DynCoDe:Dynamic Content Delivery for Internet Services

Kalyan Pucha[1], Himabindu Pucha[2] and Saumitra M. Das[3]

[1]Dept of IT, PVPP CoE, University of Mumbai, India[1]

[2,3]School of ECE, Purdue University, West Lafayette, USA[2,3]

Email: {kpucha@yahoo.com[1], hpucha@purdue.edu[2], smdas@purdue.edu[3]}

Abstract

Web services on the Internet are increasingly relying on personalized content that is dynamically generated by server application code and customized for users. Delivering such personalized content increases computational load on servers and does not fit into the current Internet web caching model leading to an increase in user latency and bandwidth consumption. In this paper, we propose DynCoDe, a novel architecture for efficient delivery of personalized web services that integrates the distribution, caching and generation of personalized content. In the DynCoDe architecture, resource intensive processes for content generation and reusable content components are pushed to the network edges increasing server scalability and content availability while reducing user latency and backbone Internet traffic. We evaluate DynCoDe under real world network conditions and show significant improvements in bandwidth consumption, user latency and server scalability.

1. Introduction

The growth of the World Wide Web has resulted in innovation in the content served to the users. Web content can be broadly classified into static content and dynamic content. Static content refers to web content that is non-volatile and served to requesting users without changes. Dynamic content refers to web content that is generated on demand and may have different attributes depending on the requesting user. In the past, web traffic has been predominantly static in nature. Several studies have been proposed to serve such static content to users in an efficient manner. Web caching is one such widely studied solution useful for reducing server overloading, network congestion and user latency by pushing content close to the edge [1].

Current web traffic has an increasing amount of dynamic content due to the growing popularity of personalized web portals and services. Personalized web portals allow users to set up user profiles subscribing to content items of their interest to be presented in the specific layout that they would like. Additionally, the advertisements on these web pages are very specific and tailored based on the user profiles. This personalized content makes up a significant fraction of the total dynamic content. Such content by its very nature is un-cacheable. User requests for personalized pages typically bypass web caches and require the server to execute application code for every user request placing a large computational burden on the server. Additionally, the average size of a personalized web page is much larger than the average size of an ordinary web page and thus consumes more network bandwidth. Thus, dynamic content delivery places computational burdens on the servers, increases network congestion and increases user latency. It has also been shown that Content Delivery Networks like Akamai do not perform as well for dynamic content [8]. Thus, serving such dynamic content efficiently is an important research problem.

In this paper, we propose and evaluate DynCoDe[1], a generic framework for delivering dynamic content efficiently and transparently to end users. DynCoDe implements caching and generation of certain coarse grained dynamic content in a distributed manner on active proxies in the Internet. These proxies reside in the edge networks closer to the user, thus reducing the burden on the web server, increasing availability, saving valuable bandwidth and reducing user latency. Such active proxies could potentially be deployed by content providers (like CNN) or by Content Distribution Networks (like Akamai).

2. DynCoDe Architecture

As mentioned earlier, personalized pages such as (My Yahoo) are generated at web servers by content generating applications (CGI scripts, C programs, database software). These applications use methods such as cookies to uniquely identify a user, locate the user's profile and then dynamically generate the web content tailored to that user. A key observation here is that 60% of the components in such dynamically generated content have been shown to be reusable [2] and thus cacheable. DynCoDe proposes to push such reusable components of dynamic content to active proxies close to the user. However, this only reduces the network congestion and user latency but does not

[1] A poster describing DynCoDe was accepted at the 12th International WWW Conference, Hungary, May 2003

improve the scalability of the server as the server is still responsible for executing application code on each user request. Thus, DynCoDe also proposes that the applications used to generate the dynamic content be pushed to these active proxies. In order to support transparent pushing of content generation code to proxies, DynCoDe also provides an execution environment whereby on a user request, a proxy dynamically links and executes the appropriate content provider's code to generate the personalized content. Note that this necessitates protocols between servers and proxies to maintain consistency in the components being reused as well as the application code for generating the content. These issues are addressed by various features of the DynCoDe architecture. We now discuss the detailed design of DynCoDe with reference to the 3 major entities in the architecture: clients, servers and proxies.

2.1 Client Architecture

Since DynCoDe is designed to be transparent to clients, no changes are required of clients to benefit from the DynCoDe architecture. This makes the deployment of DynCoDe faster and more manageable.

