Applying Caching Techniques for Dynamic Personalized Content in Software Engineering Projects

David E. Athey
PaperThin, Inc.
300 Congress Street, Suite 303
Quincy, MA 02169
(617) 471-4440
dathey@paperthin.com

Jeannie R. Albrecht[†]
Duke University
Dept of Computer Science
Durham, NC 27708
(919) 660-6556
albrecht@cs.duke.edu

ABSTRACT

The hallmark of an undergraduate program is a successful co-curricular program. This program may have several forms ranging from a programming team to an active summer research program. The important ingredient in any program is student-faculty interaction in the development of solutions to current computer science problems.

This paper explores the use of research in a software engineering course at Gettysburg College. The research involves the use of Java applets and servlets on web pages within a highly personalized knowledge portal as one alternative to improve the cacheability of pages containing dynamic content, and decrease the overall load time for these pages.

Keywords: *Dynamic content, Caching, Java Applets, Java Servlets, Coldfusion*

1. INTRODUCTION

Performing research at a small liberal arts college supports the academic mission of the institution. By being engaged in research, faculty members remain current in their field and maintain classes that are crisp and exciting for students to take.

Incorporating research into the classroom is not an easy task. Students at an undergraduate institution are just initiating involvement in a discipline and are working at honing fundamental skills within their field. Information at the research level is at best accessible to seniors and advanced juniors.

For students to appreciate research, they need to have an understanding of a discipline at four distinct levels. The first is at the foundation level where basic concepts and skills are well known and mastered. The second level is comprehending the outstanding problems and questions of the discipline. Transitioning from simply learning the material to asking questions about unresolved issues is a sign of maturity in a subject area. The third level is using the solutions of others to solve similar problems in the students own projects. The fourth and final level is students developing solutions and approaches to problems on their own.

Going beyond these four levels of understanding at the undergraduate level is very rare. It is uncommon for students to independently develop solutions to open research problems. A more realistic goal is for students to implement or refine an outline developed by a professor.

There are three basic approaches to incorporating research into a co-curricular program. The first is to have students work on projects over the summer as research assistants. The second is to have a student working throughout a semester as part of an independent study. The third, which this paper addresses, is having students work in groups to incorporate research as part of a semester project within the boundaries of a course. This last approach is a hybrid between a curricular and co-curricular approach.

The project this paper discusses involves the use of dynamic content and personalized data on the Internet today. As the presence of non-static web pages continues to increase, the effectiveness of traditional proxy caches decreases. Conventional caching approaches ignore pages containing dynamic content so that each request must be retrieved from the server. To combat this trend, it is the responsibility of the content providers to adapt their pages and make them more cacheable. The semester project addressed in the remainder of this paper explores using Java applets and servlets as one possible solution to this problem.

This paper is organized as follows. The next section introduces the problem of rendering dynamic content, followed by a section that discusses dynamic content and traditional caching techniques. In section 4, we give a detailed description of the implementation of our caching scheme. Section 5 shows the results from our performance experiments. The sixth section outlines the software engineering course in which the research was integrated, including the approach taken, results, and experience of the faculty member. Section 7 summarizes our findings and draws conclusions.

[†]Supported under a National Science Foundation Graduate Research Fellowship.

2. INTRODUCTION TO CACHING RESEARCH

In recent years, there has been an increasing trend in the amount of dynamic content on the Internet. This has had a negative impact on the effectiveness of web proxy caches since most dynamic content is uncacheable. Proxy caches typically exist at the boundaries of large organizations, and they improve the performance of the World Wide Web infrastructure by caching popular documents that are shared among multiple users [1]. Since most dynamic content includes personalized, user-specific information, it cannot be cached like static HTML pages. Each request generates a cache miss and must be served by the web server, which leads to a bottleneck at request time [2].

As the use of personalized content continues to become more prominent on the WWW, content providers must find a way to improve the cacheability of dynamic pages. Recent studies have shown that many sites containing user-specific content are actually composed of a dynamic composition of static data [2]. In these cases, much of the processing required to load the pages can be completed ahead of time and cached for quicker access. Thus while the whole page cannot be served from a proxy cache, various static components can be retrieved and loaded to keep the amount of content that must be generated at the server to a minimum.

