CacheL: Language Support for Customizable Caching Policies

J. Fritz Barnes Raju Pandey

Parallel and Distributed Computing Laboratory
Computer Science Department
University of California, Davis, CA 95616
{barnes, pandey}@cs.ucdavis.edu

Abstract

Web caching has emerged as one solution for improving client latency on the web. Cache effectiveness depends on the policies used to route requests to other caches and servers, maintain up-to-date web objects (HTML files, images, etc.) and remove objects from the cache. Traditional caches apply the same policies to all web objects; however caches lack the ability to apply specialized policies to a subset of cached objects. We present an extensible caching infrastructure in which cache administrators, servers, and end users can customize how web objects are cached, replaced, and kept consistent. This paper presents a domain-specific programming language, CacheL, for defining customizable caching policies. The resulting infrastructure will lead to the development of a more efficient and customizable web caching solution.

1 Introduction

Web caching [11, 17, 4] improves client latency through the use of intermediary servers, or caches, that accept HTTP requests from clients. When a request is made of the cache, the cache returns a copy of the web object being requested. If a replica does not exist, the cache contacts other caches or the origin server to retrieve a copy.

Caches improve web performance by migrating web objects closer to clients. Migration reduces the number of network hops a clients request must traverse, thereby reducing propagation delay and network load. Further, by not contacting the origin server, caches reduce server load. Caches take advantage of temporal, geographical, and spatial localities [7] of web objects.

The behavior of a cache is characterized by the following:

- On receiving a request, the cache checks if the requested web object is available locally. If the object is available, the cache determines whether the local copy is *fresh*, or up-to-date, using a *coherency policy*. Fresh documents can be returned to the user. *Stale* documents necessitate making an `If-Modified-Since` (`IMS`) request to determine whether the document has been modified.
- If the web object does not exist locally, the cache contacts other caches or the origin server to locate the document. The cache uses a *routing policy* for deciding how it should locate a document.
- Once the cache has retrieved the object, it uses a *placement policy* to determine whether the cache should make a copy of the object in the local store.
- When the local store exceeds a predefined high-water usage of resources, the cache removes enough objects from the object store to reach a predefined low-water mark. A cache uses a *removal policy* to determine which web objects to remove.

Removal policies can have a significant effect on the cache hit rate, because removing web objects that will be accessed again causes misses. Previous work has explored the performance of several removal policies: Least-Recently-Used (LRU), Least-Frequently-Used (LFU), Perfect LFU [2], removal of large objects from caches [19], hybrid techniques utilizing estimation of client latencies [20], and cooperative caching techniques [9, 10]. These techniques demonstrate similar performance when applying the removal policy to all web objects. The algorithms each have special cases in which they perform well. An analysis of cooperating caches [10] concluded that dynamic policies that can turn on and off cooperation might be useful. Ideally it would be useful to apply a removal policy to the set of objects for which it performs well.

Caches determine whether documents are *stale*. Stale documents must be verified to determine that they are still up-to-date. Maintaining coherency affects the performance of caches because coherency may require checks to see whether a document is up to date. Current caches [17, 16] utilize a combination of MIME headers [6] (`Last-Modified`, `Expires`, and `Cache-Control`) to determine whether a web object is fresh. Servers can specify when an object is no longer valid with the `Expires` header. Alternatively, a maximum age, or time a document can spend in the cache before being verified can be specified with the `Cache-Control`[1] header. When a request is made for a stale document the cache will make an `If-Modified-Since` request to the origin server. The server will only transmit the document if the document has changed. Coherency policies are limited by the HTTP standard. Caches need information from the origin server to determine coherency. Passing this information between servers and caches requires recognized headers. Alternatively, if servers could inform caches how to handle coherency for their documents, the server would not be limited defining coherency.