2.2 DynCoDe Proxy Architecture

We denote the active proxies mentioned earlier as DynCoDe proxies. We assume that the DynCoDe proxy is able to receive and serve user requests prior to them reaching the server by using redirection mechanisms. The multi-threaded server process (see Fig. 1) in the DynCoDe proxy is responsible for serving user requests. In addition, the DynCoDe proxy also coordinates with the server in order to update and verify content generating application code as well as the reusable content components. The various modules in the DynCoDe proxy are described below and depicted in Figure 1.

Cache Manager Module: This module implements the cache consistency and replacement mechanisms. The proxy has a disk cache where reusable content components are stored as well as a cache in program memory to store the most recently accessed items from the disk cache. The memory cache is useful since I/O from disk is much slower than memory [3]. DynCoDe uses strong consistency semantics keeping in view the kinds of applications that could potentially use this service. For example, personalized sites like Yahoo, CNN and MSN serve continuously changing news in addition to relatively static content like comics. When a new story breaks, they would like it to reach the end user as soon as possible. Upon detecting a change in content, the server sends invalidation messages to all proxies that have recently accessed and cached the content. The cache manager in the proxies invalidates this content from the disk and memory cache preventing staleness. When an object in the cache is to be replaced, it calculates the rank of each of the objects presently residing in the cache using the following formula and replaces the objects with the smaller ranks.

$$Rank = F^f * R^r * S^s$$

F = frequency of access, f = positive value, R = Recency of access, r = negative value, S = Size, s = positive to favor bigger objects, negative to favor smaller objects

Content Module: The content module on the DynCoDe proxy consists of 2 application codes, an ad generator and a page assembler, both pushed by the server onto the proxy.

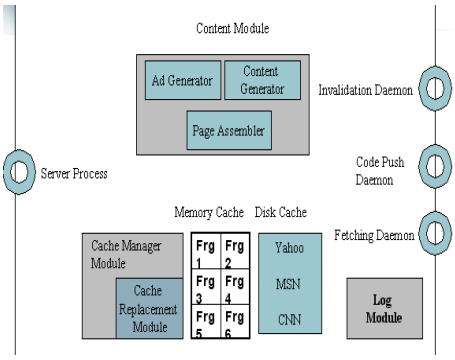

Figure 2: DynCoDe Proxy Architecture

The page assembler module (depicted in Figure 2) assembles a personalized web page for the client from the content and ad fragments. The content module take as input the user profiles and produces as output the user specific web page that is sent back to the client. The content module interacts with the fetching daemon and cache manager module to retrieve and store both content and user profiles and is implemented as dynamic libraries in C that are linked and invoked on a client request. The content module is served by various daemons: (1)The invalidation daemon listens for invalidation requests from the server and on receiving these invalidates the content both in the memory cache and on the disk, (2) The code push daemon listens for code push requests in order to retrieve the corresponding content generating application code and store it in the proxy replacing the older code and (3) The fetching daemon fetches the user profiles and content from the server when required based on the requests generated by the server code executing on the proxy.

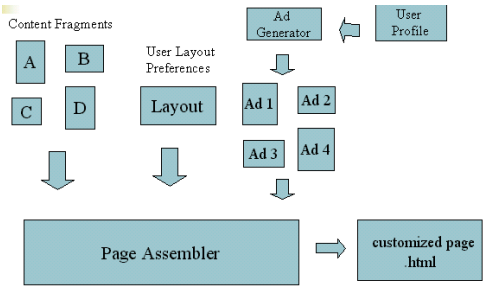

Figure 2: Content composition

Cache Organization: The DynCoDe proxy maintains a separate directory structure for each of the content providers it serves. An example directory structure for the portal MyYahoo is shown in Figure 3. The portal directory has two sub directories, one to store the content

components and one to store the code that would be executed to generate the user specific web page from the components. The same directory structure is also replicated on the server and maintained with strong consistency. Note that if the proxy serves multiple content providers each would have its own independent tree in the cache. Each DynCoDe proxy maintains two persistent connections with each content provider being served, one for fetching user profiles and content from the server and the other persistent connection to send invalidations and code from the server to proxy.

Log Module: This module keeps track of requests served by the cache for each of the content files as well as other statistics and periodically relays these to the server.

