Active Cache: Caching Dynamic Contents on the Web

Pei Cao, Jin Zhang and Kevin Beach
Computer Sciences Department
University of Wisconsin-Madison
1210 West Dayton Street
Madison, WI 53705, USA
{*cao,zj,kbeach*}@*cs.wisc.edu*

Abstract

Dynamic documents constitute an increasing percentage of contents on the Web, and caching dynamic documents becomes an increasingly important issue that affects the scalability of the Web. In this paper, we propose the Active Cache scheme to support caching of dynamic contents at Web proxies. The scheme allows servers to supply cache applets to be attached with documents, and requires proxies to invoke cache applets upon cache hits to furnish the necessary processing without contacting the server. We describe the protocol, interface and security mechanisms of the Active Cache scheme, and illustrate its use via several examples. Through prototype implementation and performance measurements, we show that Active Cache is a feasible scheme that can result in significant network bandwidth savings at the expense of moderate CPU costs.

Keywords

World-Wide Web, caching, scalability, Java applets, network security

1 INTRODUCTION

The growth of the Internet and the World Wide Web has significantly increased the amount of online information and services available to the general population of the society. However, the client/server architecture employed by the current Web-based services is inherently un-scalable. Caching at proxies that are located at network access points has been proposed as a solution to the scalability problem. Unfortunately, caching in the Web today has been seriously limited by the inability to cache dynamic documents, including documents that change upon every access, documents that are results of queries, and documents that embody client-specific information (for example, cookies). Furthermore, lack of server control on accesses to cached documents results in the "caching busting" practice from the servers, severely undermining the scope of cacheable documents.

To solve these problems, we propose a scheme called Active Cache, which migrates parts of server processing on each user request to the caching proxy in a flexible, on-demand fashion via "cache applets." A cache applet is a server-supplied code that is attached with a Universal Resource Locator (URL) or a collection of URLs. The code is typically written in a platform-independent programming language such as Java When caching a document, the proxy also fetches the corresponding cache applet. When a user request hits on the cached copy and the proxy would like to service the request, the proxy must invoke the cache applet with the user request and other information as arguments. The cache applet then decides what the proxy will send back to the user, either giving the proxy a new document to send back to the user, or allowing the proxy to use the cached copy, or instructing the proxy to send the request to the Web server. Furthermore, the applet can deposit information in a log object, which is sent back to the server periodically and when the applet or the document is purged from the cache.

Cache applets allow servers to obtain the benefit of proxy caching without losing the capability to track user accesses and tailor the content presentation. They can perform a variety of functions, for example, logging user accesses, rotating advertising banners, checking access permissions, constructing client-specific Web pages, etc. They also enable proxies to be more than just caches of static information, but rather caches of objects, i.e., data with a method that is invoked when the data are supplied from caches. In essence, they turn Web documents from *datagram* to *objects*.

The proxy, when caching a document with cache applets, has the full freedom to not invoke the applet but send the user request directly to the server. *The proxy promises to not send back a cached copy of a document without invoking the corresponding cache applet*. On the other hand, if a document is cached but the corresponding applet consumes too much resource, the proxy can simply send the request to the Web server. Furthermore, just as the proxy is not obligated to cache any document, it is not obligated to cache any applet. The proxy agrees to not service a cache hit if the corresponding applet is not in cache.

The proxy's freedom of managing its own resource and the association between cache applets and URLs allow an on-demand migration of server functionalities. The proxy only fetches and executes the applet when there are user requests to the associated URL. Controlling its own resource allocation, the proxy can devote resources to the applets associated with the hottest or the most important URLs to its users. Since the proxy that receives the user request is typically the closest proxy to the user, the scheme automatically migrates the server processing to nodes that are close to users, significantly increasing the scalability of Web-based services.

In this paper, we describe the protocol and interface between the server-supplied cache applets and the proxies in the Active Cache paradigm, and the security mechanisms to guard the applets' execution and protect the proxy. We then give examples of cache applets to illustrate the Active Cache paradigm, and discuss proxy's resource management policies. Finally, we report our experience with an early Active Cache prototype, focusing on the overhead incurred by cache applets.