Current caches support a rich set of policies for routing, coherency, and removal of web objects, yet there are many limitations. Routing policies cannot discriminate on the type of content or the semantic information of web objects. Removal policies have not explored applying different decisions to different objects although the performance in caches of objects can vary widely. For example, images result in a hit rate that is four to five times greater than hypertext (HTML) files [14]. We believe caches should be customizable while running, and should export interfaces that allow servers to participate in how objects are handled. Our solution uses a domain specific language to specify policies. We discuss how programmable policies provide customization and allow servers to participate in defining how a cache stores objects originating from that server.

[1]The `Expires` header is defined for use with HTTP/1.0, the `Cache-Control` header is defined for use in HTTP/1.1

This paper is organized as follows: we discuss the need for customizable caching policies in Section 2. In Section 3, we follow this up with a discussion of `CacheL` a domain specific language for specifying caching policies, and discuss how caches can use `CacheL` to create customizable policies. We discuss some example policy implementations in Section 4. This paper concludes with a discussion of previous work, Section 5, and a summary and future work, Section 6.

2 The Case for Customizable Policies

Caching systems utilize predefined policies for cache management and document consistency. In the previous section we have discussed some commonly used policies. These policies assume that all documents are similar and therefore a single policy is applied to all documents. However there are often differences among web objects; some examples are listed below.

- The author of a set of web pages can eliminate the concerns of coherency for images and applets. Coherency concerns are ignored, because the HTML objects that reference the images and applets can utilize a new name for modified images and applets.
- Summary pages which change on a set schedule. These pages include: index pages, table of contents for web magazines and weather forecasts.
- The news articles on web magazines and newspapers have a time of life during which they will be accessed.
- A reference material's popularity is independent of time. Accesses to these pages are more likely to change due to a paradigm shift rather than time.

In the above categories we list general types of objects. One could also categorize web objects by server location. Fetching web objects that are on international networks have different performance characteristics than nearby networks.

In the rest of this section we discuss three examples that exemplify why customizable policies are important in today's caching systems. We discuss an example of a daily Internet magazine, multiple caching hierarchies defined by content, and a scheme for hierarchical removal of documents.

We first consider caching policies for a daily Internet magazine designed to improve the hit rate in the cache for articles from the magazine. The following aspects of the news magazine require a customizable coherency policy.

- The table of contents or the magazine home page should be regularly reloaded. If there is enough demand for the magazine by the cache users, the cache can pre-load the magazine contents.
- The current edition can specify the *headline articles* or those articles that are most likely to be accessed. The cache makes use of this information to pre-fetch these articles.
- Deadlines can be applied to the articles in the cache. This is particularly useful for a news magazine as it is likely that articles will have a given time-of-life that is dependent on the client demographics of a cache and the rate at which new editions are created.

Allowing the news magazine administrators to specify the desired policies for the web objects associated with the magazine allows them to take advantage of the three items outlined above.

A second example of the usefulness of providing customized caching policies is an organization in which there are multiple caches utilized by different populations. An example of this is a university setting where the cache used by members of the physics department is likely to have high hit rates for physics documents. However the same department is likely to have a much lower hit-rate for biological issues. In this instance, the physics server could create cache entries for the biological documents, but would likely find the documents at a cache in the biology department. Although the biology department cache could be specified as a peer, it might be advantageous to make use of different hierarchies dependent on the type of documents.

A final example involves cache removal policies. Consider a hierarchy of documents that can logically be thought of as a single entity, such as an article that is spread out across several different web objects. When one of the items in the article expires it may be appropriate to also remove the other items from the cache.