The use of Java applets is one approach to making dynamic web pages more cache friendly. When using applets, the static frame of the page can be stored and served from the proxy cache, while the dynamic content can be handled by invoking the applet. The applet can then process the request and retrieve the necessary personalized information [3]. This method takes advantage of the way in which browsers render applets. The browser will reserve a space on the page for the applet and then continue loading the rest of the content. This has the added benefit of allowing the remainder of the page to load instantly instead of waiting for the entire page to be retrieved from the server.

In addition to using applets to serve dynamic content, further improvements in load time can be achieved by caching within the applets. In many cases, the applets retrieve material from servlets or images that are shared among all users. Thus there is no reason for the servlets to process each request individually. The responses that the servlets retrieve from database queries, for example, can also be cached to eliminate the overhead of database communication for later requests. In our study, we investigated the effectiveness of these techniques for an Internet based college portal system containing highly personalized content.

3.CACHING DYNAMIC CONTENT

3.1 What is Dynamic Content?

Dynamic content is a term typically used among web users to describe any user-specific data that is generated at the time of the request. This includes database queries, search engine responses, time-specific information such as stock quotes, and any data that is generated by a server that returns a different object for each access. It allows content providers to create more personalized web pages that are tailored to individual user preferences.

Since web pages containing dynamic objects change with each access, proxy caches attempt to identify and not cache instances of the personalized content [3]. Thus it is important to design these pages in a cache-friendly way and to extend the benefits gained from caching as much as possible. Fortunately, most pages that contain dynamic content also include some static content that is responsible for formatting specifications, headers and footers, and other unchanging information. This property can be leveraged to improve the proxy cache hit rate.

3.2 Traditional Caching Techniques

Caching is motivated by the simple idea that if some information is retrieved, and there is a possibility that this information will be used again in the future, then a copy of the data should be stored in an easily accessible location for quick future access. While specific caching techniques vary greatly, the overall goal in all caching strategies is to move the web page or data object “closer” to the user in terms of network distance. Typically objects stored on computers with the same Local Area Network (LAN) can be retrieved much quicker than those that are retrieved from remote computers on other networks.

Although most web browsers provide some amount of caching on disk drives and in memory, special cache servers (or proxies) are often used to provide a shared cache to a number of different users in a certain location connected to the same LAN. If a page can be retrieved from a proxy cache residing on the same LAN rather than having to travel across the wide area to a remote host, the time until the request is satisfied is significantly reduced. In the case of proxy caches, the proxy server attempts to identify and avoid storing pages containing dynamic content. Thus the benefits of caching are not gained.

Another traditional caching technique is to prefetch web pages before users request them, so that they are stored in the local or proxy cache for future use. In this case, the proxy server must speculate on

which page the user will request next based on the observed past actions. Successful prefetching will lower the latency seen by users when requesting objects, increasing the performance and hit rate of the proxy cache.

One important feature of these techniques is that they both focus on delivering a complete web page to the requesting user in a reduced time. If dynamic content exists, these methods are ineffective and pointless. Smarter strategies are needed to achieve the same performance gain as static page caching.

4. OUR IMPLEMENTATION

In an effort to take advantage of the fact that most web pages contain a combination of dynamic and static content, our caching technique separates the pages into cacheable and uncacheable sections. Unlike traditional caches, our method does not try to deliver the web page to the user as one contiguous object, but rather in pieces that are retrieved separately.

We assume the most important goal is to get at least part of the data back to the user as quickly as possible. Since a user cannot view an entire web page at once, by retrieving the static sections quickly from cache and using Java applets to fill in the smaller dynamic sections, the user is able to begin viewing the static information while the dynamic data is still being loaded. This minimizes both the amount of personalized data that must be sent over the network for a given request, as well as the time the user must wait to begin viewing the page.

