World Wide Web (2006) 9: 253–275
DOI 10.1007/s11280-006-8556-0

Extending Proxy Caching Capability: Issues and Performance

Wei Hao · Jicheng Fu · Jiang He · I-Ling Yen · Farokh Bastani · Ing-Ray Chen

Received: 31 March 2004 / Revised: 23 April 2005 /
Accepted: 8 November 2005 / Published online: 8 June 2006

Abstract Proxy caching is an effective approach to reduce the response latency to client requests, web server load, and network traffic. Recently there has been a major shift in the usage of the Web. Emerging web applications require increasing amount of server-side processing. Current proxy protocols do not support caching and execution of web processing units. In this paper, we present a weblet environment, in which, processing units on web servers are implemented as weblets. These weblets can migrate from web servers to proxy servers to perform required computation and provide faster responses. Weblet engine is developed to provide the execution environment on proxy servers as well as web servers to facilitate uniform weblet execution. We have conducted thorough experimental studies to investigate the performance of the weblet approach. We modify the industrial standard e-commerce benchmark TPC-W to fit the weblet model and use its workload model for performance comparisons. The experimental results show that the weblet environment significantly improves system performance in terms of client response latency, web server throughput, and workload. Our prototype weblet

W. Hao (✉) · J. Fu · J. He · I.-L. Yen · F. Bastani
Department of Computer Science, University of Texas at Dallas, Dallas, TX, USA
e-mail: weihao@utdallas.edu

J. Fu
e-mail: jxf024000@utdallas.edu

J. He
e-mail: jianghe@utdallas.edu

I.-L. Yen
e-mail: ilyen@utdallas.edu

F. Bastani
e-mail: bastani@utdallas.edu

I.-R. Chen
Department of Computer Science, Virginia Tech, Northern Virginia Graduate Center, Alexandria, VA, USA
e-mail: irchen@cs.vt.edu

system also demonstrates the feasibility of integrating weblet environment with current web/proxy infrastructure.

Keywords e-commerce · proxy caching · service migration · web system

1. Introduction

Proxy caching has been extensively used to reduce the network traffic generated by HTTP and to shorten the response latency. There have been extensive research works in maximizing the benefits of web caching, including improving cache management policies [5, 15] and using proxy arrays to provide collaborative caching [4, 10, 24]. Improving proxy-caching techniques can potentially increase the cache hit rate. However, due to the increasing appearance of non-cacheable web objects, cache hit rate is limited no matter how good the caching scheme is. One major category of non-cacheable objects are those generated dynamically by web processing units, such as CGI scripts, Java Servlets, Active Server Pages, etc. Some other data objects are made non-cacheable due to security requirements or statistics collection purposes. In [11], it shows that accesses to web processing units constitute 15% of the web requests. Recently there has been a major shift in the usage of the Web. Most of the e-commerce applications now require more web server processing. A study on a large Web-based shopping system shows that 95% of the client requests were for dynamic objects [3]. These client requests are forced to be processed at web servers and can result in heavy server load, increased network traffic, the lengthened response latency.

The server and network overloading problem can be eased by using replicated servers. However, the locally distributed server architecture can reduce the load on web servers, but not communication latency or network traffic. With a high access rate, the link between the server cluster and the Internet can still become a bottleneck.

Edge servers can be used to reduce the response latency and network traffic to a certain degree. Recent study on web server offloading has a similar concept [13]. WebSphere Edge Server [23] is another edge server offloading example. These approaches deploy part of the application logic on edge servers. However, the improvement by edge servers can be limited. First, most of the edge server approaches consider static offloading. For example, in WebSphere, servlets are pre-allocated to edge servers in advance. Also, edge server offloading is a heavyweight mechanism. Thus, these approaches cannot easily adapt to changing client access patterns. Also, the level of replication in edge server approaches is generally limited. Consider K independent web sites. In edge server approach, N widely distributed web servers can be used for each web site to handle the workload. But dedicating one platform to one web site may result in the waste of resources. If KN independent servers are used to share the workload of all the K web sites, then the same resources can be placed even closer to clients and provide better load sharing, thus, yield much better performance improvement. The proxy servers, that are playing crucial roles for caching static web pages at present, can provide the natural infrastructure for implementing the independent servers approach.

However, current web and proxy protocols do not support caching and execution of web processing units. We need to extend proxy capability to support caching of a wider variety of objects. Some dynamic object caching techniques can be used to improve Web access performance. In [6], an approach for active cache has been presented. Web server sends cache applets with web pages. Proxy server, when delivering cached web pages, invokes the corresponding cached applet to perform necessary processing. If the cache applet needs to access data objects or other resources on the web server during its execution, it has to access remotely regardless whether the remote accesses are cost-effective. Also it separates the concepts of web pages from processing code, so it adds difficulties to web site administration. ESI (Edge Side Include) [9] is an industrial standard proposed by Akamai with other companies for fragment caching at edge servers. A simple markup language is used to define the dynamic assembly of web contents at the edge of the Internet. Some other research works [8, 17] have been proposed based on the fragment cache concept but define different frameworks. Fragment cache can improve web system performance and reduce network traffic due to the finer caching granularity and dynamic page assembly. But fragment caches do not consider web processing components, so traditional non-cacheable web objects are still non-cacheable on proxy servers. In [18], an approach to cache dynamically generated HTML pages has been proposed. Essentially, dynamic content is cached and delivered to the clients that request for exactly the same dynamic content as in a previous request. The advantage of this approach is its simplicity and easiness to use on web servers or proxy servers. If the frequency of clients requesting for the same dynamic contents is high, then this approach is effective. But in some web applications, the clients rarely request the same dynamic contents. In these cases, this approach is not very helpful.