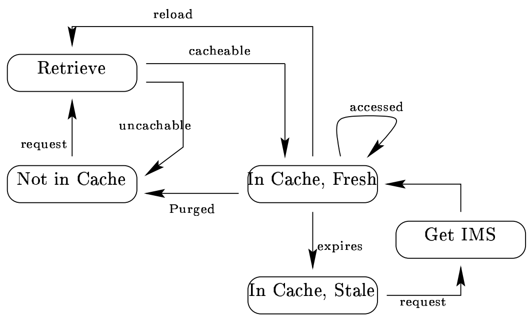

Figure 1: States in a cache

3 The `CacheL` Language

In this section, we will discuss language characteristics of `CacheL`. Our goal is to create a language that can be used by clients, web servers, and cache administrators in order to create customizable caching policies. Our system provides the ability to specify different routing, placement, and coherency policies for objects, as well as changing the way in which removal of objects is performed. In this section, we discuss a simple example to introduce the language. We then discuss some of the characteristics of the language. We follow this by giving details of how the language can be implemented.

3.1 Overview

A cache combines local policies defined by the cache administrator and policies specific to web objects defined by the server administrator from which the objects originated. `CacheL` is used to specify actions that are to be taken when an event occurs for a web object or set of objects. We are concerned with the events that cause changes in the state of a cache. Figure 1 shows a state diagram of the different states for an object in the cache and the events that cause a cache to change from one state to another state.

As an example, we consider a pre-fetch caching

```
Event: New-Store
   use_count = 0

Event: Access-Offline
   use_count = use_count + 1

Event: Stale
   if ( use_count > GetMime( "X-Cache-Policy-Count" ) )
      CacheFetchIMS( CurrentURL() )
      use_count = 0
   endif
```

Figure 2: Example policy that pre-fetches documents when they expire

policy for pages that change often. When we attempt to decide whether to pre-fetch the page, we want to take into account the popularity of the page which we are pre-fetching. This policy would be appropriate for network magazines or organization pages that change periodically on a set schedule.

In Figure 2, we show this pre-fetch policy. The policy specifies actions to take when `New-Store`, `Access-Offline`, and `Stale` events occur in the cache. The policy specifies a use counter, `use_count`, that measures how often the web object is accessed. Whenever the web object enters the cache, a `New-Store` event occurs. Policy actions are executed when their corresponding events occur. For instance, in this example the `New-Store` event initializes the `use_count` variable to zero. Web object accesses cause `Access-Offline` events. For these events the `use_count` is incremented. When the object expires a `Stale` event occurs, the policy determines if the number of accesses is greater than the server's predefined requirement specified by the X-Cache-Policy-Count MIME header. If this requirement is met, the policy instructs the cache to update the current object by executing an HTTP `GET IF-MODIFIED-SINCE` (IMS) request. The prefetch policy allows us to change the way in which a set of objects is maintained by the server. In this example, it allows us to eliminate the necessity of making IMS requests for a set of objects.

We now describe the details of the language. `CacheL` consists of three components: events, actions, and operations. Changes in cache and object states trigger events. Policy scripts can register actions or code to be executed when an event occurs. Cache operations allow the script to control how the cache reacts to these events.

3.2 Cache Events

The building block for providing extensible caches is the use of events within the cache. Administrators can write actions to enforce a policy for an arbitrary set of web objects. We consider the following events.

Route events occur when objects must be retrieved. The action must send an HTTP request to a cache or the origin server, return an alternate document in the cache, or respond with an error.

New-Store events occur after new documents are retrieved or Get IMS requests for a stored document return a new document.

Access-Inline events occur when requests are made for a document in the cache. These events occur during retrieval of the document for the client. Scripts can perform checks for coherency as well as modify MIME headers that are passed to clients.

Access-Offline events occur when requests are made for documents in the cache. Execution of actions associated with this event are independent of the steps to actually retrieve a web object and therefore these actions do not affect the retrieval time of a web object.

Stale events occur when the cache must verify that the cached document has not changed on the origin server. Stale events occur independent of the requests for an object.

High-Water events occur when the cache is full.

Sort events occur when the cache compares two different objects to determine which of the two documents should be removed from the cache.

Purge events occur when a document is removed from the cache.

Timer events occur as a result of policy-defined alarms.