4.1 CNAV: Campus Knowledge Portal

In January of 2000, a group of IT professionals from Gettysburg College along with several recent graduates began the development of a Campus Knowledge Gateway named CNAV. CNAV interfaces with campus data warehouses, filtering the data from each source as it transports the data into its own repository. Once the data is stored within CNAV, CNAV then associates information by using ID, interest, and keyword relationships.

Once CNAV has established relationships among its information, it then secures its knowledge through two primary mechanisms: group and user defined privacy settings. Users can then access the data and request personalized information relating to courses, finances, campus events, and contact information from any computer connected to the Internet.

CNAV also provides administrative modules such as employee timesheets, computer helpdesk, inventory, advising tools, and an online portfolio system. At Gettysburg College and other institutions, CNAV has become the virtual town hall where members come to learn about campus events, voice opinions, and interact with other members of the community.

4.2 Caching Techniques in CNAV

Our caching strategy was developed within the scope of the CNAV campus portal. Since web pages in CNAV contain high amounts of dynamic content, traditional caching methods did not achieve a high performance gain. We focused on the two most frequently accessed pages in the system that correspondingly have the most dynamic and personalized content. Both of these pages contain approximately ten sections of user-specific data, as seen in Figure 1. The areas of the page where dynamic content is retrieved are referred to as *channels*.

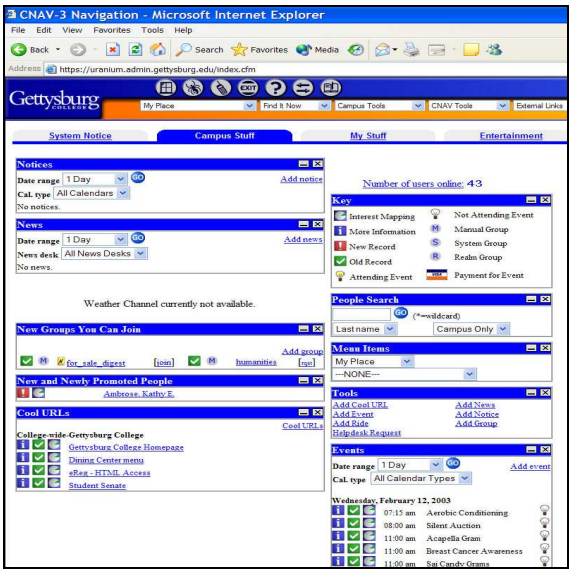

Figure 1: This is a screen shot of CNAV that shows the multiple channels containing dynamic content.

As stated above, our basic approach to caching in CNAV is to deliver a web page in pieces rather than wait for an entire page to become available. This works especially well for our target pages since they are a combination of personalized channels and images, in addition to the templates containing static content.

Our strategy has three components. First, we create a general template page that is appropriate for each realm of user. Users in CNAV are classified into a realm such as faculty, student, alumni, and parent. This template page can be replicated on several servers using traditional caching techniques. Second, the template page has content

holes that are first filled with an empty container to allow the browser to render and size the page. Third, servlets exist to process information from the database to produce dynamic personalized content. The container requests its content from a servlet that uses a combination of cached data, database queries, and algorithmic processing to produce the necessary content for the container.

For our purposes, the first type of container is an HTML image tag with predefined width and height. For the image tags, the servlet returns gif or jpg data for the image. The URL that is sent to the servlet in the SRC attribute of the IMG tag contains the ID of the user who is currently viewing the page and an ID of the entity to which the image is associated. The servlet uses both IDs to personalize the image content it sends back.

An example of the personalized image content can be found in the Event Channel of CNAV. There are three images that inform the user about each event. The most significant of these images is the interest matching image that indicates to a user how interesting the event is based on their preferences. This image is a pie chart that is calculated for each person/event pair. The servlet that does this calculation loads an interest vector for every user and every event. It then does a bitwise AND operation to calculate the intersecting interests for the events and users. Based upon the number of intersecting interests, the servlet returns an appropriate pie chart image indicating the number of overlapping interests.

The second type of container is a Java Applet with a swing based HTML panel. In this case, the servlet returns HTML that the applet renders. The applet contains all the information for one single channel. This allows for a fixed sized channel which simplifies page layout. The user can then use scrollbars to view the contents of the applet and individual channels. This is a side benefit of using applets for the entire channel content. Similar to images, when using applets the user's ID and any personalized data is embodied in the URL that is sent to the servlet.

