other non-cacheable factors occurred. Thus, it is similar to the case *Cache-Control: max-age=0*. Cache servers need only revalidate their local cached copies with the origin server when a request arrives. This action can use this revalidation technique to improve an object's effective cacheability. Figure 4.9 shows that the most common case is the combination of factors 9 and 10. It seems that the server emphasizes that these objects are all dynamically generated.

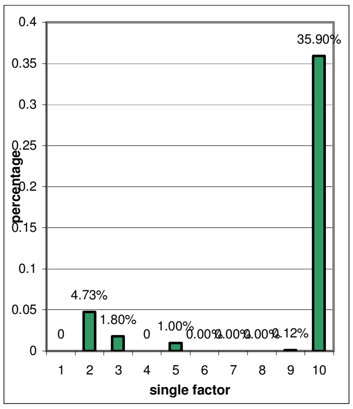


Figure 4.8 Single Factor

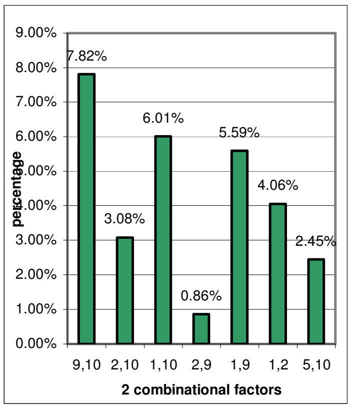


Figure 4.9 Two Combinational Factors

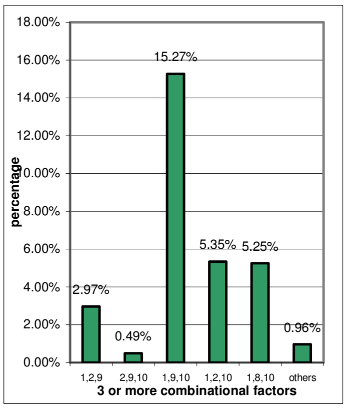


Figure 4.10 Three or More Combinational Factors

One of the main purposes of this study is to rank their importance in terms of improvement gained from fixing a given factor. In other words, we want to find out which factor will contribute to the largest improvement in cacheability if it is fixed. To do this, we perform multi-factors analysis and the result is shown in Figure 4.11. The graph shows that with this measurement parameter for optimization, factor 10 should be fixed first, followed by factor 9 and then factor 2.

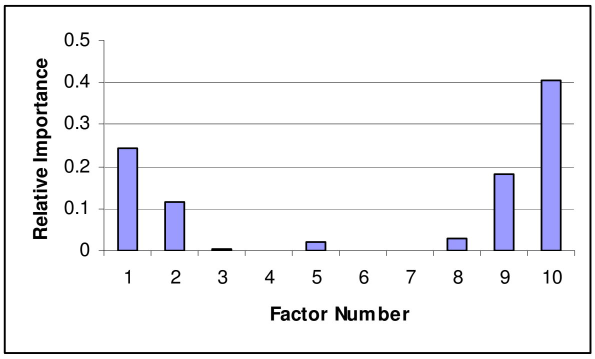


Figure 4.11 Relative Importance of Factors Contributing to Object Non-Cacheability

4.3 Conclusion

Despite the fact that there is a lot of research currently ongoing in web caching, most of them concentrate on whether an object should be cached. There is no further analysis on the cacheability of a cached object. The proposed Effective Cacheability (E-Cacheability Index) mathematical model presented in this chapter attempts to go one step further, by (i) first determining whether an object can be cached, and (ii) further determining the effectiveness of caching such an object, if it is cached. This further determination is in the form of a relative value, which can be used as a quantitative measurement for the effectiveness of caching the object.

In addition, most research only analyzed the influence of individual factors that affect the cacheability of an object. Little work is made in performing a detailed analysis on the relationship among these individual factors, and the effects of their simultaneous occurrence. This chapter conducted a detailed study and monitoring experiment to analyze the combinational effects of multiple factors that affect the cacheability of an object. This study further emphasized the usefulness of the *E-Cacheability Index* such as using E-Cacheability Index as a hint for replacement policies in the cache,

Chapter 5 Effective Content Delivery Measure

In this chapter, we would like to propose a similar measure for content cacheability, called the Effective Content Delivery (ECD) measurement, from the origin server's perspective. It aims to use numerical measurement as an index to describe object's cacheability in website, so that the webmasters can gauge their content and maximize the content's reusability. Our measurement takes into account the followings:

- For a cacheable object, we study its appropriate freshness period that allows it to be reused as much as possible for subsequent requests, and that subsequent validations should not be unnecessary.
- For a non-cacheable dynamic or secure object, we study the percentage of the object that gets changed, and
- For a non-cacheable object with low or zero content change, we study its cacheability when the non-cacheable decision is made due to the lack of some server-hinted information.

Trace and monitoring experiments were conducted in our study on web pages on Internet to further ascertain the usefulness of our model.

5.1 Proposed Effective Content Delivery (ECD) Model

The Internet is rapidly gaining its importance as a core channel for communication in many businesses. This has resulted in websites becoming more complex and with embedded objects to enhance the presentation of websites in order to attract their potential consumers.

"Content Delivery Measure" might have several possible assessing mechanisms, such as response time and so on. And one essential way for delivery improvement is to dig into content itself, through maximizing the potentials of content cacheability which in turns can reduce the delivery latency. If content can be moved closer to clients, this will result in shorter retrieval distance as well as higher delivery efficiency.

Therefore, the model is proposed based of objects cacheability. There are two categories of objects that we study, cacheable objects and non-cacheable objects. Due to the distinct nature of these two exclusive classes, their effectiveness needs to be studied separately. In our study, we propose a quantitative measurement of object cacheability for effective web content delivery, called the **Effective Content Delivery (ECD)** Index. In any of these two cases, the ECD measure indicates that the content settings of an object is more effective if the ECD measure gives a higher value, and vice versa. Each of these cases will be discussed in the sub-sections below.

5.1.1. Cacheable objects

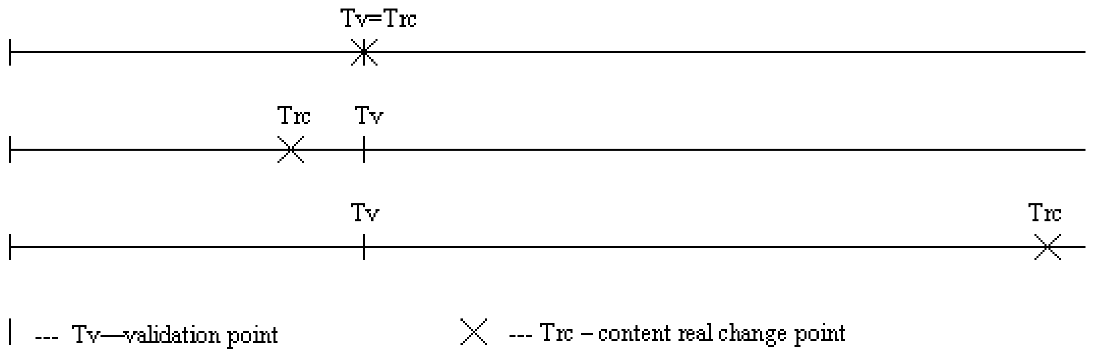
In this category, the ECD is defined for cacheable objects. Its main focus is to maximize object reusability so as to be able to be retrieved from the cache by subsequent requests as long as possible, thereby reducing the waiting time of users.

From our analysis in chapter 4, once an object is available to be cached, its freshness period and validation condition should be considered. To the origin server, objects with higher freshness periods and lower useless validation times tend to have a larger ECD measure.

Useless validations are validations that return an unchanged object from the origin server and this will result in unnecessary bandwidth consumption. If each time a validation is performed after an object has expired, and the result returned is the same copy of the object for another period of freshness, the freshness period might not be set properly.

The higher the rate-of-change of content for a given number of validations, the higher will be the ECD measure. A cacheable object with a high ECD measure tends to have an appropriate **freshness period** and a high **change possibility**, which indicates that the freshness period is set properly, as the copy of the object changes each time validation is made.

The following example explains how to set the **change possibility**. We use ***chpb*** to represent the change possibility.



$$\begin{cases} \text{Case 1:chpb} = 1 & \text{If } Tv = Trc & Chpb = 100\% \\ \text{Case 2:-1} < \text{chpb} < 0 & \begin{cases} \text{If } Trc < Tv & Chpb = Trc/Tv - 1 \\ \text{If } Tv < Trc < 2Tv & Chpb = 1- (Trc - Tv)/Tv \end{cases} \\ \text{Case 3:0} < \text{chpb} < 1 & \text{If } Trc >= 2Tv & Chpb = Tv/(100*(Trc - Tv)) \end{cases}$$

We can conclude that the larger the *chpb* value is, the more effective is the content delivery.

For example: Tv=3h, if Trc =2h ==> chpb = -1/3,

if Trc = 5h ==> chpb = 1/3,

if Trc = 8h ==> chpb = 0.006

5.1.2. Non-cacheable object

In this category, the ECD is defined for non-cacheable objects. Non-cacheable objects might not necessarily mean that their contents are constantly changing each time they are accessed. The change rate (how often the content really changes when it is accessed) and content change percentage (how much the content really changes when compared to the original content) are essential aspects in our analysis.

Although both factors need to be considered, their significance is different to different types of non-cacheable objects. Non-cacheable objects can be classified into four types, differentiated by the reasons that make them non-cacheable. They are (i) non-cacheable secure objects, (ii) non-cacheable objects directed explicitly from server, (iii) non-cacheable objects based on proxy preference, and (iv) non-cacheable objects due to missing headers. ECD for each of these four categories of objects will be discussed in details below:

- **Non-cacheable secure objects**

Secure objects usually refer to web objects that are encrypted for point to point transmission. A good example is information related to the submission of a user's private particulars on Internet (for example a credit card number, or a pin number for Internet banking). As the information requires confidentiality, such interactions need to be made on secure data transmission. However, it is observed that many websites enforce information confidentiality not just on the sensitive information but for the entire page, which have decorative objects and company logos that are definitely static and public. If the percentage of this relatively static, public portion of the page is higher than that of the secure portion, it will result in unnecessary bandwidth usage because of the improper reusability setting of content.

Higher value of the **change percentage (C_{perc})** (the percentage of a page's content that is changed) of a page indicates that at each content page transfer, the unnecessary work performed by the origin server and the amount of unnecessary bandwidth consumed will be lower. Therefore, objects with a higher C_{perc} should have a higher ECD measure.

However, it must be highlighted that due to the secure https protocol for the entire page, such page cannot be cached. Thus, if webmasters can separate the non-cacheable and cacheable portions of such pages, reuse of the cacheable portions will result in bandwidth saving and reduced access latency.

- **Non-cacheable objects directed explicitly by server**

In the header settings of such objects, there are explicit server hints specifying that they are completely unavailable for caching. Examples of such hints are the settings of “Cache-Control: private” or “Cache-Control: no-store”.

Such hints are representations of strong preferences directed from servers. They indicate that the whole objects are definitely non-cacheable. Furthermore, any intermediate proxies cannot interfere or modify them.

Besides considering the rate of content change, the percentage of content change in these pages is also an important factor. Therefore, the focus of ECD here is on the **change percentage** (**C_{perc}**) of content in these pages. If there is a high percentage of the page content that gets changed, it will be appropriate for the entire page to be retrieved from the origin server. Thus there is little benefit to cache portions of it because the server's setting is quite appropriate. And this will result in a high ECD measure of such a page. However, if the percentage is low, the delivery of this content from the server will be considered as ineffective, as a great portion of the page could be cached and reused. Thus, the ECD measure of such a page is low.

Similar to "non-cacheable secure objects", webmasters could possibly observe the ECD measure due to the percentage of change and make the necessary adjustment to get a higher ECD measure. This can be done by removing unnecessary objects in the page or by separating the objects into cacheable (for non-changing part) and non-cacheable (frequently changing part) groups.

- **Non-cacheable objects based on the caching proxy preference**

Besides the protocol rules (here we focus on HTTP1.1) that decide whether an object is cacheable or not, the caching proxy also makes decision based on its proxy preferences. Different proxies have different proxy preferences.

Objects in this category are not explicitly directed as non-cacheable. However, some wrong or inappropriate settings might cause the proxy misunderstand the object cacheability according to the proxy preferences. For example, the inappropriate setting in *Last-Modified* leads to negative freshness period calculated by the Squid proxy and this makes the object to be treated as non-cacheable.

To study whether the proxy's preferences is accurate enough to make decision on object cacheability, we apply the **change rate** (**C_{rate}**) of an object to measure the rate of change of the object whenever it is accessed. The **change rate** (**C_{rate}**) is the number of times an object really changes over its total access times. Higher values indicate that content validation for a cacheable object or the re-transfer for a non-cacheable object does not yield unnecessary work by the origin server (the fresh copy of the object is indeed different from the previous copy).

For example, a 100% change rate means that the content really changes in every validation request. A 0% change rate means that every time a caching proxy sends a validation request to the origin server, it always receives the response that the object is unchanged. In the latter case, making this object with 0% change rate as non-cacheable is inappropriate, as this will result in unnecessary work to the origin server and redundant traffic in the network.