3.3 Specifying Actions with `CacheL`

Policy scripts use `CacheL` to specify actions in response to an event. Actions consist of a set of cache operations and expressions. Customized policies can be created using these actions. Cache operations include contacting other caches and servers, affecting documents stored locally, and setting alarms to schedule an action in the future. We enumerate some of the pertinent cache operations:

- `CacheFetch( URL )`: requests a web object from the object's originating server.
- `CacheFetchIMS( URL )`: performs an `If-Modified-Since` request for the given web object.
- `CachePost( URL, Data )`: contacts a host with a post request and includes the `Data` in the request. This is primarily used for actions to pass information back to their originating server.
- `CacheICPFetch( HOST, URL )`: requests a web object from another cache using the Internet Caching Protocol.
- `CacheResponse( URL )`: sends a local copy of `URL` to the client as a response to the clients request.
- `CachePurge( URL )`: removes the specified object from the local store.
- `CacheStore( URL )`: instructs the cache to store the object that is being requested.
- `CacheSetTimer( Date, Time )`: sets an alarm to go off.
- `CacheLog( Message )`: provides a debugging mechanism that writes the given message to the cache's log file.

3.3.1 Expressions

`CacheL` saves the state associated with a policy through the use of variables. The language supports relational, arithmetic, and logical expressions; conditional execution; set, array, numeric, and string data types; and iteration over sets. The language also provides special facilities for manipulating date/time values as well as MIME headers associated with objects. We ignore these details here.

3.3.2 Granularity of Policy Specifications

Policies may apply to a single object as well as sets of objects. Policies that govern a single document, such as coherency and routing, are applied to a single object, although the same policy may be reused for many objects. Our example policy which prefetches objects when they expire is an example of a policy specification on the object level.

Removal policies often deal with multiple objects. This policy utilizes characteristics of a set of objects in order to make decisions as to which objects to remove from the cache store.

Variable state in `CacheL` is associated at the granularity of the policy. If a policy is applied on an object level, then variables in the policy are scoped by objects. For example, two documents utilizing the pre-fetch on expire policy would have different usage counters. Similarly policies associated with a set of documents would scope the variables with the set of documents. `CacheL` also provides policy variables which can be accessed by a given policy when operating on any object or set of objects.

3.4 Associating Policies with Web Objects

A *cache administrator*, the person responsible for the cache, and a *server administrator*, the person running the web site from which a web object originates, define a cache's policy for an object. We consider techniques utilized by both cache and server administrators to define the policy applied to a given web object or set of web objects.

3.4.1 Cache Administrators

Cache administrators can define a set of web pages, or a MIME header that indicates which policy should be used for a given web object. At this point the policy is registered for use with that web object. In Figure 3, the configuration file specifies the policies to utilize for the documents on the CNN and ABCNews web sites. The configuration file specifies the objects a policy applies to and the URL of the policy script. Cache administrators can utilize wildcards for easy specification of policies applying to multiple objects at a host or multiple hosts. While

```
www.cnn.com/index.html          http://mycache.org/TOC.plcy
www.cnn.com/features/*          http://mycache.org/Article.plcy
www.cnn.com/*                   http://mycache.org/cnn_removal.plcy
www.abcnews.com/index.html      http://mycache.org/TOC.plcy
...                             ...
```

Figure 3: Entries from a `mycache.org`'s CacheL configuration file that define web objects to which different policies are applied. The configuration file can be used to define different policies for the same document.

restricting policy placement to cache administrators provides limited security,[2] it restricts the ability of server administrators who may have the best knowledge of their objects to define the policy mechanisms appropriate for a given web object.

3.4.2 Server Administrators

In addition to allowing cache administrators to define policy mechanisms, server administrators can define policies relevant to the objects on their server. We consider two techniques for server administrators to specify policies. First, one adds an X-Cache-Policy MIME header that specifies the URLs of policy scripts to apply to the object. When a cache retrieves an object with an X-Cache-Policy header it will retrieve the policies if they are not already in the cache. Then it will apply actions for the web object so that the desired policies can be enforced. A drawback of this mechanism is the additional support that is needed by the MIME headers and servers for handling the policy mechanism specification.