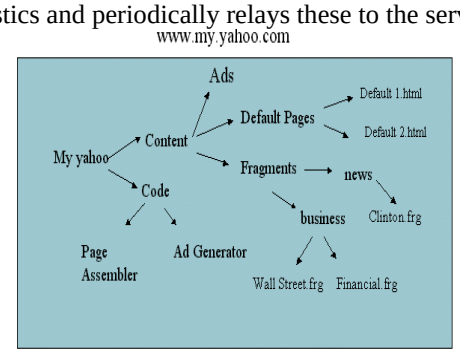

Figure 3: Disk Cache Structure

2.3 Server Architecture

The DynCoDe server is implemented as a concurrent multi-threaded server. It consists of many modules described below. The server actively pushes code to the DynCoDe proxies and typically only serves content to proxies that have not cached the content.

Content Consistency Module: This module keeps track, on a per proxy basis, all the content that it has sent to the proxies. When the content changes it goes through its data structures to see which proxies have the modified files cached on them and sends them a INVALIDATE request for that file. This maintains content consistency.

Code Push Module: When the content generating application code used to build personalized pages changes, the server pushes this new modified code to all the proxies that are serving it. For example, code changes may occur due to a change in content design or a new algorithm for displaying advertisements.

Cache Manager Module: The server optimizes performance and reduces disk I/O by maintaining data and profile caches shared across connections. The probability of collision in use of the cache is reduced by record level caching both in the profile and data cache.

Authentication Module: This module accepts and authenticates connections from DynCoDe proxies.

2.4 Operational Protocol

The DynCoDe proxies are configured with the hostnames of the content providers that they have to provide service for. The proxy establishes persistent TCP connections for each such newly added content provider server. It then

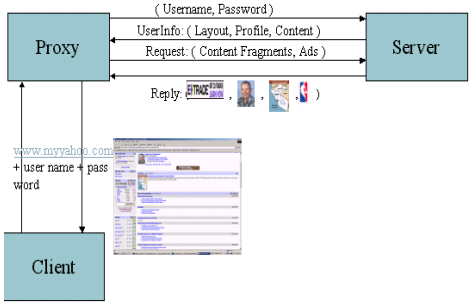

Figure 4: Operational Protocol

creates the appropriate directory structure for that content provider and downloads the application code, which is in the form of dynamic libraries, over the persistent connection from the content provider. The server uses one of the persistent connections to send invalidations and code to the proxy and the proxy uses the other persistent connection to fetch content from the server. Once the proxy downloads and links the application code, it is ready to accept client requests for that particular content provider.

The operational protocol is depicted in Figure 4. An initial client request to the server would contain the user name and pass word (e.g, using cookies) in order to authenticate the request and also be able to generate content specific to that particular user. Client requests are intercepted by the proxy which checks to see if it has already linked the application code of that particular content provider and if not, links it and invokes the function *buildPage* (part of the page assembler). The *buildPage* function in the dynamically linked library takes the authentication or identification information of the user as input. It then checks the cache (using the API provided by the Cache Manager) to see if the user profile has been cached either in the physical memory or on disk. If not then it fetches the user profile (using the Fetch daemon). The retrieved user profile is then passed to the content modules, which determine what ads and content are appropriate for the user. Both the physical memory and the disk are checked to see if they have a copy of the ads and the content and if not, it is fetched over the persistent connection with the web server. Subsequently, the Page Assembler, which now has all the appropriate ads and content for that particular user, constructs the dynamic page as per the layout preferences of the user and sends the customized page to the client browser. To maintain consistency, the server keeps track of the files cached by a particular proxy and sends invalidations to the nodes which have those cached. Invalidations are detected as resource changes and written to a global invalidation file. The server then sends the appropriate invalidation requests to the proxies over the

persistent connections and the proxy receives these messages and invalidates the content in the memory and removes the file from the disk.

2.5 Other Design Issues

Many web caches propose hierarchical structures. However, DynCoDe does not incorporate a caching hierarchy due to problems associated with them related to placement restrictions, queuing delays and redundancy of data [4]. A DynCoDe proxy node is a node placed close to the client and uses a distributed cache architecture [5]. We have not incorporated an inter-cache protocol such as CARP/ICP since it is likely that the DynCoDe node has a high probability of hits once it has achieved the user locality we want to provide. In this scenario, inducing more traffic on the network using inter-cache protocols and the associated protocol delay is unnecessary. For pricing, content providers need to access statistics. Current web caches hide client accesses to the content and hence prevent the collection of these statistics by content providers. Due to this, the providers are forced to mark content un-cacheable, thus making ISP caches unusable. In order to alleviate this problem, DynCoDe proxies support for pushing access statistics to the server. The DynCoDe proxy periodically sends back the logs that it stores and hence the server gets all the information that it needs to price its services and have incentive to make more of their content cacheable using the DynCoDe architecture.