2 THE ACTIVE CACHE PROTOCOL

In Active Cache, the Web server specifies the association between a cache applet and a URL-named document by sending a new entity header, "CacheApplet," with the document:

- CacheApplet: code = "code.class", archive = "code.jar", codebase = "codebase_url"

The header follows the convention for applet specification in HTML documents. It specifies that the applet "code.class" at code base "codebase_url" is the cache applet for the document, and associated classes are grouped in an archive (4). Codebase and archive directives are optional. Since HTTP/1.1 allows the introduction of new entity headers, and requires that if a proxy does not recognize an entity header, it should forward the header, the server can be assured that Active-Cache enabled proxies will receive the header even if they have parent proxies.

For security concerns, we require that the codebase of the applet, if present, has the same server URL as the document. That is, only the Web server who provides the document can supply the applet for it.

An Active-Cache enabled proxy agrees to fulfill the following obligations:

- if the document is cached and a user request hits on the document, it will either invoke the cache applet, or send the request directly to the server.
- if the applet's execution fails due to any reason, the request is sent to the server;
- if the applet's execution succeeds, the proxy will take the appropriate action based on the return value of the FromCache method (see below);
- each applet can deposit information in a special log object (whose name is the name of the applet appended by ".log"), and the proxy will send the log object back to the server periodically. If the proxy evicts the log object from the cache, it will first send the object to the server.

In other words, the proxy will not return a cached reply to the user unless the cache applet has been executed successfully, and the applet can deposit information in the log object which will eventually be reflected back to the server.

However, the proxy decides whether it wants to cache the document, when it fetches the applet and the archive, and whether it wants to invoke the applet. Furthermore, the proxy can evict any document or any applet from its cache at any time. The only constraint on the proxy is the above agreement.

The proxy's freedom means that the cache applet may not process every request to the document, and some requests might instead go to the original server. The applet does not capture every request to the document, but captures every "cache hit" to the document.

2.1 Active Cache Interface

The cache applet must implement an interface called "ActiveCacheInterface." Currently, we require that the applet be written in Java. The interface defines a function called FromCache. The function is called when an access hits in the document, and the arguments of the function call include the user request, the IP address of the client machine, the name of the client machine, the document in cache, and a new file descriptor for the applet to write when constructing a new Web page. The prototype of the function is listed below:

- public abstract int FromCache(String User_HTTP_Request, String Client_IP_Address, String Client_Name, int Cache_File, int New_File);

The arguments are included based on the principle that any information that the server can gather if the request goes to the server can be seen by the cache applet. The function can only return three values:

- 1: the content placed in New_File is to be returned to the user as the reply to the HTTP request;
- 0: the content in Cache_File can be returned to the user as the reply;
- -1: the request must be forwarded to the server.

All other return values are treated as -1.

The cache applet can only call the ActiveProxy class to perform its functions. The ActiveProxy class provides the native methods for file access , cache query, locking and unlocking as well as sending requests to servers. Currently, the methods include:

- public native static boolean is_in_cache(String URL);
- public native static int open(String URL, int mode);
- public native static int close(int fd);
- public native static int create(String URL, int mode);

- public native static int read(int fd, byte[] buf, int size);
- public native static int write(int fd, byte[] buf, int size);
- public native static long lseek(int fd, long off, int where);
- public native static int send_request_to_server(String HTTP_Request);
- public native static int lock(int fd);
- public native static int unlock(int fd);
- public native static String curtime();

As we gain more experience with the Active Cache paradigm, other necessary methods might be added to the ActiveProxy class.

An applet's execution can be aborted at any time. All changes to files and objects are voided if the execution is aborted. In other words, file changes are not committed until the applet finishes its execution.

The restrictions on the use of the proxy calls are the following:

- is_in_cache, open for read, lseek, read and close can only be called on URL-named documents that are from the same server. In other words, the proxy verifies that the server URL of the document is the same as the server URL of the cache applet;
- open for write, create, write, lock and unlock can only be applied to URL-named objects that the applet has created; the create call automatically appends the applet's URL to the object's name;
- send_request_to_server can only send HTTP requests to the Web server; the method automatically connects to the server where the cache applet comes from.
- curtime provides a timing granularity of one second.