The alternative technique is to utilize a standard directory, e.g. `http://www.origin.-com/cache-policy/`, on web servers which can be checked by caches to locate policies that exist for web objects at that server. This directory includes an index that defines the different policies available, and the web objects to which they apply. This technique has a disadvantage: if servers do not commonly add policies to web objects, this technique will require overhead to access the policy index at the server.

3.4.3 Arbitrating Policies

One problem that evolves once we have multiple policies is how the policies work together. This is primarily a concern when we wish to remove web objects, but also crops up in how we maintain coherency for objects. With respect to removing objects, even if a server would like a given object to be always cached, problems can arise if people are not accessing the object or worse if all servers wanted all of their objects cached.

In coherency, clients may want to apply their own policies for what is considered out-of-date for a given document. Consider the New York Times online service, which is updated every ten minutes. Its publishers would prefer caches to update their home page with the same frequency. However, the cache would rather provide information to users with at most a half hour out-of-date policy in order to take advantage of geographical locality. Additionally, a user might recognize the fact that they only read the New York Time web site once a day, so they only care that the news is current for today.

Arbitrating between the policies desired by clients, servers, and the cache can be a complicated task. We consider the following two forms of arbitration. In arbitrating removal policies, we allow servers to specify the removal policy for a set of documents. Therefore, a policy can rank the order in which a set of documents are to be deleted, but it does not affect documents that are not inside the set. When determining the ordering for documents between different sets we utilize the default removal algorithm for comparing objects (i.e. LFU, LRU.)

In arbitrating coherency policies we can either arbitrarily specify how the cache will operate (i.e. it will ignore the server specified expiration times), or extend servers to allow better arbitration. When we extend servers, the server should inform the cache of the expiration time of an object, as well as the maximum length of time the server will allow caching of objects after the deadline. Essentially the server is specifying a range during which the cache and clients can specify the policy they desire for the given web page.

[2] As long as the administrator is actively responsible for policy content.

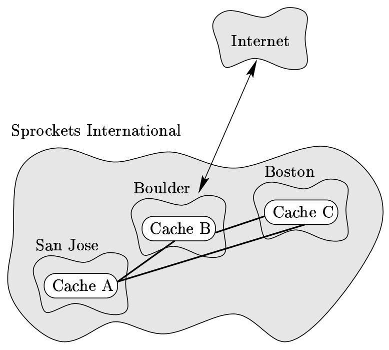

Figure 4: Cooperative Caches located at different sites for Sprockets International

4 Examples

We now explore two examples that utilize our extensible caching system for specifying alternative policies for subsets of cached objects. The first application utilizes the characteristics of an object to apply different removal policies. The second application we consider uses extensible policies to handle protocol extensions for caches.

4.1 Removal Policy Customization

We examine the ability of a caching administrator to specify how objects should be removed from the cache store. The caching administrator controls several caches for an organization distributed across several different work sites. A cache exists at each of these sites and a single connection to the Internet is provided. The bandwidth available to the Internet is limited[3], and the bandwidth between cache locations is plentiful. This scenario is pictured in Figure 4, where Sprockets International has offices in San Jose, Boulder and Boston, with a single connection to the Internet. As a result of limited bandwidth to the internet, the cache administrator would like to utilize removal algorithms that favored sharing of objects in caches. However, the administrator would like to cache popular sites, such as a trade journal at all sites.

One description of this policy is shown in Figure 5. This policy utilizes several cache operations we did not discuss earlier. In particular, we assume that the cache we are implementing this policy for has a directory that contains objects cached by nearby servers [15, 13, 5]. The operation `CacheDirectoryLookup` is utilized to consult the local entries in the cache table. Additional operations that are used are the `CacheLess` and `CacheGreater` operations which are used to define the ordering between two documents in the cache. If the default cache algorithm is to be used for the comparison instead of this algorithm, the policy can use the `CacheCompareDefault` operation.