To better describe how this system works, consider the event channel. In the event channel, the user can specify preferences on the calendar indicating which categories of events are drawn from the data warehouse to populate their screen. For example, users can choose to see all campus events, or they can choose just the sporting events. Java applets are used in the channels to retrieve dynamic content. The applets communicate with Java servlets and images to obtain the user-specified data from the server. In the URL that is sent to the servlet, the user's ID is sent along with the calendar categories that the user requested. The servlet then pulls the events either from the data store or from its memory, which also acts as a cache, and then sends back the formatted HTML to the applet showing the specific list of events the user is requesting.

The following is a time line depicting the process by which personalized data is retrieved. Figure 2 illustrates the procedure.

Time 0: The server is started and a template page is built for each realm in the portal. This template is updated periodically throughout the day. The template is built in Coldfusion Markup Language. This template can be stored in the first-level cache since it is common to all users. It is accessible by several front-end Coldfusion proxy servers that serve the page to requesting users.

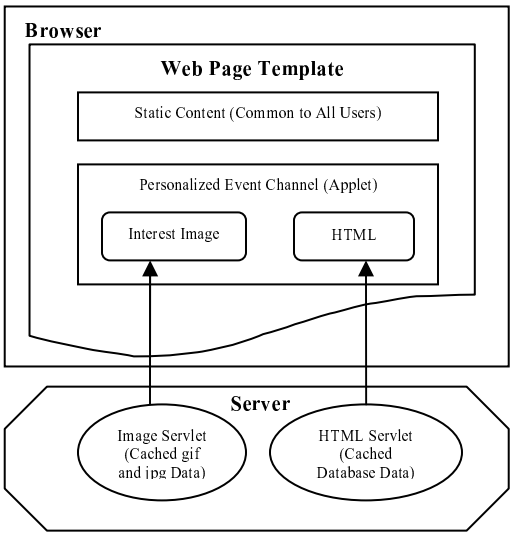

Figure 2: This diagram shows how channels interact with servlets to retrieve user-specific data.

Time 1: Each servlet is initialized on the server with intermediate data from the database. Preloading intermediate data acts as a local cache for the servlets since it attempts to prefetch data and avoid expensive database queries. The cached data is stored in global memory so that each servlet thread can access it. The servlet updates its cache periodically throughout the day.

Time i: A user requests the page from a standard web browser.

Time i+1: The Coldfusion application server renders the template page by inserting the user's portal ID into the appropriate locations in the template.

Time i+2: The static HTML content is loaded into the browser.

Time i+3: The user begins to read the static content while the Event Channel in the template uses an applet to request the image source from the image servlet. Since applets are given a predefined space on a web page by browsers, the space on the page for the image is reserved although the image has not been retrieved. The other Event Channel applet starts and requests its content from the HTML servlet that stores database query results in formatted HTML.

Time i+4: The other channels in the template operate in a similar fashion. Each servlet on the server processes its data along with the additional information in the requesting URL. The servlets then retrieve either the requested image or HTML and send the results back to the applets in the browser.

Time i+5: The requested images are displayed by the browser and the Event Applet displays the HTML received from the servlet. The user can now view the personalized content.

4.3 Implementation Status

Currently, the production version of CNAV employs two caching techniques. The main template page for each user realm is generated every hour and updated in the Coldfusion servers. Additionally, image content on the opening portal page is cached using the prefetching technique previously described. For each user-specific image, the source for the image is a servlet that selects the appropriate image to send to the browser. These techniques have been in full production for two years. At times, there can be up to 100 images produced this way on an opening page, depending upon the number of News and Events objects posted to the portal.

The implementation of different channels in the applet container is still in the testing phase of the development process. Using applets improves performance by decreasing load time; however some challenges arise with formatting. Currently, the testing is focused on the web page layout and style. Applets are scheduled to be moved to the full production machine during the summer of 2003. There are also some minor Java version compatibility issues within browsers that need to be resolved with legacy applets. Some methods available to applets in older Java versions have been deprecated, and these applets need to be rewritten before the caching applet container can be rolled out to production.