Furthermore, all exceptions are caught and result in a return value of -1 for the FromCache method.

Objects created by an applet (including the log object) are invisible to the outside world except to the original server that sends the applet. A server can gather the log objects of its applets through the Internet Cache Protocol (ICP) (3). ICP is originally designed for cache sharing and serves as a convenient channel for servers to gather log objects.

In addition to the ActiveProxy methods, the only other packages present in the Java run-time environments are java.util, java.text, java.math, java.security and java.sql. All other packages, including java.io, java.lang, and java.systems, are simply not loaded. The proxy can do so because the cache applet has only one purpose, that is, performing per-request processing of the document, and thus needs a very simple security interface.

2.2 Cache Applet Examples

The Active Cache protocol allows a rich variety of processing to be accomplished at the proxy system. Below, we describe the applets we have implemented.

2.2.1 Logging User Accesses

One of main reasons many Web servers disable caching is to collect information on who access their documents. Both server and proxy suffer from such practice, as the former has to buy a lot of resources to handle the volume of incoming requests, and the latter has to pay for the Internet traffic. Active cache solves the dilemma by using a log applet.

We have implemented a log applet, whose FromCache method simply writes the client IP address, the date of the access, and the HTTP request to the log object. The applet is assured that the log object will be sent back to the server eventually. The Java code looks like the following:

```
  public static int FromCache(
String User_HTTP_Request,
String Client_IP_Address,
String Client_Name,
int Cache_File,
int New_File) {

int fd = open("logapplet.log", APPEND_ONLY);
int status = lock(fd);
String date = new curtime();
log_to_file(fd, date, Client_Name,
User_HTTP_Request, Client_IP_Address);
status = unlock(fd);
close(fd);

return(0); // 0 means use the cached file
}
```

2.2.2 Advertising Banner Rotation

Another reason that servers disable caching is to change the presentation of the document upon every request, for example, putting on different advertising banners. This again conflicts with proxy caching. Active Cache solves the problem by attaching an ad-banner-rotation applet with each document.

The applet, when invoked, first checks for the object that specifies the banners, their positions in the document, and their frequencies of appearance. If the object is not in cache, it sends a request to the Web server to fetch the object. The applet then goes through the cached document, and for every image that is specially marked to be an advertisement banner, decides which banner should be put there according to the specifications, and changes the image URL. It then puts the new document in the New_File and returns 1.

The applet implements a simple frequency-based rotation. Other algorithms can be implemented. The applet can record the state needed by the algorithms in the object. It can further record in the log object the banner choices that it has made.

2.2.3 *Access Permission Checking*

Existing cache systems for the Internet typically provide very limited support to access control of server contents: for example, only allowing the same user to access the document again. Using cache applets, the server can gain the benefits of proxy caching without sacrificing the access control.

The cache applet, upon receiving a request from the user, checks whether the user request is accompanied with an access authorization. An access authorization is a cookie that contains a signed statement from the server. If not, the request is sent to the server (whose response would include a cookie for future requests if the user is allowed to access the contents). Otherwise, using the server's public key, the applet verifies whether the server has signed the certificate. If so, the applet grants access to the document. If not, the applet merely returns -1 and the request is redirected to the server, who will send the appropriate access violation messages.

2.2.4 *Client-Specific Information Distribution*

Today, many information providers allow users to specify preferences of the categories of information that they would like to receive. A typical example is `my.yahoo.com`, which is among the busiest sites on the Internet. The client-specific pages currently cannot be cached at the Web proxies, increasing the load at the Web server and the traffic on the Internet. Active cache solves the problem again by using a cache applet that constructs client-specific pages based on a database of base documents.

We have implemented a simply cache applet for this purpose. Upon receiving the client request, the applet first probes a database object to see if it stores the mapping between the client ID (extracted from the cookie) and its preference. If not, it fetches the preference from the server. After obtaining the preference, the applet composes the Web page. For each individual information item, it first tries to read the item from the cache, and if the item is not cached, fetch it from the server and cache it. It then composes the page and returns it to the user.