[3]Bandwidth limitations are either physical or due to the fact that the Internet Service Provider charges for bandwidth use

```
# Actions for implementing caches that prefer removing
# objects that are stored in nearby caches.

Event: Compare( objA, objB )
   # Determine whether object A is in a peer cache
   if ( CacheDirectoryLookup( objA ) != "NotFound" )
      objACached = false
   else
      objACached = true
   endif

   # Determine whether object B is in a peer cache
   if ( CacheDirectoryLookup( objB ) != "NotFound" )
      objBCached = false
   else
      objBCached = true
   endif

   # if both are cached in peers use the default ordering
   if ( objACached && objBCached ) CacheCompareDefault()

   # otherwise favor removal of the object in a peer cache
   else if ( objACached ) CacheLess()
   else if ( objBCached ) CacheGreater()
   else CacheCompareDefault() endif
```

Figure 5: Actions used to implement a specialized removal policy

In order to only apply the above algorithm to files that are not located at our popular servers we utilize the following assignment of the policy to web pages:

The following excerpt from the `CacheL` configuration file allows us to use a default removal policy for popular servers and our alternative cost-based policy, Figure 5, all other objects.

```
*.acm.org/*         default
*.nytimes.com/*     default
*.redhat.com/*      default
*/*                 http://sprockets.org/Removal.plcy
```

In this manner, when comparing pages that are not in the popular list of web sites the default removal policy for the cache is utilized to determine how to sort which objects are removed first. If we compare objects from servers not in our popular list they will use our defined sorting rules, Figure 5. When we compare a popular object with an unpopular object we utilize the default comparison techniques.

4.2 Hit Metering and Usage Limiting

CacheL can also be used to implement new protocols. An interesting example is hit metering and usage limiting (RFC 2227) [12]. CacheL provides all of the facilities necessary to make these changes for caches. The cache can associate a counter with web objects to keep track of metering and make the appropriate GET IMS requests to the origin server to update the count of clients making requests for this object.

In Figure 6 we describe most of the actions for implementing hit-metering and usage limiting. When we receive routing events we add headers to signify to the server that we support hit-metering. Additionally when routing IMS requests to the server we append information about the number of accesses to this page. When a document arrives for storage, we parse the headers to determine whether we should provide hit-metering, usage limiting, or both. When accesses are made, we determine during the request whether we have exceeded the usage limit. We increment our count of how often a page was accessed using the `Access-Offline` event.

5 Related Work

Caughey et al. [3] describe an architecture for caching web objects. Their ideas are similar to ours in that by caching objects they can customize the way in which the caching occurs. Their goal is to provide *open caching* that exposes the caching decisions to the users of the cache. Our work differs in that we are interested in minimizing the interaction between clients and caches while still providing customizable policies.

Other languages have been proposed for use on the web. One of these is WebL [8], which is used to provide document processing. The goal of this system is to provide both reliable access to web services through the use of service combinators and techniques for gathering information and modifying documents through markup algebra. WebL is utilized for a different purpose than CacheL, and in fact combining the two may be a fruitful. It would allow caches to make simple customizations of web pages rather than requiring uncacheable web pages that utilize CGI scripts at the server.

The squid web cache [4, 18, 16] is one of the more popular web caches currently deployed in caching architectures. Squid is designed to be used in a hierarchical mesh of cooperating caches. Caches utilize siblings and parents in order to satisfy requests. Squid provides static implementation of user policies. Configuration files allow the cache administrator to make modifications of runtime settings. This can be used to provide multiple hierarchies for documents from different domains.

An alternative for communicating between caches and organizing the hierarchies is discussed by Zhang et al. [21]. They present an adaptive technique for organizing caches that make use of multicasting cache query messages.