- **Non-cacheable objects due to missing headers**

The study conducted in [27] found that 33% of HTML resources do not change. However, this portion of the resources cannot be cached because the origin server does not include cache directives that will enable the resource to be cached. Similar to the first case, [19][20][21] pointed out that cache control directives and response header timestamp values are often not set carefully or accurately. To solve this problem, webmasters require some helpful measurement to give hints on how these settings can be optimized. As these objects' measurements are similar to those of "non-cacheable objects based on the caching proxy preference", we also measure the change rate (**C_{rate}**) of the object.

5.1.3. Complete model and explanation

An object's cacheability is vital to the webmaster who wishes to design a webpage that is not too slow to be accessed. One aspect to achieve this goal is for him to take note of the cacheability of objects within the webpage. As mentioned in the previous section, objects should first be judged in which class (cacheable or non-cacheable) it belongs to because the ECD measure for these two types of objects is different. Thus, the model that we will propose in this section will use cacheability as the first and foremost term to be considered in the equation.

For cacheable objects, there are two main factors affecting the ECD measurement: (1) judging an object's cacheability whether an object is cacheable or not, and how long it can be cached, and (2) the object's change possibility when its freshness period has expired and the cache has to validate with the origin server. Furthermore, the cacheability of an object depends on two factors – **Availability_Ind** and **Freshness_Ind**, which were explained in detail in Sections 4.1.2 and 4.1.3.

For non-cacheable objects, the change rate and change percentage mentioned in Section 5.1.2 should both be considered for every object, so it overall effective value should be the combination (multiplication) of these two factors. However, as was mentioned in Section 5.1.2, the two factors have differing significance for different types of non-cacheable objects. The formula for ECD is thus given below:

$$\begin{cases} \text{For cacheable object:} \\ \quad \mathit{ECD} = (\mathit{Availability\_Ind} * \mathit{Freshness\_Ind}) \times \mathit{chpb} \\ \text{For non-cacheable object:} \\ \quad \mathit{ECD} = (C_{prec} \times C_{rate}) \end{cases}$$

The “*” operator handles the situation when the object is non-cacheable. The existence of non-cacheability factors will enforce the resulting index to be zero, otherwise is 1. “$\times$” is the normal “multiply” operator for the corresponding calculation.

From the discussion of the factors affecting the *Availability_Ind* and *Freshness_Ind* in Chapter 4, the equation for cacheable object can further be expanded into the following:

For cacheable object:

$$\text{ECD} = ((\prod_{i=C1}^{C3} x_i * OR\_fresh-op)\times chpb)$$

$$= (x_{C1(1)} * x_{C1(2)} * x_{C1(3)} * x_{C2(1)} * x_{C2(2)} * x_{C2(3)} * x_{C2(4)} * x_{C2(5)} * x_{C2(6)} * x_{C3(1)}$$

$$* OR\_fresh-op(x_{C4(1)}, x_{C4(2)}, x_{C4(3)})\times chpb)$$

The value of the change percentage (***C***$\boldsymbol{C_{perc}}$) is in percentage. The higher the value of C_{perc}, the lesser is the origin server’s unnecessary work and the network traffic.

$$\boldsymbol{C_{perc}} = \begin{cases} > 0 \text{ and } < 1 & \text{less effective, content does not totally change} \\ 1 & \text{most effective, content changes completely} \end{cases}$$

Similar to C_{perc}, the change rate ($\boldsymbol{C_{rate}}$) is also in percentage, and the higher the value, the more effective is the content settings.

$$\boldsymbol{C_{rate}} = \begin{cases} 0 & \text{least effective, validation object does unnecessary job} \\ > 0 \;\&\; < 1 & \text{less effective, content does not change in every validation} \\ & \text{most effective, content changes in every validation} \end{cases}$$

5.2 Result and Analysis of Real-time Monitoring Experiment

We chose 10 websites that are the most popular websites in computing industry that are listed in hot100.com [46] on April 17, 2002 (See Table 5.1). Attractive content might make them lead in the industry. However, different content settings that result in different content retrieval time might also be the other reason for their popularity. Here, we would like to try to compare their effectiveness in content settings by calculating their respective ECDs. Hopefully, this can help webmasters to enhance their website design.

Data were extracted and monitored from the first two levels of embedded objects of these 10 hot computing websites. They were collected using the "wget" software in Linux version. After they were filtered into unique ones, they were classified to 10 groups according to their homepages.

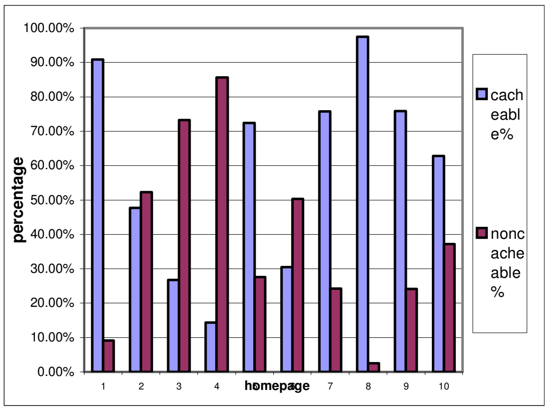
Squid, as the state-of-the-art caching proxy, was instrumented in the experiment to separate the objects into five types: (i) cacheable with validation information objects, (ii) non-cacheable secure objects, (iii) non-cacheable objects directed explicitly by server, (iv) non-cacheable objects based on proxy preference, and (v) non-cacheable objects due to missing headers. All these requests were successfully processed between the caching proxy and the origin server (server return code was 200).

For the cacheable objects, Squid automatically monitors and validates them as soon as they are not fresh according to its server-hinted freshness period. Every validation timestamp, server return code and all other related returned information were recorded. Such action was executed for one hundred times for all of these objects. For non-cacheable objects, no matter which type they belong to, the whole bodies were transferred and stored locally. They were compared with copies of their previous bodies in every one minute interval for 100 times to determine whether they were changed and what percentage they changed. After all these were finished, detail analysis was performed to extract useful statistics. To produce meaningful results, distribution or bar grafts were applied.

Figure 5.1 and Figure 5.2 represent the effectiveness of content delivery of the top 10 web pages. Figure 5.1 shows relative percentages of cacheable and non-cacheable objects. Because the percentage of cacheable image files in techdepot(2) is the largest among these 10 web pages, their longer freshness period enables its ECD value to be the highest one. This means that it has most effective content settings regarding to delivery. High percentages of cacheable objects in explorermicro(1), youlearn(8), pcpartsfinder(10) enable them to be re-delivered content more effectively.

No.	URL	No.	URL
1	shop.explorermicro.com	6	www.extremetech.com
2	www.techdepot.com	7	www.ebay.com
3	www.planetlearn.com	8	www.youlearn.com
4	www.nextag.com	9	www.mindmodel.com
5	www.voicerecognition.com	10	www.pcpartsfinder.com

Table 5.1 Web Sites Used in Our Simulation



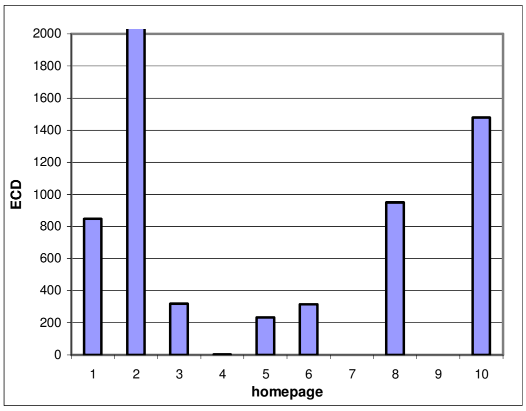
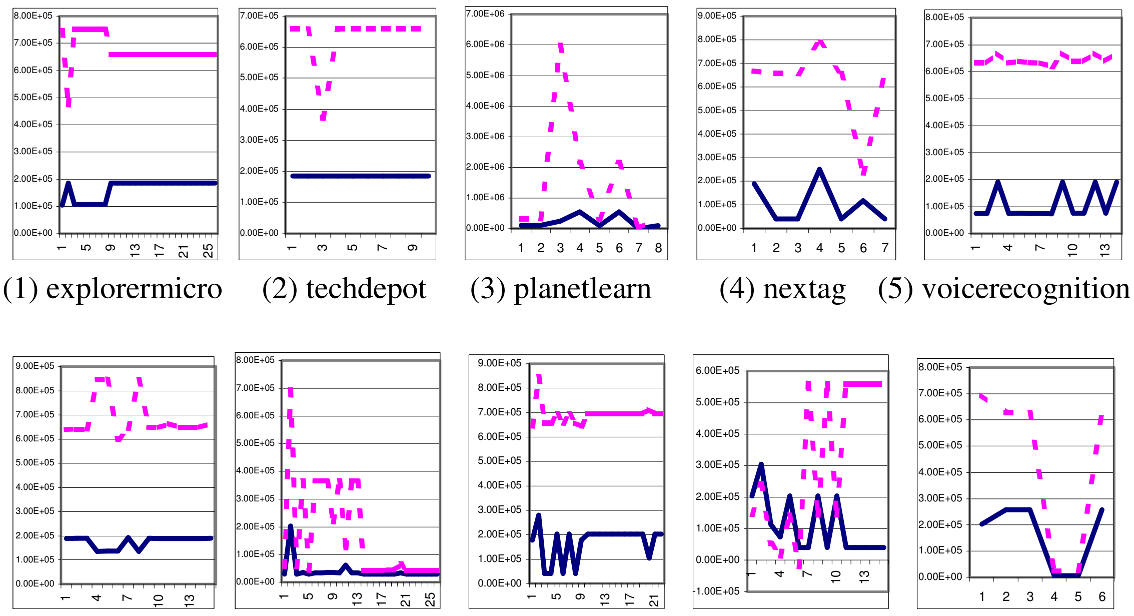


Figure 5.1 Cacheable, Non-Cacheable Objects Taken-Up Percentage (*left column is Cacheable, Right One is Non-Cacheable*)

Figure 5.2 Average ECD of Every Web Page



(1) explorermicro (2) techdepot (3) planetlearn (4) nextag (5) voicerecognition

(6) extremetech (7)ebay (8) youlearn (9) mindmodel (10) pcpartsfinder

Figure 5.3 Cacheable Objects' Average Server Directive Cached Period vs Real Changed Period (10 subgrap) (*x-axis is tracing times, y-axis is ECD*)

Figure 5.3 shows the difference between the average server directive cached period for cacheable objects and their real changed period. The broken line represents the real changed period, while the solid line represents the server directive cached period.

From the figure, we can deduce that most server directive cached period is very different from the real changed period. Most of them are quite conservative, which results in a lot of useless validations and wastes network bandwidth. Even worse, some servers even set a wrong directive, which causes the cache to serve outdated data to the clients.

Figure 5.4 indicates the average change possibility for cacheable object in every monitored web page. The change possibilities of these pages are all quite low. Ebay(7), mindmodel(9) even have negative change possibility, indicating that there might be incorrect content settings in these two web pages. The change possibility in techdepot(2) is quite high. This is one of the reasons that give it a high ECD value. It indicates that the content settings for its cacheable objects are quite effective.

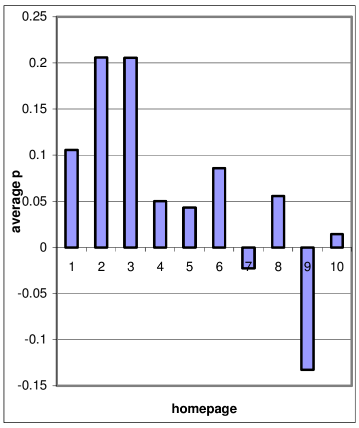


Figure 5.4 Average chpb for Cacheable Objects in Every Web Page

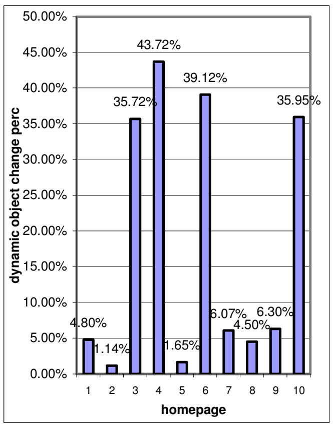


Figure 5.5 Average Change Percentage

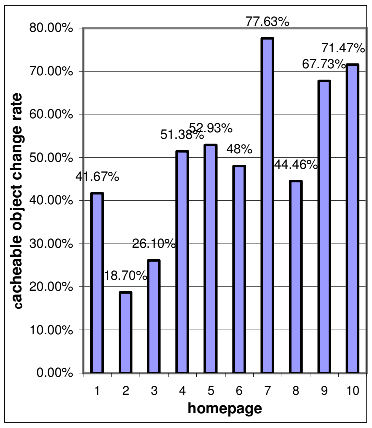


Figure 5.6 Average Change Rate

Referring to non-cacheable objects, Figure 5.5 and Figure 5.6 show their average change percentage and average change rate respectively. It seems that the relative static part of the page takes up more than 50% of the content. In other words, if the content provider can separate them to be cacheable and non-cacheable, it will result in substantial saving in network bandwidth. Figure 5.6 also shows that the change rate is quite low in techdepot(2) although its total ECD is quite high. It suggests to enhance its content setting in non-cacheable objects.