Thus, the cache applet filters out the redundancy in the information transmitted by the server for the client-specific pages, and allows individual information items to be cached and reused by the proxy. For a proxy with a large client population, the savings in network bandwidth can be significant. It also allows schemes such as Pointcast to avoid having to write its own proxy servers, and support broadcasting schemes such as SkyCache.

2.2.5 *Server-Side Include Expansion*

Similar to the ad-banner-rotation applet, the SSI applet allows expansion of Server-Side Include (SSI) (5) variables at the proxy site, thus allowing the correct caching of SSI-based dynamic documents. The applet scans the cached document, and for each specially marked SSI variable, update the value of the variable in the document, and put the new document in New_File. The proxy can correctly expand variables DATE_GMT, DATE_LOCAL, REMOTE_ADDRESS, and REMOTE_HOST. Other

SSI variables are typically related to the server and do not change from request to request.

2.2.6 Delta Compression

Studies (8) have shown that transmitting the changes (deltas) between the new and old versions of dynamic information can reduce network traffic significantly. Delta compression can be easily implemented with cache applets.

We have implemented an applet that upon receiving a client request, sends a request to the original server, including the current request and the last-modified-time of the document. After the server responds with the difference between the new document and the cached version, the applet constructs the reply to the client request using the diff and the cached version.

A similar cache applet can support delta compression of query responses. For example, a query of a particular company's stock quote can be handled by the applet, which simply asks the server for the number and then composes the Web page based on a cached response.

3 SECURITY MECHANISMS

In the Active Cache environment, the proxy system is particularly vulnerable to two types of security attacks: an applet's illegal access to information belonging to other Web servers, and denial-of-service attacks. To guard against the first type of attacks, we rely on a type-safe language with built-in security mechanisms, a well-defined security interface, and static examination of the code. To guard against the second type of attacks, we rely on user-request triggered execution of cache applets, run-time accounting of resource consumptions, and limiting the resource consumptions.

3.1 Language-based Protection

For the language of the cache applet, we choose Java because of the large amount of research efforts invested on Java security. Its platform-independence and built-in security mechanisms contribute to our decision. In the Active Cache environment, the Java security problem is simplified by the fact that the applet is used for only one purpose: processing a user request at the proxy site and constructing a response. Thus, the java.io libraries, the process/thread management libraries and other system-related libraries are simply absent from the cache applet's run-time environment to prevent the security problems associated with their use.

The security interface of the cache applets is quite simple: all an applet can do is to ask whether some objects exist in the cache, read and write the objects, and communicate with the server where the applet comes from. Furthermore, accesses to objects are restricted as described in Section 2.

The restrictions are designed with the assumption that each server is a security entity, applets from the same server trust each other, and applets from different servers do not cooperate. Thus, an applet has read access to all information in documents and objects from the same server. However, each applet can only manipulate the objects that it creates or is attached to. We impose this constraint for simplicity and safety. The flexibility of the scheme is not compromised, because one applet can always be attached to many documents.

The restrictions are enforced by two mechanisms. The ActiveProxy class implements the constraints. Java's type safety and run-time mechanisms prevent applets from bypassing the ActiveProxy class and gaining raw access to information and resources. Recent research results have significantly improved the robustness of Java's run-time environments (10). Thus, we rely on the existing mechanisms to force the applet to use the ActiveProxy class to access its objects as well as the computation and networking resources. The ActiveProxy class is also the place where we keep track of resources consumed by each applet.

In addition, we use static examination of the classes and functions called in the cache applet code to prevent manipulations of Java "class" methods to circumvent our restrictions. When the cache applet is first loaded, the proxy examines the symbol table to check for "class" method calls and calls to unloaded packages. If the applet cannot pass the inspection, the proxy will not cache the reply.