3. Performance Evaluation

The advantages of caching, whether static or dynamic caching, are improved response time, bandwidth savings, better availability and increased server scalability. To measure the above parameters with respect to our architecture under real network conditions, we set up a test bed. The test bed consists of a content server running on a node at University of Illinois, Chicago and the DynCoDe proxy and the clients running in Pittsburgh at locations in Carnegie Mellon University and a residential colony. To simulate the clients we use Apache Bench, which produces a steady stream of requests. Earlier studies have observed that the number of items in a typical personalized page were less than 1000. Each item was typically less than 1KB and also each customized page was typically over 20KB. We in our own measurements have noticed the typical personalized MSN and MyYahoo pages have an average size of about 100 KB mainly due to presence of images. So we took measurements for file sizes ranging from 10KB to 100 KB and also for an invalidation rate of 1 minute for each of the objects i.e. a new version of each object comes out every 60s. The very high refresh rate exhibited by content such as stock tickers was used to measure the worst-case performance of the DynCoDe architecture, because due to the high invalidation rate it now has to fetch the records from the server much more often decreasing the bandwidth savings and degrading the response rate.

3.1 User Response Time

As can be seen from Figure 5, the response time to the clients is cut by at least half in the DynCoDe architecture. This happens across different page sizes as well. This is due to the fact that the proxies are placed at the access networks closer to the clients than the server, communicating with which would require going across the WAN and the delay incurred thereof. The reduction by half in the response time is impressive in view of the high invalidation rate; each object is invalidated every minute. If the invalidation rate were assumed to be more reasonable, as in the real world, then less content would be required to be fetched from the server thus improving the response time even further.

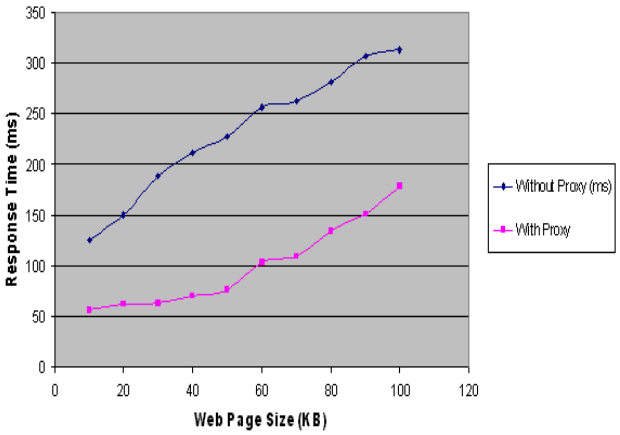

Figure 5: Response Time

3.2 Bandwidth Savings

For calculating the bandwidth savings we assumed that the DynCoDe proxy would be serving only a modest 10 requests for the personalized pages each second. We plotted the bandwidth savings vs. the invalidation rate as shown in Figure 6. Bandwidth savings are calculated as the amount of traffic that is not sent over the WAN due to the use of DynCoDe. Intuitively as the invalidation rate decreases the bandwidth savings would increase, as most of the content in the cache could be used in building the customized pages and less content needs to be fetched from the remote web server. Significantly even a high invalidation rate of 1 minute for each of the objects in the cache leads to 83% savings in bandwidth. A more realistic assumption that content changes only every 5 minutes leads to a 97% increase in bandwidth savings. These savings in bandwidth would translate into monetary savings that would otherwise have been invested in provisioning the bandwidth between the clients and the web servers.