5.3 Conclusion

In this chapter, we proposed the Effective Content Delivery (ECD) model index as a mean to measure the effectiveness of websites' content in terms of delivery.

The model results in a simple numerical value. This kind of content self-checking gives webmaster hints on whether the content setting is appropriate for effective delivery. If the value is not satisfactory, the model can also help to trace the reasons behind, so that the webmaster can do proper adjustment based on the analysis result. The measurement details the three main elements of consideration in the model and gives reasons for their importance.

In order to show that our model gives good measurement for the content delivery effectiveness of websites, we also conducted real-time monitoring experiments on many actual commercial websites. We analyzed the differences in the content delivery of popular websites using our model, and showed that many of the more popular websites indeed score a higher value on our ECD model index. This model index could possibly be a useful tool for webmasters to understand, plan and enhance the web content settings of their websites in order to achieve a higher ECD measure.

Chapter 6 Adaptive TTL Estimation for Efficient Web Content Reuse

In this chapter, we propose a novel adaptive mechanism for accurate TTL (or Time-To-Live) estimation. Our proposal here is based on problems analyzed in previous chapters that most current mechanisms setting the TTL of object are not that satisfied.

6.1 Problems Clarification

The idea behind content delivery and distribution is to either cache or replicate copies of data to some proximity server near the client for faster access. Potentially, this can save network and server I/O bandwidth, provide better accessibility of web objects, and reduce client's perceived latency. One key pre-condition for this approach is the accurate setting of the TTL of a web object. The TTL of an object is the time period specified as an attribute to the copy of the object such that any use of the content within this period can be made without the need to contact (or revalidate) the original content server [33]. Note that the TTL is closely related to another attribute of the object, the life-span. The life-span of an object refers to the time period during which the content of an object is guaranteed to be "fresh" and is valid to be used [6]. While the life-span depends solely on the content nature (i.e. when the content is changed), the TTL setting might depend on not only on the life-span but also many more factors such as server storage policy, requirement for monitoring and tracking of client behavior, and the risk level of retrieving outdated content by clients.

Setting accurate TTL value for a web object is actually a complicated task. Since most (if not all) content servers do not guarantee 100% accuracy about the TTL of objects they provide on Internet, this setting process can actually be viewed as the content freshness risk analysis. Correct TTL setting, of course, can get the benefits that we mention above. However, in general, improper setting is often observed on Internet. This will result in one of the two possible consequences. If it is too aggressive, meaning that the TTL is set to be longer than it should be, there will be risk of retrieving outdated object content. On the other hand, if it is too conservative, meaning that the TTL setting is much smaller than it should be, the benefits of reduction of bandwidth usage, I/Os, and perceived latency might not be able to obtain.

Currently, there are two main places where the TTL setting will be done. The first place is the content server. This is done through server directives. In theory, this should be the ideal place to perform the TTL setting, as the content server, together with the site administrator, should have the best knowledge about the modification dynamics of a web object. However, what is observed on Internet is quite difficult. As we will show in the later part of this paper, the server directives are often found to be too conservative, resulting in over-demanding for network bandwidth. Furthermore, this is often done in an ad-hoc manner.

The second place is in the proxy cache where the TTL of an object might be calculated through predefined rules or formulas. For example, heuristic rules based on keyword matching in the URL name such as the presence of "cgi-bin", the suffix ".cgi(*.asp/*.php)", or a question mark "?" are often used to identify dynamic objects with TTL value of zero. SQUID, the most popular proxy cache being deployed, calculates the TTL of an object based on its Last-Modified time and the current time. (In SQUID, the

TTL setting for an object depends on the difference between the Last-Modified time and the current time. The argument behind is that the longer an object is not changed, the lesser will be the probability for it to be changed in future [47].) These are often found to be too simple because this setting mechanism is independent of the past history of the life-span of an object. Thus, it still results in one of the two TTL setting problems mentioned above (either too conservative to cause unnecessary bandwidth usage or too aggressive to cause reuse of outdated content).

Speculation on the improper TTL setting phenomenon in the content server suggests the following explanations. Firstly, the life-span value of an object is not a constant; hence adaptive TTL prediction will be needed. Note that it is interesting to observe that while quite a lot of efforts have been spent on the prediction of object's TTL, we cannot find literature to study the adaptability of predicted TTL of web objects. Secondly, TTL prediction is often too difficult for most system administrators to handle. And there lacks of an automatic adaptive mechanism to accurately set the TTL of web objects based on both the content modification history and input from the content provider. Thirdly, while web site administrators and content providers are experts in content management and creation, they often do not have good understanding on how performance can be lost in content delivery. The challenge comes from the dilemma that most efficient content management technologies such as ASP and database have negative side-effects on content delivery. Balancing these two aspects is often found not to be easy. Lastly, there are also observed cases where the improper setting is due to the carelessness of webmaster. This might even result in the data integrity problem of reusing the stale copies of web content in proxy/browser cache.

To address this problem, we start with the assumption that any predictable life-span of an object should generally be seen as a random variable defined over some probability distribution function (PDF) (such as Gamma or WeiBull functions which will be changed with respect to time. Then the TTL estimation of a web object can be viewed as two-steps process. The first step is to define the validity of predictability potentials, together with its associated expectation value (if any), of the life-span of a web object through stochastic process. Then the second step is to take care of the changing behavior of the PDF of the life-span through correlation pattern recognition modeling. Through active monitoring on the content updating dynamics of real web objects on Internet, we show that our TTL adaptation mechanism is far more accurate than both directives sent out by content servers and TTL settings by current proxy caches. This result is important because it allows content servers and proxy caches to conduct their own content freshness risk analysis for more efficient web caching.

In this chapter, we sometimes interchange the use of the two terms "TTL" and "life-span" because most (if not all) related works, HTTP protocol definition, and proxy/web servers do not provide any differentiation between them and they often just use the term TTL. This is possible because without additional constraints to TTL setting, such approximation for their equality is quite reasonable. However, we would like to point their fundamental difference in the beginning of this section.

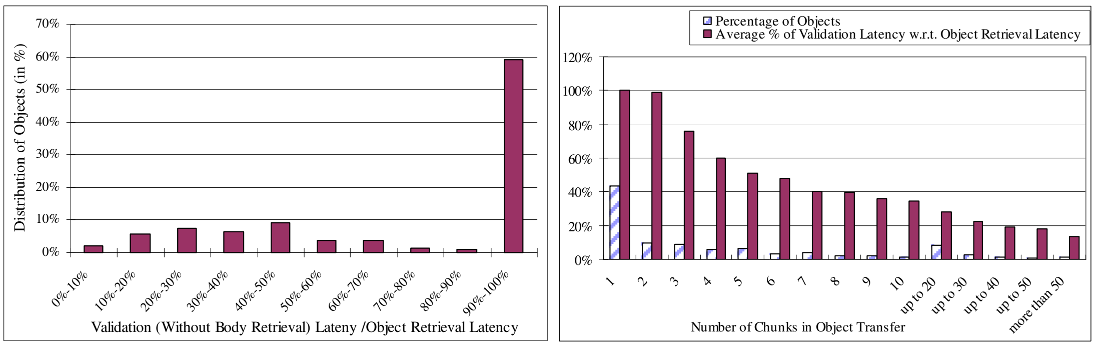
The outline for the rest of the chapter is as follows: we start by justifying our study of TTL adaptation mechanism through research on existing re-validation situation. After presenting our concept on “Two-Steps” TTL adaptation model, we use live experiment to verify the model’s practicability and feasibility.

6.2 Re-Validation with HTTP Response Code `304`: Cheap or Expensive?

Before we go into the study of our TTL adaptation mechanism of web objects, we would like to justify the problem statement of our research here. Is it important to study the TTL predication of web objects, given that there is a re-validation mechanism supported by the HTTP protocol? When the object content for a web request is found in the local client/proxy cache and its expire-time has already passed, a conditional `GET` request (typically, this is an `If-Modified-Since (IMS) GET` request) can be sent to the content server. The object body is returned to the client/proxy only if the local cached copy is different from the master copy in the content server.

It is true that under the situation where the local copy of the object is still fresh, the reply of such conditional `GET` request does not consume a lot of bandwidth. However, the cost, in terms of the latency, is actually quite high. Some of the previous work on web caching consistency did preliminary analysis on the proxy traces to find out the relative statistical magnitudes of the service latency for the HTTP response code `200` and `304`. But the result of their work is too limited to give insight about the cost of object validation without body fetching because their requests for the response code `200` and `304` might not refer to the same object or study under the same network/server workload.

To gain a deeper understanding on the cost of validation without body fetching (i.e. HTTP `304` response code `Not-Modified`), we took traces from the NLANR ftp site [NLANR], extracted the URL names, and repeated the actual retrieval of objects. The total number of objects tested is 1,013,335. The network to conduct the experiment is the high speed Singapore Advanced Research Network. What we would like to measure is the time for the first byte to arrive at the client side and that for the whole object to be retrieved. Here, we approximate this first byte retrieval time to be the latency of the HTTP response code `304` (`Not-Modified`) and the whole object retrieval time to the latency of the HTTP response code `200` (`GET`). This approximation is quite reasonable. One additional advantage of this approach is that the bandwidth and server I/O allocation for the two responses are guaranteed to be the same. Note that issuing two web requests (one for response code `200` and the other for code `304`) even within a short period of time might still suffer from the fluctuation of the network and server workload and this might result in inconsistent measurement. Figure 6.1 (a) and (b) show the result.



(a) (b)

Figure 6.1: Normalized Validation Time w.r.t. Retrieval Latency of Web Objects

Figure 6.1(a) shows the distribution of the normalized validation time of object with respect to its entire object retrieval latency. This clearly shows that the validation time is actually of similar order as the entire object retrieval time. About 59.25% of the web objects show that the validation time is similar to the entire object retrieval time. And only 2.09% of the web objects have validation time that is less than 10% of the entire object retrieval time. This is expected because most of the objects on Internet are small.

Since data are sent through Internet in a chunk by chunk manner and the typical chunk size is about 1.1Kbytes to 1.3 Kbytes, this explains the huge percentage of objects with validation time being close to the object retrieval time. Only when the objects are very large (e.g. JPEG images), then the normalized validation time becomes insignificant. However, there is a counter-argument to this situation. Most large objects are images and they are found to be quite static. It is quite unlikely that they need frequent re-validation.

Figure 6.1(b) plots the distribution of the normalized validation time and the object frequency distribution against the number of data chunks found in the actual content transfer of a web object. The figure supports our observation and argument in Figure 6.1(a). There are about 43.5% of the objects that are transferred in one single data chunk and only 14.07% of the objects are transferred in more than 10 data chunks. Furthermore, the relatively small data transfer time for chunk other than the first one, as compared to the network setup time, can be reflected by the 100%, 98.63% and 76.25% of the normalized validation time for object transfer with one chunk, two chunks, and three chunks respectively.

All these results highlight the importance of understanding (and hopefully predicting) the TTL of web objects and show the high cost of content re-validation on the fly. Prediction accuracy is important here because aggressive caching with too large estimated content TTL might result in the use of outdated data whereas conservative caching with too small estimated TTL might result in unnecessary bandwidth consumption and high retrieval delay. Similar arguments apply to web information systems as well.

6.3 Two-Steps TTL Adaptation Model

In this section, we would like to propose a two-steps TTL adaptation mechanism for efficient and accurate TTL prediction of a web object. The two steps are the **stochastic predictability** and the **adaptive updating**. In the stochastic predictability process, we will first capture the probability distribution function (PDF) of the life-span using stochastic process such as Gamma or WeiBull modeling. Based on the pattern of the distribution, the regularity of the life-span for TTL value prediction as well as its expectation value can be determined. In the adaptive updating process, correlation pattern recognition model will be used to determine any updating of the PDF of the life-span for future prediction.

Given an information source on the web, it is not difficult to imagine that its content updating will be both regular and irregular. To attract audience to (re-)visit a website, the content provider needs to constantly provide the latest information on the web. Some good examples are the newspaper and online banking/stock information. Hence, certain kind of content updating schedule can be expected and it is usually quite predictable.

However, if the study period of life-span of a web object is long enough, it is not difficult to find that a single PDF is far from enough to accurately describe the change pattern of its content. In fact, multiple PDFs are actually involved and the one being used depends on the actual work schedule. For example, during the work hours of the working day, content such as CNN news portal might be updated every fifteen minutes. However, during the lunch time, night time, or weekend, the actual content updating might be slower (with different PDFs) and sometimes even stop. Another factor that complicates the situation further is the continuously changing to the requirement for content updating to meet the client's need. Furthermore, there are situations where irregular, sudden change to the content is made. All these suggest the need to check the validity of PDF defined by previous content updating history for future life-span prediction. Whenever necessary, the PDF should be automatically adapted to reflect the new situation.

To achieve this goal, we first assume that objects might be changed according to some kind of regular pattern for a given period of time. For example, if a program updates an object n fixed periods in one day, the regularity of content change for this object will be n per day. And this regularity can be captured by stochastic process such as Gamma distribution model. After that, the correlation pattern recognition model can be used to check whether the regularity is stable and valid with respect to time. If the regularity is found to be changed, the model will capture the change and adaptively update the parameters for the new PDF. This makes the prediction more accurate because it caters for changes to the regularity, which indeed happens in actual situations.