3.2 Resource Accounting

The proxy keeps track of an applet's resource consumption in the following five aspects:

- storage size, that is, the sum of sizes of objects created and written to by the applet;
- disk bandwidth consumption, that is, the amount of file bytes read and written by the applet;
- network bandwidth consumption, including all the bytes sent and received in the Send_HTTP_Request function;
- CPU usage, including user time and system time;
- virtual memory size (an applet can exhaust kernel page table resources by allocating a large amount of virtual memory; thus, the size of the virtual address space needs to be constrained);

Mutexes and locks are not included because each applet can only lock objects created by it, and all locks are automatically relinquished upon termination of the applet execution. We also do not track process-related operations because the applet cannot spawn new threads or processes. Rather, each applet is executed in a new process when a user request for the active object arrives.

The storage size, disk bandwidth and network bandwidth consumptions are kept track of by the ActiveProxy class methods, since they must be called in order for the applet to gain access to those resources. The process running the applet also sets a one-second alarm and record the CPU time and virtual memory sizes in the alarm handler. Limiting the CPU and virtual memory sizes is implemented by the setrlimit system call before branching to the applet's execution.

To prevent denial of service attacks, the proxy imposes upper-limits on all five resources. By default, the upper limit for CPU time is proportional to the latency of sending the request to the server and receiving the response. The virtual memory size is proportional to the length of the response to the client request. The storage size and disk bandwidth limit are also proportional to the response size. Finally, the network bandwidth consumption cannot exceed the response size. The limits are designed with the assumption that the goal of caching the documents is to reduce network traffic. If the goal of caching is for reliability or other reasons, the limits can be raised by the proxy. In our initial prototype, we simply use fixed constants for these limits for ease of implementation.

4 RESOURCE MANAGEMENT POLICIES

An important design question for Active-Cache enabled proxies is the resource management policy. Essentially, the policy must make three decisions:

- should a document (with or without a cache applet) be cached?
- when a user request arrives, should the proxy invoke the applet or send the request to the server?
- what are the upper resource limits for each applet?

The decisions are made depending on the reason for the cache applet.

There are typically two reasons why a proxy wants to cache a document or object: to reduce outgoing network traffic, and to improve the availability of a distributed service. When the proxy's goal is to reduce network traffic, the proxy is willing to cache the most-frequently requested documents or objects, even if they are from untrusted servers. When the proxy's goal is to improve service availability, the proxy often knows about the service's importance to its users, trusts the server, and is willing to invest more resource to host it.

Thus, we have two categories of applet-attached documents (in other words, objects): un-negotiated ones and negotiated ones. Un-negotiated objects are from untrusted servers; their primary purpose is to perform processing at the proxy site to avoid network traffic to the server. Negotiated objects are from servers that go through a negotiation protocol with the proxy. They receive more resources and are cached at the proxy for as long as necessary.

Un-negotiated objects Resource management for un-negotiated objects is relatively straightforward. To decide whether an object should be cached, the proxy estimates the benefit and the cost of caching, and passes the information to the cache replacement module. A cost-aware cache replacement algorithm can be used to decide whether an active object is cached, for example, the GreedyDual-Size algorithm (2

). The benefit of caching is calculated in terms of saved network bandwidth, estimated by the size of the response to each client request. The cost of caching includes the storage cost of the active object, the CPU cost of the applet, and the network communication incurred by the applet in the send_request_to_server calls. When an active object is first loaded, all costs are assigned a default value. The cost estimates are then adjusted every time the applet is invoked.

The proxy always attempts to invoke the cache applet if the object is cached. However, if the CPU or the disk arm is overloaded during the execution of the applet, the proxy terminates the execution of the applet, voids all file changes made by the applet, relinquishes all its locks, and directs the user request to the server.

In general, the upper resource limits for un-negotiated applets should be kept proportional to the size of the responses to client requests, that is, the savings in the network bandwidth. The exact scaling factors still need investigation. As we build more applets on top of Active Cache, we expect to gain more insight into the issue.

Negotiated objects To establish a negotiated object, a server and a proxy should enter a protocol in which the server identifies the service, specifies the estimated storage, networking and CPU needs of the applet, and specifies the desired duration of caching. The proxy examines the amount of interests in the distributed service from its users, the importance of the service and the credential of the server, in order to decide whether to host the service. If the proxy hosts the service, it caches the object and the applet for the specified duration, always invokes the applet upon user requests, and imposes the resource upper limits as specified by the server.