5. PERFORMANCE

The performance of the image and applet container was evaluated separately. For a page with ten channels and several hundred personalized images, the browsers and all servers run the risk of timing out when trying to generate the channel page and load each image in series. If they do not time out completely, load time is unreasonable. On the other hand, pages load quickly when using servlets to generate the customized image data. During our testing, servlets enhanced pages involving ten personalized channels showed text and image information within a fraction of the time seen from alternative display methods.

We centered our testing on the Event Channel for evaluating the applet caching technique since this channel contains the most dynamic information. We artificially loaded 2000 events to be displayed at once. Even though there will not be 2000 events happening on Gettysburg College's campus during a one week period, we wanted to evaluate this technique under heavy load.

We tested four different configurations of caching. The first was no caching at all where the 2000 events were rendered by the server individually for each request. The second configuration used a template page with the applet container embedded in it. The applet requested its content from the same HTML-based page used in configuration one. The third design had the same template as the second configuration, but now the applet loaded the content generated from a servlet that pulled its data from the database for each request. The servlet generated HTML from the database query result to send back to the applet. The fourth configuration again used the template page with the applet; however, this time the content was retrieved from a servlet that had previously cached all the data necessary to generate the HTML immediately. In fact, the servlet cached the actual HTML. Figure 3 shows our results.

We recorded four times for each configuration. The first time was the number of seconds required to see any data load in the browser. The second time was the number of seconds to see event data. The time to see half of the 2000 total events was the third measurement, and the final time was the complete loading of all 2000 events.

The results both validate our approach and reveal an interesting tradeoff. The overall goal of the caching techniques is to get something as quickly as possible to the user. Our approach has accomplished this as seen in the first three timing lines on the graph. The time decreased in each experiment with the lowest time occurring with the Applet–Servlet Cache configuration. The tradeoff is that the total

load time for this configuration increased. We have determined that this is because the applet requires more time to format the data and update its scroll bars. However since the scroll bars on the applet allow for fixed sized channels with variable length content, we found this tradeoff acceptable.

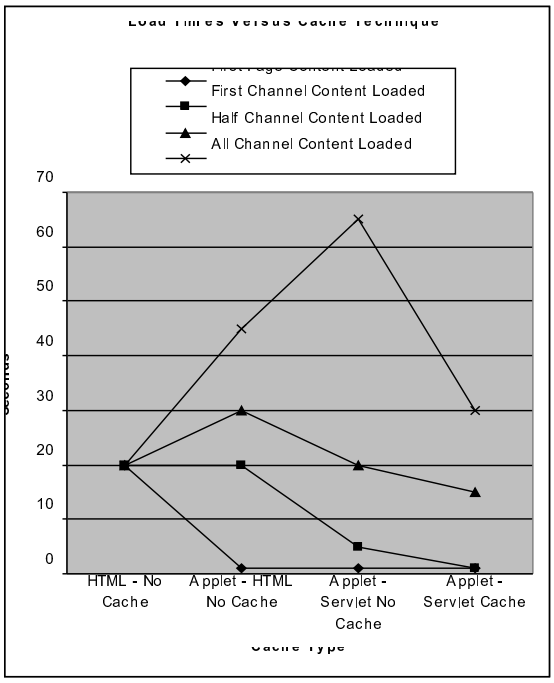

Figure 3: Results depicting load times for different cache configurations.

6. RESEARCH IN THE CLASSROOM

6.1 Software Engineering: Senior Capstone

At Gettysburg College, the software engineering course is taught as the senior capstone course that is taken by all second semester computer science majors. In this course students form groups and develop an application for clients outside the college.

This course has traditionally served another purpose at Gettysburg by providing a venue for experimenting with new technologies. For example, in the Spring of 1996 several groups started developing applications in Java with the first beta compiler. Several new approaches incubated in this course have transitioned into the main stream curriculum.