6.3.1 Content Creation and Modification

Before we go into the details of our two-steps TTL adaptation mechanism, it will be helpful to have some basic understanding on the content creation process for the web and the factors affecting its modification process.

Despite the wide variation of applications on Internet, the primary usage of the web is still for information publishing and dissemination. Thus, a URL address can be viewed as a pointer to an information source and a web request with the URL address is to get the content data for presentation at the client end. From the viewpoint of a client, a web server functions as a content presentation generator $f_{Content\_Pres\_Gen}$. It takes inputs from the client and network/server workloads. Together with predefined rules and data set in the web server, $f_{\mathrm{Content\_Pres\_Gen}}$ will generate the final content presentation to be replied back to the client. Sometimes, this content generation process might invoke application execution (such as database access) in the web server.

The content presentation Content_Pres for a given web object with address URL can be represented as follows:

Content_Pres = $f_{\mathrm{Content\_Pre\_Gen}}$(DS, $\vec{I}_C$, $\vec{I}_R, \vec{I}_S, \vec{I}_D$, T)

Each input to $f_{\mathrm{Content\_Pres\_Gen}}$ can be viewed as a class of input with shared common features to the content generation process.

Data Source DS:

This refers to the data content source pointed to by the web address URL. Note that the final presentation of a web object might not be the same as its content source because content optimization, adaptation, and personalization that are based on the rest of the parameters might take place.

Client Profile Vector $\vec{I}_C$:

This vector $\vec{I}_C$ includes all inputs related to the client's personal profile. Examples are the client's IP address, browser type, language preference, browser type, and cookie's value.

Request Input Vector $\overrightarrow{I_R}$:

It refers to the information related to the intrinsic properties of the given request URL such as the request method and object relationship with the referral. It might also include parameters associated with the request such as query parameters, access authentication input, and post data in the request body. Request preferences such as the acceptance, encoding, caching and revalidation, and security all belong to this category.

Server Setting Vector $\vec{I_S}$:

This includes server setting properties such as the message encoding method, file system and web resource ACLs. Sometimes, server status, such as the system resource usage and access history, might also cause different content presentations to be generated.

Application Source Vector $\vec{I_D}$:

This vector describes all the applications involved in creating the final presentation (as the program output) of the request. Typical applications executed for a request are database access and encryption engine.

Request Time T:

This variable specifies the time when the client makes such request with address URL.

As we can see, the life-span and the TTL of a web object are affected by multi-factors instead of a single one.

6.3.2 Stochastic Predictability Process

Many stochastic distributions of real life periods such as the life period of electric bulbs and batteries can be captured by Gamma distribution. The gamma distribution contains the exponential and Erlang distribution as special cases, and converges to a normal distribution as α_i goes to infinity. Therefore, it is natural to approximate the change interval time distribution of an object with a Gamma distribution. From the Gamma model, we can estimate when the next change of a given object will come and determine whether the frequency of change will increase, decrease or stay the same in the near future.

The Gamma Distribution Model [48] is defined as follows (Its mathematical definition can be found in the appendix of this thesis):

$$Gamma(\Delta t_{ij} : \alpha_{i,} \beta_i) : \ f(\Delta t, \alpha, \beta) = \frac{(\Delta t - \mu)^{\alpha - 1} e^{-(\Delta t - \mu)/\beta}}{\beta^{\alpha} \Gamma(\alpha)}$$

where $\Gamma(\alpha) = \int_0^{\infty} \Delta t^{\alpha - 1} e^{-\Delta t} \partial \Delta t$

αi is the shape parameter: If αi equals to 1, the Gamma distribution will recess to exponential distribution. If αi goes to infinite, Gamma distribution will recess to normal distribution. βi is the scale parameter: Its value is to reflect the size of unit in which every change interval is measured when compared to the standard Gamma distribution. μi is the location parameter. The Gamma distribution shifts to the right by the smallest possible change interval time. Hence, it can also be deemed as a safe period where the object will not change.

Content providers usually have a content updating schedule to achieve a compromise among the changing process of information sources, the workload of content updating, and the popularity of the site. Generally, the content providers have two methods to update the content. The first one is usually done using some program to do automatic content uploading to the site at relatively fixed time intervals. This happens despite the various factors affecting the content updating plan. As a result, the next change interval of a web object tends to be inherent to its previous change interval history and this change is quite stable. This is the rationale behind the use of stochastic process to describe it.

Note that in our study, we use Gamma distribution to describe the stochastic process of content change. However, this does not limit our mechanism, result, and contribution if other distribution such as WeiBull is used instead. Our focus of research here is to investigate the validity of the distribution and provide the adaptability of PDF to the system rather than to claim a particular stochastic process function.

6.3.3 Correlation Pattern Recognition Model

Even though the regularity of change of an object is observed to be stable within some time period, the factors described in Section 6.3.1 might change suddenly or be interfered occasionally. For example, the persistent connection in $\vec{I}_C$ suddenly terminates because of network congestion; server system resource in $\vec{I}_S$ is insufficient to execute the updating program in time because of heavy workload. This results in the change interval of such cases being different from previous change intervals. Hence, given a sufficiently long period, the discrepancy in the outcome of factors should be tracked all the time and the regularity of change needs to be verified and updated whenever necessary. This is to make the prediction of an object's TTL as close to the actual situation of change as possible. As a result, we need a method to measure the prediction correctness and to replace the current predicted regularity of change if the difference between the predicted situation and the actual one is greater than certain pre-defined threshold value.

With the prediction algorithm for content change, we need to identify the persistent regularity of change of an object. The object’s initial prediction reference is obtained from the Gamma distribution curve, where the data is obtained from the most recent actual change intervals recorded in the log file. Taking two periods, and using their Gamma curves, we can then determine whether their regularity is persistent by determining their similarity. This is basically in the form of correlation pattern recognition process.

The basis of TTL prediction algorithm is that since the regularity of change of objects is relatively stable within a time period, the regularity should be able to extend to the near future. And this needs to be continuously proven, and updates accordingly if the regularity of change is modified.

Correlation pattern recognition [49][50][51] is unique in that it provides a methodology to compare two groups of information with multi-dimensional data. It works based on the judgment of the correlation coefficient, which is defined as a statistical measure to quantify the "goodness-of-fit" in many curve-fitting procedures [52]. Here we use it as an indicator of fit, or similarity, between the Gamma distribution curves generated from two successive periods of content change interval data. Note that in our measurement, all segments in these curves have the same length.

The calculation method for correlation coefficient is defined as follows:

Given two groups of intervals: series X of length N as $[X_1, X_2, \dots, X_N]$ and series Y of length N as $[Y_1, Y_2, \dots, Y_N]$. X represents the Gamma distribution of content change intervals in a time period t, and Y represents that in the following period t+1.

The series X has a mean E(X) , given by the average of its values

$E(X) = (X_1 + \dots + X_N) / N.$

The mean is a measure of how far X is displaced from zero. Series X also has a variance, V(X), given by:

$V(X) = (X_{12} + \dots + X_{N2}) / N - (E(X))^2$

Mathematically, the variance of X is a measure of its size, after its mean is removed. For series Y, also of length N, the covariance between X and Y is defined by:

$COV(X,Y) = (X_1Y_1 + \dots + X_NY_N) / N - E(X)E(Y)$

The covariance provides a quantitative measure for the similarity between the two series X and Y. It will reach its maximum when Y and X are the same. To remove the effect of the sizes of the two series, the covariance should be normalized by dividing out their standard deviations, which are the square roots of their variances:

Correlation Coefficient = COV(X,Y) / SQRT[V(X)V(Y)]

The value of correlation coefficient ranges between 1.0 and -1.0, where the value of 1 indicates a perfect match (i.e. the two patterns are identical) and the value of -1 would indicate that an exact match cannot be found but that it is "upside-down". Values near zero mean there is no match at all.

In practice, it has generally been agreed that values greater than or equal to 0.8 correspond to patterns in the data that are easily discerned as "good matches" by human eye. The correlation coefficient is also a normalized statistical measure, which means that the actual numerical values of either the chart data points or pattern values have no effect on its value. Only the "shapes" of the pattern and chart segments affect it. All of these features make the correlation coefficient a good choice as an indicator of pattern matching.

In our case, if the correlation coefficient calculated from the Gamma distribution curves of two continuous periods is greater than or equal to 0.8, the object's regularity of change should be considered as stable. It is then reasonable to use TTL value(s) obtained from the previous period in the latter period. On the contrary, if the coefficient is less than 0.8, it means the object's regularity of change is different enough in the two time periods of measurement. In this case, using the TTL value from the old regularity as a reference in the new regularity period will be undesirable. If the object's freshness period obtained from the period N-1 cannot be used as a prediction result in the period N, the algorithm should use the freshness period of period N as the next prediction value for the next period (N+1).

6.4 Experimental Result

In this section, we would like to illustrate the usefulness of our TTL adaptation mechanism through real-time monitoring of content change in popular websites found on Internet. Firstly, we describe the experimental environment and setup. Then we classify websites into four different types according to the characteristics found in the PDF of their life-span. After that, we present the result obtained from the TTL behavior stage and prediction stage of our mechanism. We also compare and analyze the results obtained from the actual situation, existing algorithms/methodologies, and our approach. The result shows that our algorithm can effectively adjust an objects' TTL due to the change in content updating regularity.

6.4.1 Experimental Environment and Setup

Our experiments were performed on a Pentium II 266 CPU machine, with 96M RAM and a SCSI 9GB hard disk. The TTL adaptation mechanism (described in Section 3) was implemented in this machine, and it was used as a reverse proxy for monitoring the content change. A total of 25 most popular websites were chosen from 100.hot [46] for our study. They represent some of the most famous websites in their corresponding industries. Examples of these sites include ebay, cnn, and yahoo. The monitoring period lasted for six weeks, with two weeks in August and four weeks in September, 2002.

6.4.2 PDF Classification

There are two stages in our monitoring and prediction study: (i) TTL behavior stage to learn the change regularity of an object's life-span PDF, and (ii) TTL prediction stage with adaptive updating of the change regularity of the life-span PDF.

Before we go into the details of the two stages, we would like to classify the life-span PDF of objects and identify the interesting cases that most websites are likely to fall into. In our TTL adaptation mechanism, we will first decide whether an object has any potential for prediction. In our experiment, we use the Gamma distribution model to make this decision. And all the objects under study will be classified into either TTL unpredictable objects or TTL predictable objects.

For TTL unpredictable objects, their change intervals distribution can further be grouped into two sub-types. The first type is completely random and chaotic. Its distribution curve has no apparent peak, meaning that no interval aggregates in some time interval ranges and no regularity can be tracked. The second type is an exponential distribution, which means that its body might be dynamically generated and is different for every access.

For TTL predictable objects, we observe four sub-types of change intervals distributions in our experiment:

(1) Type 1: There is only one apparent peak, which suggests the object change mainly to be fitted in a fixed time interval.

(2) Type 2: It is an exponential distribution, which means that its body can only be guaranteed to be fresh in one minute. This type should be differentiated from the second type of unpredictable objects by the existence of some server's hint

in the HTTP header during object retrieval. For example, the existence of the header field “Cache-control: max-age=60”.

(3) Type 3: There are multiple peaks in the distribution curve, which means that the object might be updated in different fixed periods. For example, it is updated every one hour from 9 am to 9 pm, but it is updated every 2 hours from 9:01 pm to 8:59 am of the following day.

(4) Type 4: This is a combined exponential distribution with one or multiple peak distribution. It represents that either the object is updated in one or more fixed periods or the object might be dynamically generated and is different in every access. The reason why it still belongs to the same predictable type is that the object still has some change regularity. For example, one object is updated every 15 or 14 minutes, but it will dynamically generate different temporal advertisements in the first minute of every hour. Thus, this first minute will not be suitable to set any TTL.

	Unpredictable object		**Predictable object**			
Distribution Type	Type 1: random and choas	Type 2: exponential	Type 1: 1 peak	Type 2: exponential	Type 3: multiple peaks	Type 4: Combination of exponential and multiple peaks
Percentage	0	20%	28%	32%	8%	12%

Table 6.1 Percentages of Different Change Regularities

Table 6.1 shows the distribution of objects according to this PDF classification. It shows that 20% of the objects under study belong to the second type of unpredictable object, which means these objects might be all generated by program automatically. Due to the security and privacy reasons, they will be different in every access. Furthermore, in our study period, their change intervals are unpredictable.

80% of the objects under study belong to the predictable objects. This observation is expected because recent website design tends to change the structure and layout of the website more often in order to attract more customers, thus resulting in shorter life-span. Consequently, proper TTL prediction is important for credible web caching.