We have not yet designed the negotiation protocol because we need more experience with applications that need it. In particular, it is not clear whether the proxy should provide support for real-time constraints, and to what degree. Full-fledged real-time support complicates the design of the proxy. It is possible that for most Active cache applications that need latency guarantees (for example, distributed collaboration and multimedia streaming), the proxy needs only guarantee prompt execution of the cache applet upon client requests and periodic execution of the applet (via an alarm service). We plan to evaluate various design choices as we develop those applications.

Currently, resource management policies in Active Cache are still research topics. In our current prototype, all documents are un-negotiated objects. We use the simple LRU policy in deciding which document to cache, regardless of whether the document has associated cache applet or not. We also impose fixed constants on the resource consumption of all cache applets. These choices are purely for the ease of implementation, and we plan to refine them soon.

5 PROTOTYPE IMPLEMENTATION AND PERFORMANCE

We have implemented a prototype Active Cache as an extension of the the CERN httpd proxy (6). The original CERN httpd software offers traditional caching of Web documents and HTTP protocol support. We have modified the daemon to recognize the CacheApplet header, and to invoke the appropriate applet upon cache hit.

A cache-applet attached document is stored as a regular document in the CERN proxy. The CacheApplet header is stored as part of the document. The header identifies the associated applet and archive. The CERN httpd proxy handles each user request in a separate process. (Despite its process-forking overhead, CERN httpd performs amazingly well compared to other sophisticated proxies (7).) The process model significantly simplifies our implementation, because we can limit the resource consumption of applets by using setrlimit calls prior to calling the applet. Sending the log object back to the server is implemented via a HTTP "POST" request to the server. If the server is unreachable, the proxy retries the transmission periodically up to a configurable time period, after which an alert message is sent to the proxy administrator and human intervention is required.

The prototype implements the active cache protocol and the security mechanisms described before. If an applet does not pass the static examination, both the document and the applet are deleted from the cache. All objects created by the applet are stored in a special directory whose name is the applet's URL. The implementation of the write and lock methods limits the operations to the objects in the special directory only. The implementation of the read method verifies that the object has the same URL as the server URL of the applet. The Java run-time environment is set up with the appropriate security manager.

5.1 Applet Overhead

To measure the overhead incurred by the cache applets, we use the WebStone 2.0 standard Web server benchmark (9) and compare the response times of the original CERN httpd proxy and the Active Cache proxy with various cache applets. In each of our Active Cache tests, we assume that all the documents are associated with the same applet. We test the "null" applet, the user access logging applet "log," the advertising banner rotation applet "ads," the access permission checking applet "apc," the server-side include expansion applet "ssi," and the client-specific information distribution applet "csid."

Though WebStone is a Web server benchmark, it can be used to test the performance of the proxy upon cache hits of different sizes. We use the filelist.standard in the WebStone 2.0 benchmark and create five files of size 500B, 5KB, 50KB, 500KB and 5MB. The clients request the files with the specified frequency in filelist.standard. The proxy machine is installed between the clients and the server. Since there are only five files involved, once the proxy cache is warmed up, all requests are cache hits and the test stresses the proxy system, instead of the server system. We run experiments with 1, 10 and 20 WebStone clients. The experiments all last over 10 minutes to obtain stable results. Our test platform includes SPARC 20 workstations running Solaris 2.4 as the client and the server machines. Our proxy machine for the "null," "apc," "log," and "ssi" applets is a 99MHz Intel x86 workstation running Solaris 2.5. Our proxy machine for the "ads" and "csid" applets is a SPARC20 running Solaris

Proxy	1 client	10 clients	20 clients
null	1.47	1.75	1.73
log	2.16	2.31	2.24
ads	1.40	2.44	3.23
apc	2.50	2.73	2.72
ssi	3.81	4.00	3.80
csid	1.06	2.04	2.23

Table 1 Ratio between the response time when the cache applet is used and the response time under the original CERN httpd proxy.

2.6; due to a bug in the Java thread library on the Intel platform, these two applets hang on the Intel platform. We are working on resolving the problem.