Each group in the course works their project through the entire life cycle. Each group gets an overview of their client's general needs. The group then interviews the client to get a full understanding of the problem. After this interview, the group generates a full requirements document covering every aspect of the project from functional specifications to the cost of the system.

Once the requirements document has been finalized, the students implement the application. Many times, the students find themselves modifying the requirements or seeking more information from the client. As part of the last component of the life cycle, the groups present their application to the client.

6.2 Integration Approach

The computer science department at Gettysburg College has a tradition of working with students on research projects over the summer. This work finds its way into the curriculum through lectures and presentations given by the research students.

To experiment with integrating research deeper into the classroom, a group from the senior capstone course was selected to incorporate the caching techniques detailed in this paper into their web-based application. The application is an inventory control system for the athletic department at the college.

There are two twists to the project. First, the college purchases its major supplies through a bidding process where the required inventory list is given to vendors in order for them to submit such a bid. The second is the desire to control inventory while out on the playing and practice fields. Here, a wireless PDA device is used to scan bar codes of training supplies on the field. If the PDA has a network connection, then it will adjust the database. If no network connection is available, the PDA will cache the database transaction and commit it at another time.

The inventory application is well suited for this type of caching research. The best area is in the bidding processing. It is helpful for the client to have multiple channels on one page to simultaneously review the details of the bids. There is one channel that is an aggregate of all the minimum bids. However, some vendors send in non-standard bid items and thus the client must review each item in each bid.

The applet text-based cache is the solution used in the project. Since the project has no images associated with it, there was no need for the image caching. Also, the amount of data that was processed did not require the use of servlets. Each applet referenced the ColdFusion server directly without any other processing.

6.3 Results and Experiences

The applet caching solution worked well for this application. The student group presented the application using the applet and the research. Unfortunately, the students found that the application could not use all the components of the research. This is one drawback to incorporating research into class projects instead of presenting a straightforward lecture of the research itself. To counteract this situation, the professor supplemented the student group presentation with a broader, more traditional mini-lecture on the research contained in this paper.

7. SUMMARY

Disseminating research to undergraduate students involves more preparation than disseminating it to graduate students or other researchers. From the experiences in this paper, a multi-pronged approach is best. A traditional lecture is adequate, but the students have no direct experience with the research. Incorporating the research into a group project is also adequate, but at times does not highlight the strengths of all the aspects of the research. The final approach involving the student group presenting their application with certain aspects of the research, coupled with the professor's mini-lecture on the remaining aspects, proved to be the best solution.

The research embodied in the paper shows that as the popularity of personalized web pages on the Internet continues to grow, the use of traditional caching techniques is becoming less effective. Pages containing user-specific content are typically considered uncacheable and are ignored by conventional proxy caches. In most cases, these pages contain a combination of both static and dynamic data. While the dynamic data must be retrieved from the server for each request, the static sections of the page are common to all users and can be cached. Through the use of Java applets and servlets within an enterprise knowledge portal, we were able to exploit this feature to decrease the load time of personalized pages and reduce the amount of data that must be retrieved from the server for each request.

While our results indicate that a significant reduction in load time is achieved using applets and servlets, we also found that there was some overhead involved with using Java to format the data retrieved from the servers. In the future we plan to experiment with improving the applet load time, as well as address some of the formatting and compatibility problems that arise when using Java applets in different browsers.

8. ACKNOWLEDGEMENTS

The authors wish to acknowledge Professor Rod Tosten for allowing us to use his software engineering course as a forum to disseminate this research.

9. REFERENCES

[1] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin and H. Levy. Organization-Based Analysis of Web-Object Sharing and Caching. In *USENIX Symposium on Internetworking Technologies and Systems*, 1999.

[2] K. Rajamani and A. Cox. A Simple and Effective Caching Scheme for Dynamic Content. *Rice University Computer Science Technical Report*, November 2000.

[3] M. Rabinovich and O. Spatscheck. *Web Caching and Replication*. Addison-Wesley, 2002.

[4] Pei Cao, Jin Zhang, and Kevin Beach. Active Cache: Caching Dynamic Contents on the Web. In *Proceedings of ACM/IFIP/USENIX International Middleware Conference*, 1998.