Since predictable objects are valuable in our study, we will use the rest of the chapter to discuss their behavior. To make the analysis easier, representative URL for each of the four sub-types will be used in our discussion. Their URLs (the numbers correspond to their types) are:

(1) http://www.asia1.com.sg

(2) http://www.cnn.com/WORLD/index.html

(3) http://www.whitehouse.gov/index.html

(4) http://www.ebay.com/index.html

6.4.3 TTL Behavior Stage

In this stage, an object's change regularity is captured by its Gamma distribution curve, which is drawn according to the probability distribution of its real change intervals. This is done as follows. In our monitoring process, after the time interval space of content change for each object is found, we will calculate the probability of real change interval and a data point will be obtained. A line will then be used to join these real change probability points and the result is the actual probability distribution curve Based on this line, a proper set of Gamma distribution parameters can be obtained through curve fitting to this line. This curve will represent the predicted object's change regularity in this period.

Figure 6.2 gives our result in the object's TTL behavior stage. It shows the content change regularity for the four representative URLs. In the figure, x-axis represents the data groups of real change intervals, and y-axis represents the probability of real change intervals. (Note that the meanings of the x-axis and y-axis are the same for all the figures in the rest of this chapter). "Real change interval" is the period between two successive change times of an object obtained by comparing its actual body. "Probability of real change intervals" is the percentage that an object is changed at the indicated period of time. "Object's change regularities" can be obtained from the "actual probability distribution line". Each data group range in the graphs in Figure 6.2 is less than five minutes, with most being one minute. The period of study for these four curves is one week, from Aug 12, 2002 to Aug 18, 2002.

The Gamma model is aimed to grasp an object's change regularity. It will ignore those unimpressed trends or data and only focus on the main trend(s). Its function generates results based on this rule. For example, for URL (1) – asia1, its Gamma distribution curve and function generated as follows. The TTL values in data group 8 takes up more than 50 percent out of all possible TTL values, and other values are very scattered along the x-axis. Thus the model will keep track on this main trend, and ignore other insignificant ones, which can be rephrased as setting those small enough data to zero. Data groups 7, 8, 9 and 10 will be used to generate proper values for $\alpha = 40.2658$ and $\beta = 0.0726$. The steeper and narrower the peak, the bigger is the value of α and smaller is the value of β. Furthermore, the y-axis is shifted to data group 7, indicating the location parameter $\mu = 11$, and this is the time interval of the beginning value of data group 7. Since data group 7, 8, 9, and 10 only takes 78.63% of all the possible values, the corresponding Gamma function should multiply this weight.

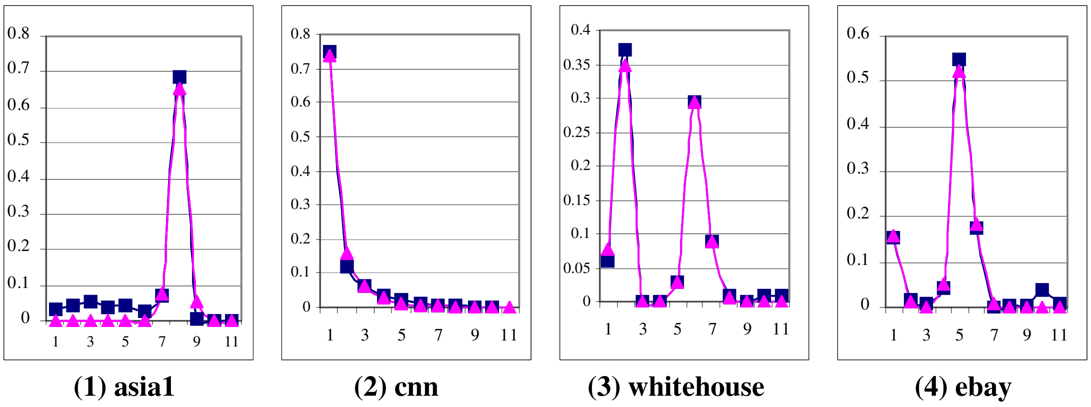


Figure 6.2 Gamma and Actual PDFs for Content Change Regularity
(*x-axis represents the data groups of real content change intervals, y-axis represents the probability of real change intervals. The line with quadrangle points is the line of actual intervals, and the line with triangle points is the fitting Gamma distribution curve.*)

Figure 6.2 confirms the argument that the Gamma function can capture the main trend in the actual probability line very well. The Gamma functions used to fit the actual probability lines in Figure 6.2 are as follows:

(1) F(x) = 0.7863(40.2658,0.0726,11)

(2) F(x) = (0.2857,2.09,1)

(3) F(x) = 0.4455(40.5,0.099,1) + 0.4216(13.3525,0.1607,40)

(4) F(x) = 0.1690(0.295,1.65,1) + 0.7651(29.6072,0.1079,5)

For (1) – asia1, we can deduce the change regularity for asia1; it is usually updated every 15 minutes. As a result, in this TTL behavior stage, its expected TTL value is 15 minutes.

With regards to the actual probability distribution line in (2) - cnn, we observe from Figure 6.2 that it is an exponential distribution. Therefore, α must be less than 1, and all the data group values can be taken into consideration when generating the Gamma distribution fitting curve. From the curve, we can deduce that cnn changes frequently. It can be guaranteed to be fresh only within one minute even though sometimes its updating period is more than that. Its expected TTL is still one minute.

For (3) - whitehouse, there are two main trends (two peaks) from the actual probability distribution line. Thus the Gamma model needs to use two functions to represent the case, each one of which trying to capture one trend. The two functions are combined together by adding them together. From the curve and our other experimental result, we can deduce that whitehouse has two updating periods: one is during the US daytime and it is 15 minutes, and the other is during the US nighttime and it is 60 minutes. Therefore its expected TTL is 15 minutes in the US daytime, and 60 minutes in the US nighttime.

For (4) - ebay, there are many advertisements on this website. Sometimes, the trend for its content change might be unpredictable. At the other times, however, it can also be predictable. As a result, its Gamma distribution curve is a combined exponential distribution and normal distribution. With the help from our other experimental data, we find that the website updates only with a small part of the content in about every 10 minutes. However, in the first minute of every hour, it will update most of the content. Due to this major update, different user accesses to this site will result in different updated or non-updated parts with different temporary advertisements. Thus, its expected TTL is 0 for requests in the first minute of an hour, and 9 or 10 minutes for other requests.

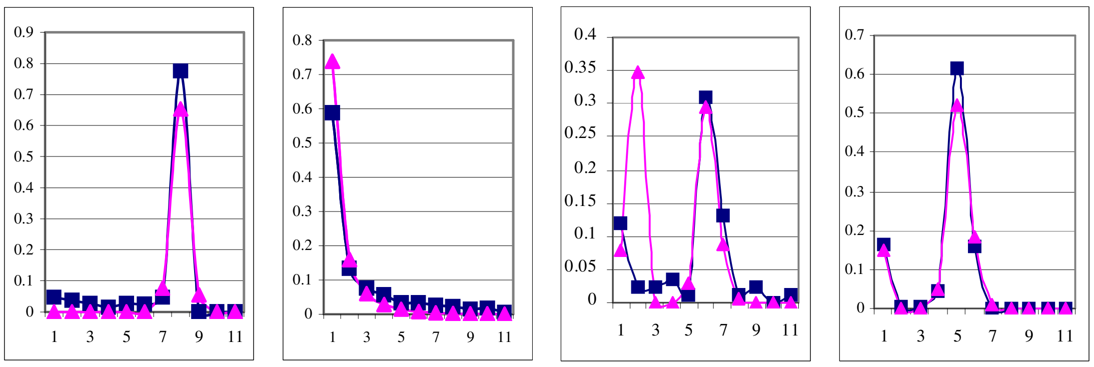
6.4.4 TTL Prediction Stage

After we obtain the expected TTL from the study of object's change regularity in the TTL behavior stage, we would like to use it as the prediction value in the subsequent periods as much as possible. And our continuous monitoring is aimed to determine whether it is accurate to use the previous expected TTL as the future prediction value, and if necessary, to adaptively adjust the value according to the actual regularity replacement. Such detection and adjustment are all based on the correlation coefficient theory as we described in Section 6.3.3.

To make our discussion easier, we sub-divide this prediction into two parts: adaptive learning and stable result prediction.

6.4.4.1 Adaptive Learning

With respect to the weekly data, when the correlation coefficient is less than 0.8, which means the change regularities in the two weeks are different, the future prediction value should be adaptively adjusted according to the latest change regularity. This is to lessen mistakes in the future prediction. Figure 6.3 and 6.4 show the result of this case.



(1) asia1 **(2) cnn** **(3) whitehouse** **(4) ebay**

Figure 6.3 Gamma Distribution Curve from Aug 12 to Aug 18 vs Actual Probabilities Distribution Line from Aug 19 to Aug 25

The correlation coefficient calculated from Figure 6.3 shows that for the website of whitehouse, it is not appropriate to use the change regularity of the previous week (Aug 12 to Aug 18) to predict the following week's (Aug 19 to Aug 25) change situation. This is due to the following week's change situation being substantially different from the previous week's regularity. In the previous weeks, there are two update time intervals every day for the website of (3) - whitehouse. However, there is only one update time interval in the following week. Thus, the predicted basic parameter set needs to be adjusted for this website. The other three websites' correlation coefficients are still bigger than 0.8, meaning that their change regularities are still the same as those of the previous two continuous weeks.

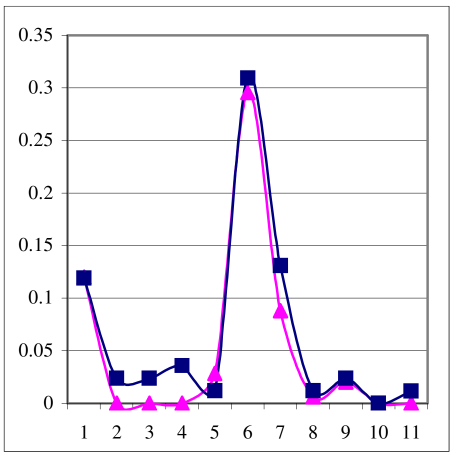
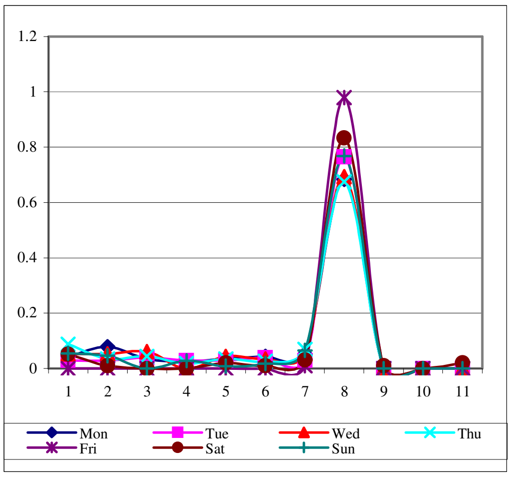


Figure 6.4 Re-learning the Change Regularity for (3) - whitehouse from Aug 19 to Aug 25

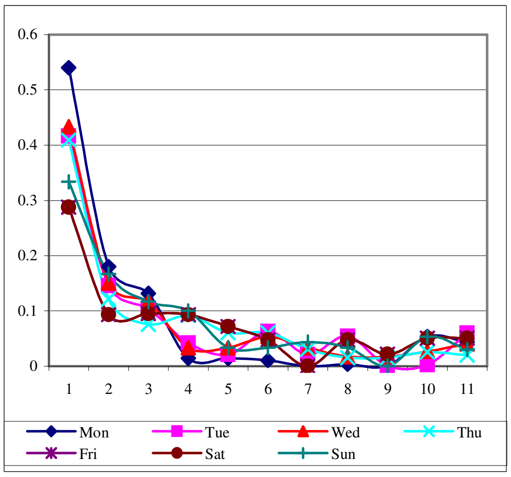
For (3) - whitehouse, it change regularity needs to be re-learnt to obtain proper prediction value. The adjustment will be invoked by generating a new Gamma distribution curve to fit this week's real change intervals line (from Aug 19 to Aug 25). Figure 6.4 shows the curve-fitting situation in the re-learning process. The prediction value for (3) - whitehouse will be adjusted to 60 minutes.

6.4.4.2 Stable Result Prediction

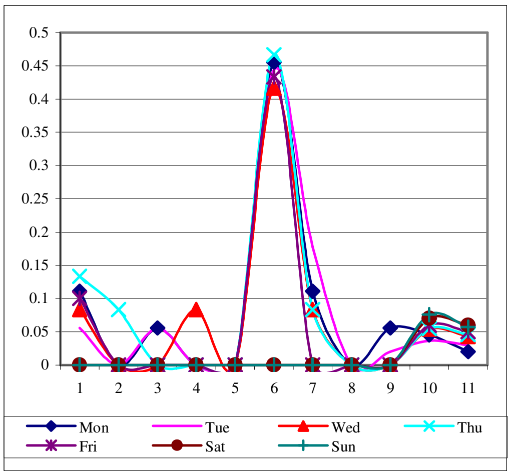
From our experiment, we found that overall speaking, no matter whether the prediction is a daily or weekly process, most objects' change regularity are quite stable within the whole September period of our study. This can be seen from the comparison among the actual probability lines in Figure 6.5, which are daily based, and in Figure 6.6, which are weekly based.