Table 1 lists the response time degradation of each applet, that is, the ratio between the client response time when invoking the applet and the client response time under the original CERN httpd. The "null" applet result shows that the mechanics of establishing a Java virtual machine and invoking the applet costs 47% to 75% degradation in response time. There are a number of reasons for it, including the increase in the process image and the corresponding increase in the forking cost, and the CPU overhead for finding the class, finding the method, and invoking the method. The other applets increase the client latency by a factor of 1.5 to 4.

Monitoring the proxy system using "vmstat" shows that the performance degradation is mostly caused by CPU overheads. In particular, the "ads" applet and the "ssi" applet incur CPU overhead that is proportional to the document size because they scan the cached document. The CPU overhead appears to heavily depend on the coding of the applet and the efficiency of Java implementations, particularly string operations. As the speed of Java improves and as we fine-tune our applet implementation, the CPU overhead will be reduced.

The measurements show that for a proxy system to support the Active Cache protocol and yet maintain the same throughput, its CPU resource needs to be increased. Fortunately, the workload is easily parallelizable and multi-processor systems or clusters of workstations can improve the throughput significantly. In most proxy systems today, disk arms and network connections are typically the bottlenecks, not the CPU resources. Whether the Active Cache paradigm will change the situation remains to be seen.

To summarize, our prototype implementation shows that Active Cache is a practical and feasible scheme to implement in proxies. It increases the CPU demands, and in a sense, trades local CPU resources for network bandwidth savings. Given that in today's technology, microprocessors are typically the cheap resources, the tradeoff is well worthwhile. We believe that the benefits of Active Cache greatly outweigh its implementation cost, and every proxy should support the Active Cache protocol.

5.2 Internet Traffic Reduction through Active Cache

In today's bandwidth-hungry Internet, Active Cache's trading CPU for Internet traffic reduction is well worthwhile. Companies typically spend $6,000 to $10,000 *per month* for a T1 line to the Internet; a PC workstation that can function as a proxy costs less than $2,000 today.

Compared with traditional caching, Active Cache can increase cache hit ratios significantly. One source of the increase is the widespread use of cookies. Studies of HTTP traces from the AT&T WorldNet Internet Service Provider (1) have revealed that 30% of all user requests carry cookies. Traditional proxies cannot cache replies to these requests because doing so would violate the semantics of the HTTP/1.0 protocol. As a result, they can only achieve object hit ratios around 35% and byte hit ratios around 30%. Active Cache proxies, on the other hand, can cache replies to requests carrying cookies with the help of server-supplied cache applets, and can achieve object hit ratios around 55% and byte hit ratios around 41%, according to the AT&T trace study.

Another source of the increase is the popularity of user-tailored information providers, such as "my.yahoo.com." These information providers allow a user to choose among a collection of news and information items to be included in a customized Web page. The number of items is typically less than 1000, and each item is typically less than 1KB. Active cache proxies, at the help of applets from these Web sites, can cache the individual information items and provide the customized pages from the cache. Since each customized page is typically over 20KB, as long as the proxy has over 100 users, the cache applets will cut more than half of the traffic to these Web sites.

Though it is difficult to perform trace-driven studies of Active Cache since most proxy traces anonymize the URLs, existing studies do demonstrate the need and the potential benefits of Active Cache. Given the sharp contrast in the cost of computing versus the cost of wide-area communication, Active Cache will be an integral component of future Web caching technologies.

6 RELATED WORK

Many prior research projects influence the design of the Active Cache scheme, in particular, mobile objects and agents. The evolution of Java as a mature language for mobile code makes the scheme possible.

Cache applets are similar to regular browser applets and servlets[4]. Compared with regular browser applets, cache applets have a simple, uniform security interface, which greatly simplifies the security problem. Compared with servalets, cache applets run at proxy sites as "guests" and face many resource constraints.

There are many studies on Web proxy caching. However, few have addressed the caching of dynamic contents. Robinovich et al have proposed macro-encoded HTML documents that are expanded at the browser site[1]. The work is similar to ours in the sense that both try to move the work away from the server. Indeed, a cache applet can easily implement the processing for macro-encoded documents. Compared with macro-ended HTML, cache applets are more flexible and powerful, and can support functions such as access control.