Push-caching [7] or server dissemination [1] is a server-driven technique for caching web objects. The origin server contacts caches, performing the tasks of locating objects, maintaining coherency and removing objects from caches. Push caching allows servers to set the policies of files, however it requires the server to negotiate resources with caches and maintain state about which caches maintain copies of documents.

6 Summary and Future Work

Caching systems are beginning to become common, however information producers and cache administrators are often limited in the policies that can be applied to the web objects that are cached. This paper describes the CacheL language and modifications to caches in order to provide customizable policies for web pages.

Currently, we have designed the language specification for CacheL. The next step will be to implement CacheL using an interpreter and modify an existing cache in order to provide customization of user policies. The performance of a working implementation will be examined.

```
#
# Variables Used:
#
#     bSupport    : is hit-metering and usage-limiting supported
#     bDoMeter    : were we instructed to do metering
#     bDoLimit    : were we instructed to do usage limiting
#     cu, cr      : usage counters (cu=use, cr=reuse)
#     tu, tr      : counters for usage-limiting
#     mu, mr      : limits placed on usage (mu=use, mr=reuse)
#
# All variables are scoped to be associated with a web object

Event: Route( type )
   if ( type == "Fetch" )
      Request-Mime-Add( "Meter:" )
      Request-Mime-Append( "Connection:", "Meter" )
      CacheFetch( CurrentURL() )
   else if ( type == "IMS" )
      if ( bSupport )
         Request-Mime-Append( "Connection:", "Meter" )

         if ( bDoMeter )
            Request-Mime-Add( "Meter: will-report-and-limit, count = " +
               cu + "/" + cr )
            cu = cr = 0
         endif
      endif
      CacheFetchIMS( CurrentURL() )
   endif

Event: New-Store
   if ( Mime-Exist( "Connection:", "meter" ) )
      bSupport = true
      if ( Mime-Exist( "Meter:", "dont-report" ) ) bDoMeter = false endif
      if ( Mime-Exist( "Meter:", "max-uses" ) )
         bDoLimit = true
         tu = tr = 0
         mu = MimeParse( "Meter:", "max-uses=%", "DIGIT" )
      endif
      if ( Mime-Exist( "Meter:", "max-reuses" ) )
         mr = MimeParse( "Meter:", "max-reuses=%", "DIGIT" )
      endif
   else
      bSupport = false
   endif


Event: Access-Inline( type )
   if ( bSupport )
      if ( bDoLimit )
         if ( type = "Fetch" ) tu++ endif
         if ( tu < mu ) CacheResponse( CurrentID() )
         else CacheFetchIMS( CurrentURL() ) endif
      endif
   endif

   CacheResponse( CurrentID() )


Event: Access-Offline
   if ( bSupport )
      if ( bDoMeter )
         if ( type = "Fetch" ) cu++ else cr++ endif
      endif
   endif

# Create a POST message to update the hit-counts
# that accessed this page
Event: Purge
   ...
```

Figure 6: CacheL actions to implement Hit-Metering and Usage Limiting

7 Acknowledgements

We would like to gratefully acknowledge Earl Barr, Brant Hashii, Kelsi King, Scott Malabarba and the anonymous reviewers for their valuable suggestions.

References

[1] Bestavros, A. WWW traffic reduction and load balancing through server-based caching. *IEEE Concurrency 5*, 1 (Jan-March 1997), 56–67.

[2] Breslau, L., Cao, P., Fan, L., Phillips, G., and Shenker, S. On the implications of Zipf's law for web caching. In *3rd International WWW Caching Workshop* (1998).

[3] Caughey, S. J., Ingham, D. B., and Little, M. C. Flexible open caching for the web. In *Proc. Sixth International World-Wide Web Conference* (Santa Clara, California, USA, April 7–11 1997), vol. 29 of *Computer Networks and ISDN Systems*, pp. 1007–1017.