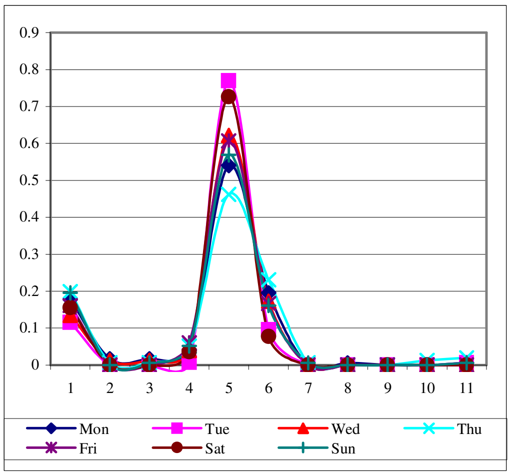
(1) asia1



(2) cnn



(3) whitehouse

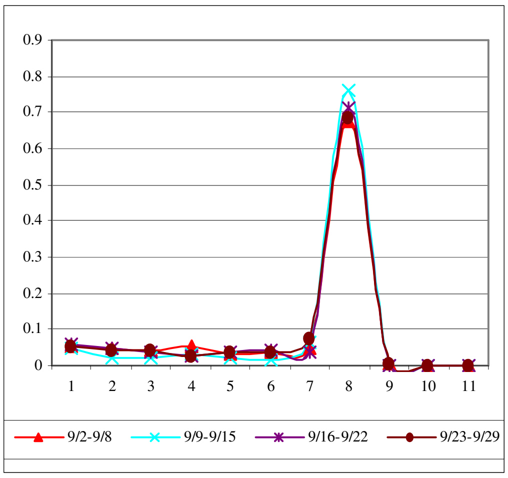


(4) ebay

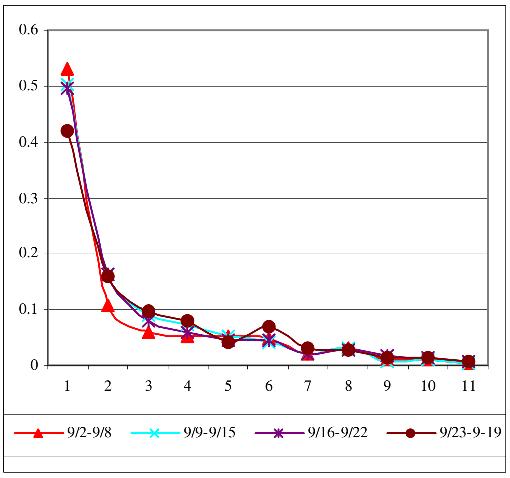
Figure 6.5 Probability Distribution with Daily Real Change Intervals

Figure 6.5 shows the probability distribution with daily change intervals in one week, from Aug 19 to Aug 25. One line represents one day's distribution situation. The group ranges of every line in the same graph are the same. It seems that the object's daily real change lines are similar in the week. In addition, our experimental data for the daily distribution in the subsequent five weeks also show that the daily distribution line is similar to each other in the same week.

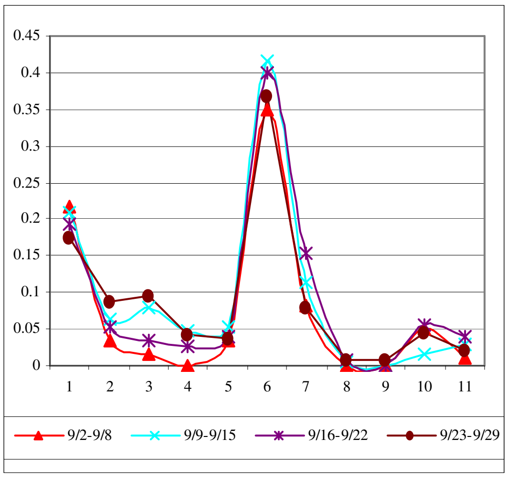
According to the correlation theory, the correlation coefficients for the Gamma distribution curves of daily change regularity in the same week are all bigger then 0.8. This implies that the change regularity is similar in every day of the same week. Thus the expected life-span period from the first day can be used as the prediction value for the following days of the same week.



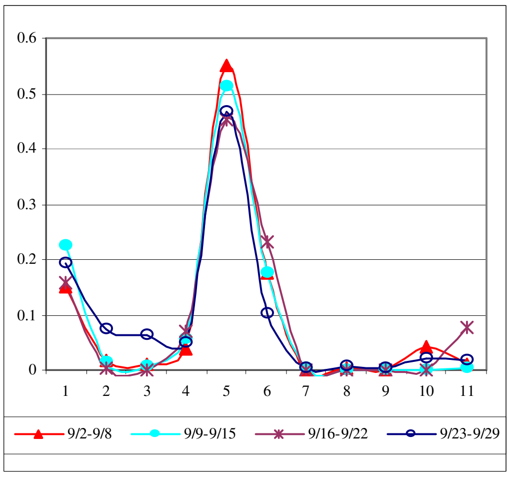
(1) asia1



(2) cnn



(3) whitehouse



(4) ebay

Figure 6.6 Probability Distribution with Weekly's Real Change Intervals

Figure 6.6 shows the probability distribution for weekly change intervals in the successive weeks, from Sep 2 to Sep 29. One line represents one week's distribution situation. Once again, the weekly change regularity is quite stable in this month. The results shown in Figure 6.5 and Figure 6.6 suggest that the objects' change regularity can be daily or weekly based in our four weeks of experimental period.

Comparing the results in Figure 6.5 and Figure 6.6, it seems that the similarity obtained weekly is better than that from daily. Thus, we use this for our next stage of content change monitoring - using the change regularity from the first week of September to predict the change regularity of the following three weeks of September.

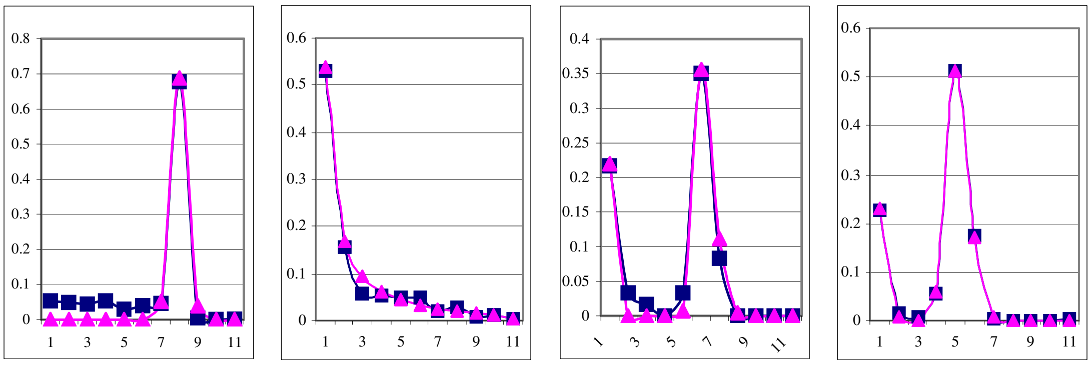
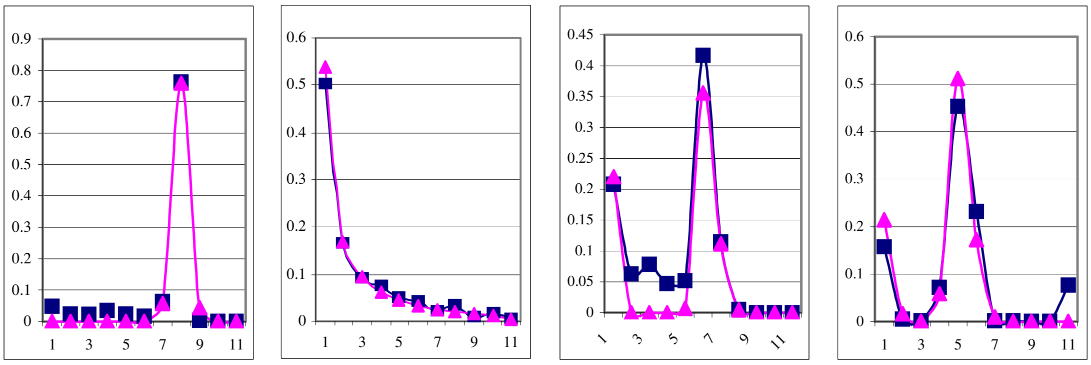
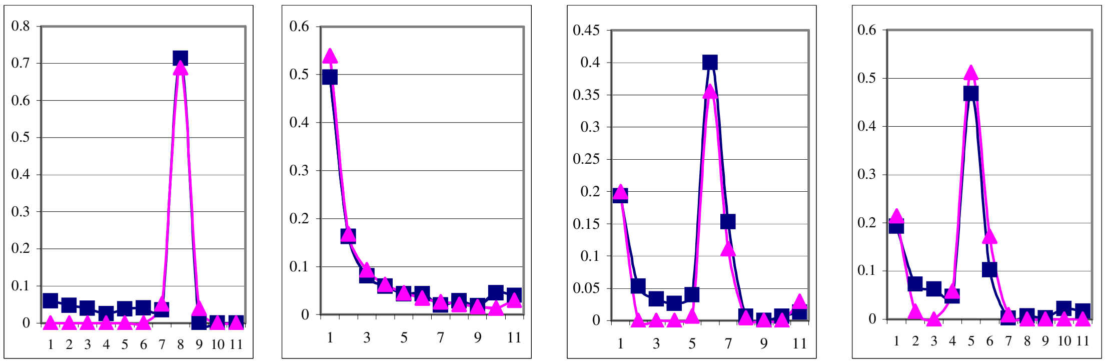


Figure 6.7 Learning Process for Capturing the Change Regularity from Sep 2 to Sept 8



(a) Week 1


(b) Week2

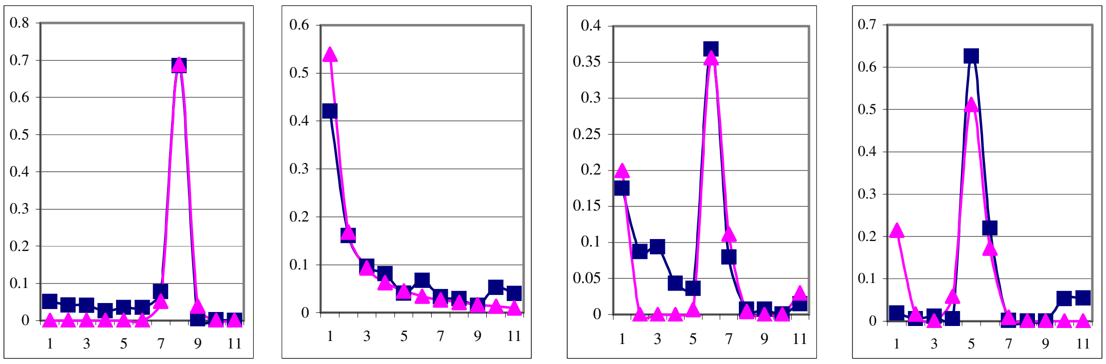

(C) Week 3

Figure 6.8 Predicted Result from Sep 9 to Sep 29 Based on Learning Result in Sep2 to Sep 8

Figure 6.7 shows the behavior stage in the first week of September and Figure 6.8 shows the prediction monitoring stage in the following three weeks. Every row of curves represents the prediction situation of the week. After obtaining the predicted value from the first week, the correlation coefficients calculated for the following three weeks show that the objects' change regularities are stable in this month. Therefore, our prediction effect based on first week representational values is quite good for the month of September.

6.4.5 Result Analysis and Comparison with Existing Solutions

In the last section, we already showed that our approach of using Gamma distribution model as our basic reference is effective to capture an object's change regularity for future value prediction. With the correlation pattern recognition model, we also show that either the validity of using previously obtained result can be justified or new prediction references can be obtained through proper adjustment.