Finally, the principle of Active Cache is very similar to Active Networks, in that both treat packets as objects instead of datagrams. The difference is that Active Cache focuses on Web caching and proxies, while Active Networks focus on Internet evolution and routers. Proxies typically have full-fledged operating systems and less stringent performance requirements than routers, making Active Cache a somewhat easier research problem.

7 CONCLUSION AND FUTURE WORK

We propose the Active Cache protocol to support caching of dynamic documents on the Web. We have described the motivation behind the protocol, its design, interface, security mechanisms and resource management strategies. Using examples, we illustrate the flexibility and the potential of the scheme. Using prototype implementation and WebStone-based performance measurement, we show that cache applets typically increase the client latency by a factor of 1.5 to 4, and the degradation is mainly due to CPU overhead.

Much future work remains. We are currently extending Active Cache to support caching continuous media in the proxies. In particular, we are investigating cache applet implementations of RTSP, a protocol for transmitting continuous media on the Web. We are also extending Active Cache to support the notification protocol NTSP and its applications. Another important area that we are currently working on is resource management policies in Active Cache proxies. We are investigating appropriate resource limits and negotiation protocols, and the performance of cost-aware cache replacement algorithms. Finally, we are investigating ways to optimize Active Cache implementations and cache applets.

REFERENCES

[1] Ramon Caceres, Fred Douglis, Anja Feldmann, Gideon Glass, and Michael Rabinovich. Web proxy caching: The devil is in the details. In *Proceedings of 1998 SIGMETRICS Internet Server Performance Workshop*, June 1998. http://www.research.att.com/ misha/.

[2] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, December 1997.

[3] National Lab for Applied Network Research. Icp working group.

http://ircache.nlanr.net/Cache/ICP/, 1998.

[4] Javasoft Inc. Java applets, servlets and the jar guide. *http://www.javasoft.com/products/*, 1997.

[5] Netscape Inc. Generating dynamic html documents. *http://www.netscapeworld.com/nw-05-1997/nw-05-clue.html*, 1997.

[6] A. Luotonen, H. Frystyk, and T. Berners-Lee. CERN HTTPD public domain full-featured hypertext/proxy server with caching. Technical report, Available from http://www.w3.org/hypertext/WWW/ Daemon/Status.html, 1994.

[7] Carlos Maltzahn, Kathy Richardson, and Dirk Grunwald. Performance issues of enterprise level web proxies. In *Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modelling of Computer Systems*, pages 13–23, June 1997.

[8] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta encoding and data compression for http. In *Proceedings of ACM SIGCOMM'97*, August 1997. Available from http://www.research.att.com/ douglis/.

[9] Gene Trent and Mark Sake. WebSTONE: The first generation in HTTP server benchmarking. Technical report, MTS, Silicon Graphics Inc., February 1995. available from http://www-europe.sgi.com/TEXT/ Products/WebFORCE/WebStone/paper.html.

[10] Dan S. Wallach, Dirk Balfanz, Drew Dean, and Edward W Felten. Extensible security architecture for java. In *The 16th Symposium on Operating System Principles*, May 1997.

8 BIOGRAPHY

Jin Zhang received his bachelor degree of Computer Science in 1992 from Tianjin University, PRC. He received his Master's degree of Computer Science in 1995 from Tsinghua University, PRC. Currently he is a graduate student of Computer Science Department of University of Wisconsin-Madison. He joined the WisWeb group and worked with Professor Pei Cao since Sept. 1997. His main project is ActiveCache.

Pei Cao received her Ph.D. degree from Princeton University in 1995 and is currently Assistant Professor of Computer Science at University of Wisconsin-Madison. Her research interests are in World-Wide Web, operating systems and computer architecture. She currently leads the WisWeb research project which focuses on various aspects of Web caching. She is a member of the IEEE Computer Society Task Force on Internetworking.

Kevin Beach received his bachelor degree of Mechanical Engineering in 1997 from Rutgers University, USA. He is now a graduate student at Computer Science Department of University of Wisconsin-Madison. He joined the WisWeb research group in January 1998. His current project is Multimedia over Active Cache.