[4] Chankhunthod, A., Danzig, P. B., Neerdaels, C., Schwartz, M. F., and Worrell, K. J. A hierarchical internet object cache. In *Proceedings of the USENIX 1996 Annual Technical Conference* (1996), pp. 153–63.

[5] Fan, L., Cao, P., Almeida, J., and Broder, A. Z. Summary cache: A scalable wide-area web cache sharing protocol. In *ACM SIGCOMM '98 Conference. Applications, Technologies, Architectures, and Protocols for Computer Communication* (Vancouver, BC, Canada, 2–4 September 1998), vol. 28.

[6] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., and Berners-Lee, T. Hypertext transfer protocol – HTTP/1.1. RFC 2068, UC Irvine, Digital Equipment Corporation, M.I.T., January 1997.

[7] Gwertzman, J. S., and Seltzer, M. The case for geographical push-caching. In *Proceedings Fifth Workshop on Hot Topics in Operating Systems (HotOS-V)* (1995), IEEE CS Press, pp. 51–55.

[8] Kistler, T., and Marais, H. WebL – a programming language for the web. In *7th International World Wide Web Conference* (Brisbane, Qld., Australia, April 14–18 1998), vol. 30 of *Computer Networks and ISDN Systems*, pp. 259–70.

[9] Korupolu, M. R., and Dahlin, M. Coordinated placement and replacement for large-scale distributed caches. http://www.cs.utexas.edu/-users/dahlin/papers.html, November 1998.

[10] Krishnan, P., and Sugla, B. Utility of co-operating Web proxy caches. In *7th International World Wide Web Conference* (Brisbane, Qld., Australia, April 14–18 1998), vol. 30 of *Computer Networks and ISDN Systems*.

[11] Luotonen, A., and Altis, K. World-wide web proxies. In *Computer Networks and ISDN Systems* (1994), First International Conference on the World-Wide Web, Elsevier Science BV. Available from: http://www.cern.ch/PapersWWW94/luotonen.ps.

[12] Mogul, J., and Leach, P. Simple hit-metering and usage-limiting for http. RFC 2227, October 1997.

[13] Rousskov, A., and Wessels, D. Cache digests. In *3rd Internation WWW Caching Workshop* (1998).

[14] NLANR hierarchical caching system usage statistics. http://www.ircache.net/Cache/Statistics/.

[15] Tewari, R., Dahlin, M., Vin, H. M., and ay, J. S. K. Beyond hierarchies: Design considerations for distributed caching on the internet. Tech. Rep. TR98-04, The University of Texas at Austin, 1998.

[16] Wessels, D. Squid internet object cache. http://squid.nlanr.net/.

[17] Wessels, D. Intelligent caching for world-wide web objects. In *Proceedings INET '95* (Honolulu, Hawaii, June 27–30 1995).

[18] Wessels, D., and Claffy, K. ICP and the squid web cache. *IEEE Journal on Selected Areas in Communication 16*, 3 (April 1998), 345–357. Available from: http://ircache.nlanr.net/ wessels/Papers/.

[19] Williams, S., Abrams, M., Standridge, C. R., Abdulla, G., and Fox, E. A. Removal policies in network caches for world-wide web documents. In *ACM SIGCOMM '96 Conference Applications, Technologies, Architectures, and Protocols for Computer Communications* (October 1996), vol. 26, ACM, pp. 293–305.

[20] Wooster, R. P., and Abrams, M. Proxy caching that estimates page load delays. In *Computer Networks and ISDN Systems* (September 1997), vol. 29 of *Sixth International World Wide Web Conference*, Elsevier, pp. 977–86.

[21] Zhang, L., Michel, S., Nguyen, K., Rosenstein, A., Floyd, S., and Jacobson, V. Adaptive web caching: Towards a new caching architecture. In *Third International Caching Workshop* (June 15–17 1998).