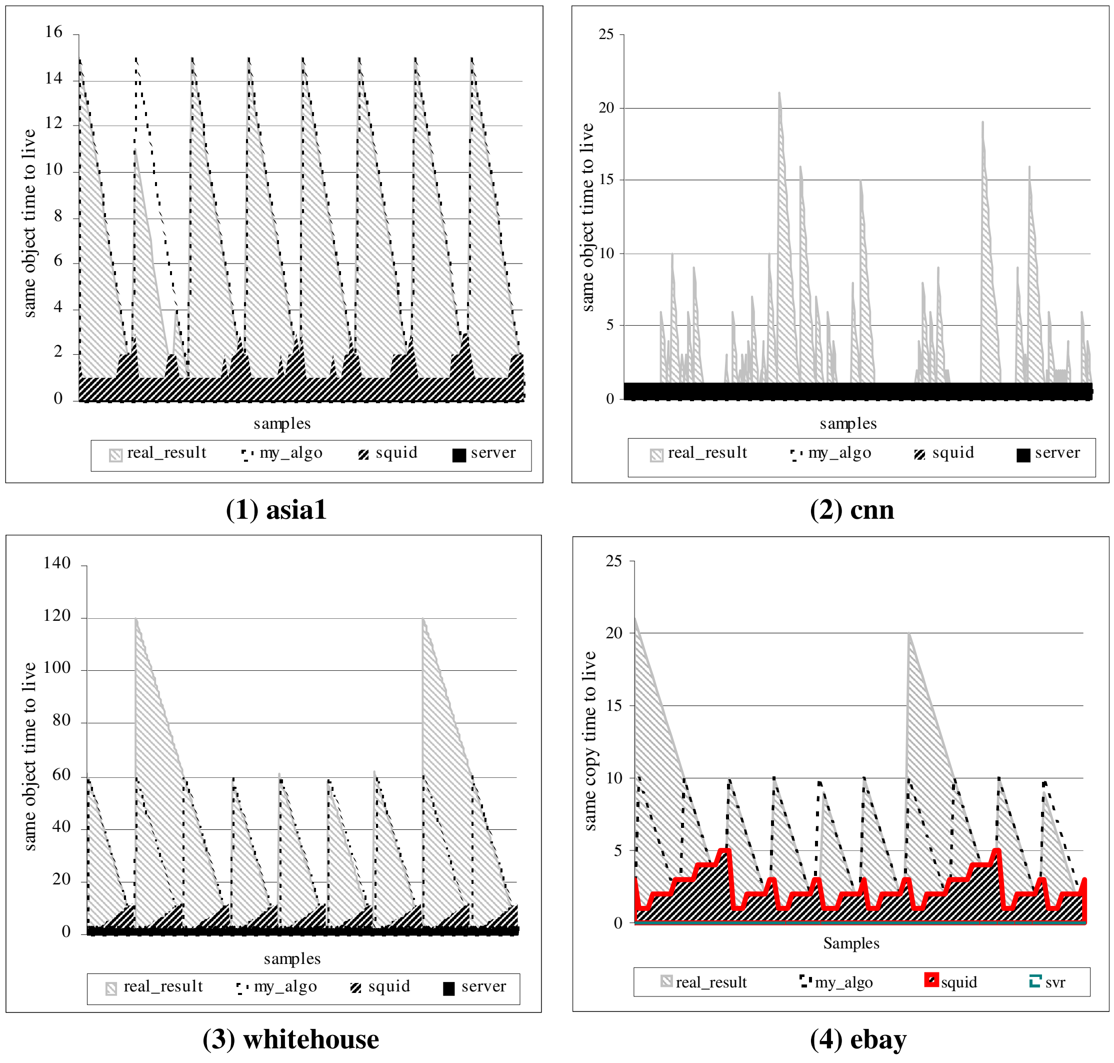


Figure 6.9 Comparison of our Prediction Results with Those from Actual Situation, Squid's Algorithm and Server Directives

.

In this section, we would like to further our study by comparing our predicted results with those from the Squid's algorithm, server directives and actual situation. Squid's algorithm is included in our study because it is the most popular proxy cache currently being deployed and it has rules to determine an object's TTL. We are also interested in the server directives because we want to find out how accurate (or efficient) the content providers can predict TTLs for their content. The result is shown in Figure 6.9. We use the values deduced from Sept 2 to Sept 8 to predict the life periods from Sep 9 to Sep 29. The figure also shows the comparison situation in the latter period.

In the figure, fill-in grey blocks are the results of the actual change intervals (In fact, it should be a line of the time interval points. To compare with other results easily, we fill in the area below the line). The broken lines are results of our algorithm. The bold lines are results of Squid’s algorithm. The normal lines are results calculated from the server directives. The x-axis represents the request times, and the y-axis represents the remaining object’s life period. For example, with reference to the actual situation, when ebay was requested for the first time, we observed that this body changed after 20 minutes. That is, when the value along the x-axis is 1, the remaining life period along the y-axis is 20. When the value along the x-axis is 2, the y value (the remaining life period along the y-axis) will become 19, and other values can be analogically deduced. Therefore, the closer and lower the other three lines to the edge of the actual block, the better will be the prediction. It is because this means the prediction result is close to the actual change situation.

From (1)(2)(4) graph in figure 6.9, we can see that many of our prediction results just overlap with the result of the actual situation, which means these results are exactly correct in those periods. But some are quite lower than the edge of the actual block, which means we still need to waste some bandwidth and do useless work (redundant validation) on the same fresh objects. Some lines are even higher than the blocks. These are prediction mistakes and will result in returning stale objects to clients in those periods.

Analyzing the Squid prediction results, most of its predictions are much lower than the edge of actual block, which means Squid will perform a lot of useless work and result in large amounts of network traffic and server burden. However, at most of the end of the triangle blocks, which means the object will change soon, Squid's prediction results' line are always higher than the blocks. This phenomenon is because Squid's algorithm to predict objects' TTL is only based on the assumption that the longer the object did not change, the lesser possibility for it to change, and without any detection process to verify this kind of assumption. The period indicates the difference between Last-Modified timestamp and current timestamp.

Furthermore, its assumption seems to be incorrect from our experiment. For example, ebay's first version in our experiment has a 20 minutes lifetime. When it was been requested in 19th minute, it will have just a minute of lifetime left. However, Squid will assume that since it has lived so long, it can live even longer. Its prediction of lifetime for the body retrieved by the request at the 19th minute will be 5 minutes. Therefore, this heuristic prediction will generate many mistakes, and will increase the chance to send stale objects to clients.

Analyzing the servers' directives, most of them are too conservative. Some do not even have an indication of the lifetime period of an object, which results in objects being uncacheable, or being cached too short, thus resulting in unnecessary network traffic.

In graph (3) of Figure 6.9, our prediction results are the same as the results from Squid and server directives. Most of cnn objects' lifetime is 1 min, others is chaotic.

The statistic average comparison for these three kinds of results with the actual situation is shown in Table 6.2. The predicted values are obtained from the first week representational values. The actual situation is in the following three weeks in September.

	(1)Asia1	(2)cnn	(3)whitehouse	(4)ebay
	Percentage of less than actual TTL prediction			
My algorithm	1.32%	83.83%	23.33%	43.93%
Squid's algorithm	77.72%	83.83%	80.37%	76.48%
Server directives	100%	83.83%	100%	100%
	Percentage of equal to actual TTL prediction (accuracy)			
My algorithm	94.2%	16.17%	60.88%	43.04%
Squid's algorithm	10.13%	16.17%	2.28%	8.63%
Server directives	0	16.17%	0	0
	Percentage of greater than actual TTL prediction (mistake)			
My algorithm	4.48%	0	15.79%	13.03%
Squid's algorithm	12.15%	0	17.35%	14.89%
Server directives	0	0	0	0

Table 6.2 Comparison from the Results of My Algorithm, Squid's Algorithm and Server Directives with the Actual Situation

The table shows that most of our algorithms' results are better than Squid's and the server's results. Conservativeness in server directives and the Squid's estimate will result in unnecessary bandwidth usage, increased network traffic and increased server burden. If we observe the proportion between the times of unnecessary requests (receiving the same body) because of conservative prediction, and the total times of every minute's request in our experiment, and we use that as the measurement of wastage, we will find that the result from our algorithm will waste less than 100% on average. However, Squid's and server's result will cause wastage that is more than 10 times the amount of our algorithm.

The accuracy of our prediction, which means the predicted value is exactly the same with object's actual TTL, is significantly higher than others. This indicates, on the whole, that the prediction base of our algorithm is close to the actual situation.

Our adaptively monitoring procedure also lessened the mistakes in prediction, and increased the credibility of the caching proxy compared to Squid's heuristic algorithm.

6.5 Conclusion

The problems we noted in chapter 4 and chapter 5 reflect that most current mechanisms to set the TTL of objects are either ad-hoc or too conservative. More importantly, there is no measure in the prediction process to take care of the changing life-span pattern of an object. The resulting discrepancy not only causes unnecessary network bandwidth consumption but also potential outdated content reuse.

In this chapter, we study the sensitivity of the expectation life-span values for TTL prediction and to propose a novel adaptive mechanism for accurate TTL estimation. Through proper stochastic modeling (such as Gamma or WeiBull) on the probability distribution function (PDF) of the life-span value of a web object based on previously observed values, its predictability as well as expectation value (if any) can be obtained. Then with correlation pattern recognition model, any changing behavior of the life-span value of the object can be reflected as an update to its corresponding PDF. Both the details of the mechanism and its simple implementation in reverse proxy cache are given. Through active monitoring on the content updating dynamics of real web objects on Internet, we show that our TTL estimation is far more accurate than both directives sent out by content servers and TTL settings by current proxy caches. This result is important because it allows content servers and proxy caches to conduct their own content freshness risk analysis for more efficient web caching.

Chapter 7 Conclusion and Future Work

In this chapter, we present the conclusions resulting from the work performed in this thesis, and suggest areas of future work for possible avenues of further research.

7.1 Conclusion

In this thesis, we delved into objects' attributes, and analyzed how objects' content settings will affect the effectiveness in their cacheability from both the perspectives of the caching proxy and origin server. Furthermore, we proposed our solution in helping origin servers to enhance their correctness content settings by the effective prediction of objects' time to live period, which will improve objects' cacheability and efficient content delivery.

When analyzing cacheability effectiveness, we base on objects themselves and model all factors affecting their cacheability to obtain numeric values for quantitatively achieving a complete measurement. To further ascertain the usefulness of these models, corresponding appropriate simulation experiments were conducted. These experiments illustrate our models' usefulness in aiding the adjustment of the caching proxy's policy, origin servers' design strategy, and even certain areas of web caching research.

We observed that most objects' cacheability can be improved by proper content settings, especially in properly setting an object's time to live, from model measuring the effectiveness of objects' cacheability. We proposed the use of Gamma Distribution Model to capture an individual object's change regularity and predict a more accurate future freshness period for the object.

To demonstrate that our model can provide effective predictions, we presented our experimental architecture that offered the following benefits: ability to maximize content reusability, maximize accuracy to sent correct object to clients and minimum bandwidth requirement. Our experimental results, in comparison with origin servers' rough settings and the Squid's heuristic algorithm, demonstrates that our prediction algorithm significantly improves correctness in prediction of objects' freshness periods, and directly benefit web caching.

To summarize, we present the following specific contributions of this thesis below:

- From the perspective of the caching proxy, our proposed Effective Cacheability (EC) mathematical model provides a measurement to measure the effectiveness of caching an object.
- Still from the perspective of the caching proxy, we conducted a detailed study and monitoring experiment into analyzing the combinational effects of the many factors that affect the cacheability of an object, and to study the relationships between the individual factors. This study further emphasized the usefulness of having the Eeffective Cacheability (EC) mathematical model.
- From the perspective of the origin server, our proposed Effective Content Delivery (ECD) model index can aid content providers in analyzing content settings when modifying the complexity and attractiveness of a website. We also conducted an experiment to demonstrate that the usefulness of the index.

- Since most current mechanisms to set the TTL of objects are either ad-hoc or too conservative, we proposed to take care of the changing life-span pattern of an object during the prediction process. This novel adaptive mechanism for accurate TTL estimation based on the concept of Two-Steps TTL Adaptation Model. These two steps are making use of existing two mathematical models, the Gamma Distribution Model and the Correlation Pattern Recognition Model. Our experiment demonstrated the feasibility and effectiveness of our algorithm, and the results reflected a significant improvement in the veracity of an object's freshness period prediction, when compared with servers' directives and the existing Squid heuristic algorithm.

7.2 Future Work

In this section, we present several directions for future work that are motivated from our work.

- Exploit more benefits of our numeric measurement in aid of web caching research

  The experiment conducted for our proposed Effective Cacheability model has hinted on possible cache replacement research. The model can be used to determine what to cache in the cache, and when the cache is full, what to remove. Experiments can be performed to see whether the measurement can be an independent metric, or it can work with other metrics to enhance existing cache replacement policies. Meanwhile, experiments can be performed to investigate using the measurement in other web caching research areas, such as prefetching.

- Properly shorten regularity capturing period for a real-time prediction system

  We plan to implement our prediction algorithm into a real-time delivery system. That will result in our experiment architecture being modified to be more practical. The system should be implemented on the reverse proxy as an independent module, where it can aid the origin server in examining the content settings, and capture the historical behavior of objects.

  After our experiment proves that our algorithm is suitable for object change regularity prediction, we can shorten the regularity capture period, the learning object behavior stage, and pick up the best observation period to have effective prediction. For example, we observe that ebay change regularity is similar in every hour from Aug 12 to Aug 18: its time-to-live period is 1 minute in first minute of the hour, following is 9 minutes, following are all 10 minutes. Therefore 1 hour is the best learning period for ebay. It may be different among all objects. Our system should have automatic pick-up function.

- Choose good detection period

  The adaptive monitoring period can be called upon to work in a suitable period for good detection of the behavior of objects. The period to detect whether our prediction result accords with actual situation is the same as its original learning period. Our simulation should have a memory function for every object. Since credibility of the object's freshness is very important, the interval cannot be too long. But too short an interval may waste large amount of time on useless work. Therefore proper selection on the rest interval is as important as the detection period. It should be one of emphasized function in the system.

- More detailed analysis on the factors that affect cacheability

  The work in Section 4.2.5 can be furthered by using a more well-established Factor Analysis approach to analyze in detail the contributions of the various factors to the EC measure [53]. This will give clearer understanding of how each factor impacts the EC measure.