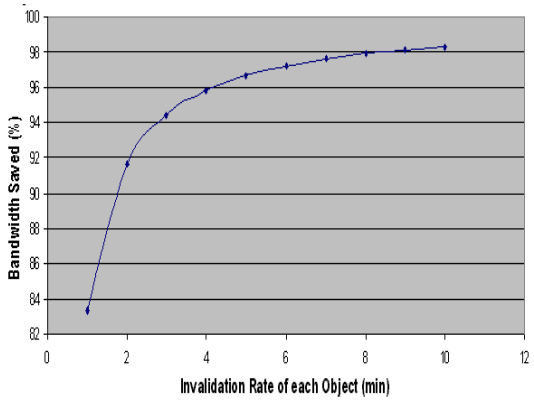

Figure 6: Bandwidth Savings

3.3 Server Scalability

From our measurements we have seen that maintaining the per-proxy state consumes very little CPU time compared to actually processing the dynamic content requests for the clients. Figure 7 shows that as the number of proxies increases, the memory overhead on the Web Server to maintain state for the proxies increases linearly. The significant fact is that the amount of memory needed to keep state for 100 DynCoDe nodes is only 60MB, which is quite reasonable given the fact that these 100 DynCoDe nodes could potentially increase the serving capacity of the server 100 fold. This supports our assumption that server side invalidations used in our architecture are feasible.

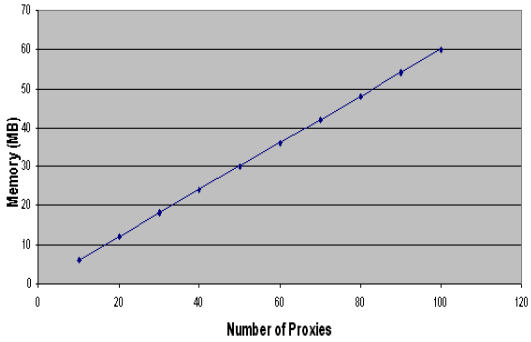

Figure 7: Memory Overhead

4. Related Work

Web caching and dynamic caching in particular has been a hot topic for research in current times. But no single best solution is yet widely accepted or standardized. In [6], a scheme for publisher centric dynamic caching is implemented. Such architectures are vital to support general-purpose caches that span multiple administrative domains wherein security is of primary importance. DynCoDe implements personalized content caching and distribution within a single content distribution network. In [7], the authors describe the architecture of Active Cache, a programmable cache, closely related to DynCoDe. However, unlike DynCoDe, it does not propose server co-ordination, has no support for invalidations and is implemented in Java, which significantly degrades its performance, compared to the DynCoDe architecture implemented in C. Also in the DynCoDe architecture both the server and the proxies co-operate and interact with each other to provide many more benefits than the Active Cache that has no interaction with the web server. The Server Side Invalidations that we have used in our architecture have also been applied to distributed file systems as in AFS. DynCoDe incorporates page composition from fragments similar to schemes such as that proposed in [9].

5. Conclusions

In this paper, we proposed and evaluated the DynCoDe architecture to efficiently and transparently push the generation and caching of personalized dynamic web content to active proxies in the edge networks, closer to the clients. The major benefits of DynCoDe include improved bandwidth consumption, user response time, server availability and scalability. The evaluation of the architecture under real network conditions has indicated that the user response time can be cut by half, bandwidth savings under the worst-case conditions are greater then 80% and the server request serving capacity could be improved by orders of magnitude. As part of future work we would like to generalize the DynCoDe architecture to other types of dynamic content like transactions and database driven content on the web. The performance results presented in this paper were based on reasonably assumed and accepted web characteristics. We would also like to use real server traces from content providers and ISPs to further evaluate the performance of DynCoDe.

References

[1] T. M. Kroeger et al., “Exploring the bounds of Web latency reduction from caching and pre-fetching”, In Proc. of USITS 1997

[2] C. E. Wills et al., “Studying the impact of more complete server information on web caching”. In Proc. of WCW 2000

[3] Vivek S. Pai et al, “Locality-aware request distribution in cluster-based network servers”, In Proc. of ASPLOS 1998

[4] P. Rodriguez et al, “Web caching architectures: hierarchical and distributed caching”, In Proc. of WCW 1999.

[5] Z. Wang, “Cachemesh: a distributed cache system for the World Wide Web”, In Proc. of WCW 1997

[6] Andy Meyers et al., “A Secure, Publisher-Centric Web Caching Infrastructure”, In Proc. of INFOCOM 2001

[7] P. Cao, J. Zhang, and K. Beach, “Active cache: Caching dynamic contents on the web”. In Proc. of Middleware 1998.

[8] B. Krishnamurthy et al, “On the use and performance of content distribution networks”, In Proc. of IMW 2001

[9] J. Challenger et al, “A publishing system for efficiently creating dynamic web content”. In Proc. of INFOCOM 2000