- Further studies into the effectiveness of content delivery

  A more detailed approach to studying the effectiveness of content delivery may yield more specific and direct results that can be used by web server owners to improve upon their content. This includes performing more detailed analysis, or looking for better ways of measuring the effectiveness of content delivery.

- Further comparisons between previous research on TTL versus Adaptive TTL Estimator

  Perhaps a suitable collection of TTL work representing some of the previous ideas should be considered and then comparisons of performance made between such work and tour adaptive TTL Estimation for efficient web content reuse. This can further prove that our TTL adaptation algorithm performs better than other previous TTL work.

Bibliography

[1] C.E. Wills, M. Mikhailov and H. Shang. *N For the Price of 1: Bundling Web Objects for More Efficient Content Delivery*. Proceedings of the 10th World Wide Web Conference, 2001.

[2] J. Wang, (1999). *A Survey of Web Caching Scehemes for the Internet*. ACM SIGCOMM Computer Review, 29(5): p36-46, 1999.

[3] B.M. Duska, D. Marwood, and M.J. Feelay, (1997). *The Measured Access Characteristics of World Wide Web Client Proxy Caches.* Proceedings of USENIX Symposium on Internet Technologies and Systems, 1997.

[4] G. Barish and K. Obraczka. *World Wide Web caching: trends and techniques.* IEEE Communications Magazine, 38(5):178-- 184, May 2000.

[5] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams and E. A. Fox. *Caching proxies: limitations and potentials.* Proceedings of 1995 World Wide Web Conference,1995.

[6] P. Cao, J. Zhang and K. Beach. *Active cache: caching dynamic contents on the Web.* Technical Report *CS-TR1998 -136*3, 1998.

[7] D. Wessels. *Information Resource Caching FAQ.* URL: **http://ircache.nlanr.net/Cache/FAQ/**

[8] B. Liu, G. Abdulla, T. Johnson and E. A. Fox. *Web response and proxy caching.* Proceedings of WebNet98, Orlando, November. 1998.

[9] E. Cohen and H. Kalpan. *Refreshment Policies for Web Content Caches.* Proceedings of the INFOCOMM 2001 Conference.

[10] A. Chankunthod, P.B. Danzig, C. Neerdaels, M.F. Schwattz, and K.J. Wornel, (1996). *A Hierarchical Internet Object Cache*, Proceedings of USENIX Symposium on Internet Technologies and System, 1996.

[11] P.S. Yu, and E.A. MacNair, (1998). *Performance Study of a Collaborative Method for Hierarchical Caching in Proxy Servers*. Proceedings of the 7th World Wide Web Conference, 1998.

[12] D. Povey, and J. Harrison, (1997). *A Distributed Internet Cache*. Proceedings of the 20th Australian Computer Science Conference, 1997.

[13] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay. *Design considerations for distributed caching on the Internet.* Proceedings of Nineteenth International Conference on Distributed Computing Systems, May 1999.

[14] A. Dingle, and T. Partl, (1996). *Web Cache Coherence*. 5th International World Wide Web Conference, 1996.

[15] P. Cao, and C. Liu, (1997). *Maintaining Strong Cache Consistency in the World Wide Web.* Proceedings of the 17th IEEE International Conference on Distributed Computing Systems, 1997.

[16] *B. Krishnamurthy, and C.E. Wills, (1997).* Study of Piggyback Cache Validation for Proxy Caches in the World Wide Web. Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems, 1997.

[17] B. Krishnamurthy, and C.E. Wills, (1998). *Piggyback Server Invalidation for Proxy Cache Coherency.* Proceedings of the 7th World Wide Web Conference, 1998.

[18] B. Krishnamurthy, and C.E. Wills, (1999). *Proxy Cache Coherency and Replacement – Towards a More Complete Picture.* Proceedings of the 8th World Wide Web Conference, 1999.

[19] E. Cohen and H. Kaplan. *The Age Penalty and its Effect on Cache Performance*. Proceedings of VSITS 2001.

[20] J.C. Mogul. *Errors in Timestamp-Based HTTP Header Values*. Tech. Rep. 99/3, Compaq Western Research Lab, December 1999

[21] B. Krishnamurthy and M. Arlitt. *PRO-COW: Protocol Compliance on the Web.* Tech. Rep. 990803-05-TM, AT&T Labs-Research, 1999.

[22] A.Chankhunthod, P.B. Danzig, C. Neerdaels, M.F. Schwartz, K.J. Worrell, *A Hierarchical Internet Object Cache.* Proceedings of the USENIX Technical Conference, Jun 1998.

[23] S. Manley, M. Seltzer. *Web Facts and Fantasy*. Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems (USITS'97), Dec 1997

[24] S. Gribble and E. Brewer *System Design Issues for Internet Middleware Services: Deductions from a large Client Trace* Proc. of the 1997 USENIX Symposium on Internet Technologies and Systems (USITS'97), Dec 1997

[25] R. Caceres, F. Douglis, A.Feldmann, G. Glass, and M. Rabinovich, (1998). *Web Proxy Caching: The Devil is in the Details*. SIGMETRICS. Workshop on Internet Server Performance, June 1998.

[26] A. Feldmann, R. Cceres, F. Douglis, G. Glass, and M. Rabinovich. *Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments*. Proceedings of IEEE Infocom '99, Mar 1999.

[27] A. Wolman, G.M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H.M. Levy. *On the Scale and Performance of Cooperative Web Proxy Caching*. Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles, December 1999.

[28] C.E. Wills and M. Mikhailov. *Towards a Better Understanding of Web Resources and Server Responses for Improved Caching*. Proceedings of the 8th International World Wide Web Conference, May 1999.

[29] Xiaohui Zhang, *Cacheability of Web Objects*, Master thesis of computer science department in Boston University, USA, 2000

[30] Timo Koskela, Jukka Heikkonen and Kimmo Kaski, *Modeling the Cacheability of HTML Documents,* Proceedings of the 9th World Wide Web Conference, 2000.

[31] X. Zhang. *Cacheability of Web Objects*. Master Thesis of Computer Science Department in Boston University, USA, 2000.

[32] T. Koskela, J. Heikkonen, and K. Kaski. *Modeling the Cacheability of HTML Documents*. Proceedings of the 9th World Wide Web Conference, 2000.

[33] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey Mogul, *Rate of change and other metrics: a live study of the World Wide Web,* USENIX Symposium on Internet Technologies and Systems, December 1997, pp 147-158

[34] E. Craig Wills and Mikhail Mikhailov, *Examining the Cacheability of User-Requested Web Resources*, Proceedings of the 4th International Web Caching Workshop, San Diego, CA, March/April, 1999

[35] E. Brain Brewington, Cybenko George: *How dynamic is the Web?* Proceedings of the 9th World Wide Web Conference, Computer Networks 33(1-6): 257-276, 2000

[36] X. Chen, and P. Mohapatra. *Lifetime Behavior and Its Impact on Web Caching*. IEEE Workshop on Internet Applications, 1999.

[37] A. Chankhunthod, P.B. Danzig, C. Neerdaels, M.F. Schwartz, K.J. Worrell. *A Hierarchical Internet Object Cache*. Proceedgins of the USENIX Technical Conference, 1996.

[38] V.N. Padmanabhan, L. Qiu. *The Content and Access Dynamics of a Busy Website: Findings and Implications*. SIGCOMM 2000

[39] Junghoo Cho, Hector Garcia-Molina, *Estimating Frequency of Change*, Technical report, Standford University, 2000

[40] Squid Frequently Ask Question, **http://www.squid-cache.org/Doc/FAQ/FAQ.html**

[41] D. Wessels, (2001). *Web Caching*. O'Reilly & Associates, Inc, 1st Edition, 2001

[42] R.Fielding, J.Gettys, J.Mogul, H.Frystyk and T.Berners-Lee *Hypertext Transfer Protocol – HTTP/1.1* RFC 2068, Jan 1997

[43] David Kristol, and Montulli Lou, *RFC 2965 HTTP State Management Mechanism*, October 2000

[44] Balachander Kreshnamurthy, Jennifer Rexford, *Web protocols and Practice*, 2000

[45] V.C. Alex, (1992). *A Global File System*. Proceedings of the 1992 USENIX File System Workshop, 1992.

[46] hot100.com

[47] Netscape Proxy Server Administrator's Guide Version 3.5 for Unix **http://developer.netscape.com/docs/manuals/proxy/adminux/contents.htm**

[48] L. Jay Devore, *Probability and Statistics for Engineering and the Sciences*, Fourth Edition, 1995. **http://www.itl.nist.gov/div898/handbook/eda/section3/eda366b.htm**

[49] K. S. Fu, *Digital Pattern Recognition*, Springer-Verlag, 1980.

[50] M. Li, B. H. Xu and Y. S. Wu, *An H2-Optimal Control of Random Loading for a Laboratory Fatigue Test*, Journal of Testing and Evaluation, 26 (6), Nov. 1998, pp. 619-625.

[51] M. Li, Y. S. Wu, B. H. Xu, W. J. Jia and W. Zhao, *An On-Line Correction Technique of Random Loading with a Real-Time Signal Processor for a Laboratory Fatigue Test*, Journal of Testing and Evaluation, 28 (5), Sep. 2000, pp. 409-414.

[52] M.A. Rick Martinelli, *Pattern Recognition in Time-series,* Journal of Technical Analysis in Stocks & Commodities, January issue, 1995. **http://www.maui.net/~haikulab/pattrec.htm**

[53] http://www.statsot-tinc.com/textbook/stfacan.html

Appendix

Gamma Distribution

Probability Density Function:

The general formula for the **probability density function** (pdf) of the gamma distribution is $f(x) = \dfrac{(x-\mu)^{\alpha-1} e^{-(x-\mu)/\beta}}{\beta^{\alpha}\Gamma(\alpha)} \qquad x \geq \mu; \alpha, \beta > 0$

where α is the shape parameter, μ is the location parameter, β is the scale parameter whose effect is to stretch out graph, is the gamma function which has the formula

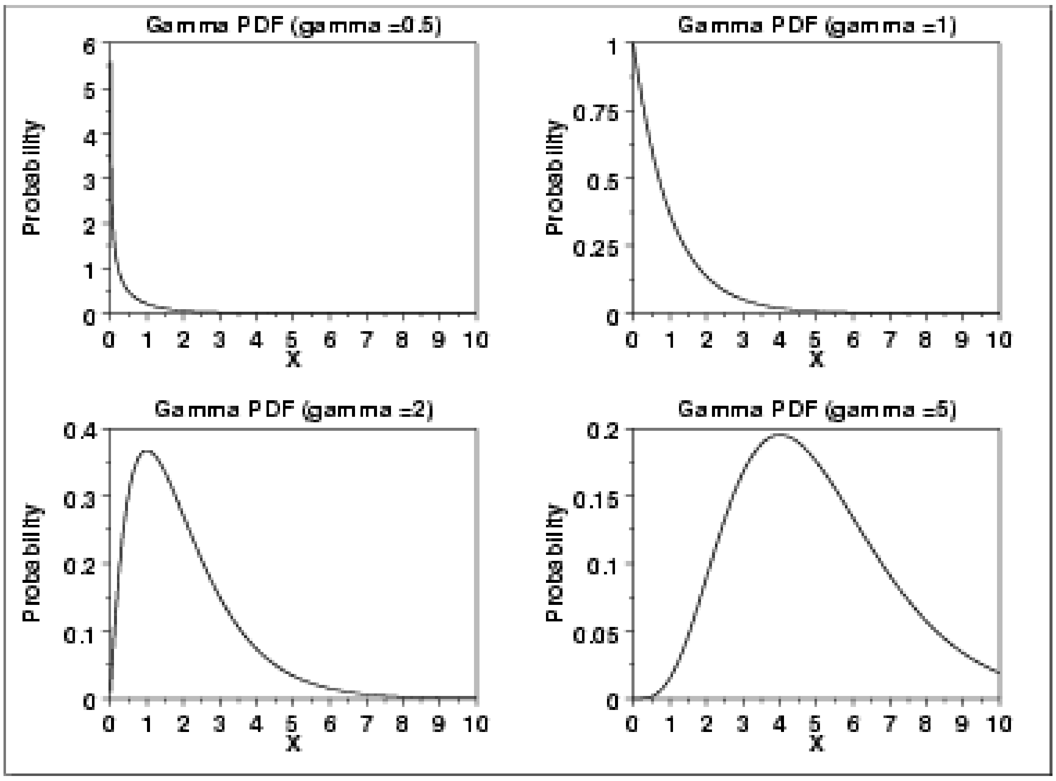
$$\Gamma(\alpha) = \int_0^{\infty} t^{\alpha-1} e^{-t} dt$$

The case where $\mu = 0$ and $\beta = 1$ is called the standard gamma distribution. The equation for the standard gamma distribution reduces to

$$f(x) = \frac{x^{\alpha-1} e^{-x}}{\Gamma(\alpha)} \qquad x \geq \mu; \alpha > 0$$

Since the general form of probability functions can be expressed in terms of the standard distribution, all subsequent formulas in this section are given for the standard form of the function.

The following is the plot of the gamma probability density function.


Gamma PDF (gamma =0.5)
Gamma PDF (gamma =1)
Gamma PDF (gamma =2)
Gamma PDF (gamma =5)
Probability
X