In [14], we introduce the concept of migratable processing unit, weblet. In the weblet environment, processing units on web servers are implemented as weblets. These weblets can migrate from web servers to proxy servers to perform required computation. Web servers and proxy servers are extended to support the execution of weblets. With the weblet approach, many web objects that were non-cacheable in the conventional infrastructure will become cacheable. It builds on top of current proxy infrastructure and can significantly increase the hit rate of proxy servers.

To validate the weblet approach, we conduct experimental study to analyze weblet environment performance. We have implemented the weblet environment and examined its performance impact on workloads generated using TPC-W benchmark [22], which is an industrial standard benchmark for e-commerce applications. The experimental study shows our proposed weblet environment can achieve significant performance improvement in terms of response time, throughput, and load. We have also implemented the prototype weblet system to demonstrate the feasibility of integrating weblet environment with current web/proxy server.

The rest of the paper is organized as follows. The design and implementation of weblet environment is discussed in detail in Section 2. Our extensions to HTTP protocol, Apache Web Server, and Squid proxy server are discussed. Weblet Migration Manager is discussed in Section 3. Section 4 describes the TPC-W benchmark and its implementation in the experiments. Section 5 discusses the experimental study. We present experimental setup, methodology and result analysis. We discuss further benefits of weblet environment in Section 6. Finally, Section 7 summarizes the paper and outlines future research directions.

2. Weblet environment overview

We consider web server as a collection of web services, some deliver web pages to clients and some provide special services other than web page accesses. Each program unit implementing a web service is considered as a weblet. (Note that the web services we discuss here is a general term, not the specific web services discussed in [19], though weblet can definitely be used to improve the performance of web service systems as well.) Existing proxy servers provide web page delivery capability so that web pages cached locally at a proxy can be retrieved and sent to the client in the same way as that at web servers. This can be viewed as one type of weblets being implemented at both the web and the proxy servers. Since it is not possible to extend the proxy servers to include all different types of web services, we need to equip the proxy servers to provide a uniform execution environment so that various weblets can be migrated and executed at proxy servers. We develop *Weblet Engine* to provide such execution environment and use it on proxy servers as well as web servers to facilitate uniform weblet execution. Basically, Weblet Engine provides a uniform execution platform for weblets, which is very similar to an agent platform that supports uniform execution of agents [1, 16].

In many cases, weblets need to carry data objects with them in order to perform their functions. Consider an online shopping system, a weblet that displays the merchandise catalog to customer needs to carry merchandise information from the backend database. Thus, additional protocols are required to manage the data migration and coordinate the migration effort with the weblet migration. Also, analysis is required to determine whether migration of a data object is cost effective. If a weblet, once migrated to a proxy server, needs to access many data objects on the original web server, then it may be more cost effective not to migrate it. Based on the locations of the data objects a weblet requires, we need to decide whether to migrate the weblet. We design *Weblet Migration Manager* on the web server to analyze the cost and make all migration decisions. The detailed decision making process will be discussed later.

Security is another critical issue in the weblet environment. A malicious weblet may try to compromise the proxy server or try to perform a DoS attack by consuming all the resources of the proxy server. Thus, security mechanism should be incorporated in the Weblet Engine to protect the resources on proxy servers. On the other hand, a malicious proxy may compromise the privacy and integrity of the weblets. Also, one weblet may exploit the security loopholes of the proxy server and compromise the privacy and integrity of other weblets. As mentioned above, each weblet may carry some data objects with it. These data may need to be shared with some weblets but protected against the accesses by other weblets. Thus, sophisticated authentication and access control protocols are needed for fine-grained security protection for proxy and weblets' resources. At the bottom layer of the Weblet Engine, we develop security control components to realize a fine-grain security control mechanism. The security issues are not discussed in this paper. Details can be found in [14].

On the implementation side, current Web infrastructure does not support proxy caching of processing units (weblets). We need to modify web servers and proxy servers to support the migration, caching, and execution of weblets and their data. Also, HTTP protocol needs to be extended to support weblet requests. The modifications to various components in the Web infrastructure to support the weblet environment are discussed in the following subsections.

2.1. Protocol extension for weblet support

We expand HTTP protocol to support the processing of weblet objects. A new suffix ".wlet" is used to differentiate weblets from the conventional requests. In HTTP 1.1 this new suffix is simply treated as a new entity header. When the proxy receives an HTTP request, it checks whether the requested object is a weblet. If the object has the ".wlet" suffix, then it is processed according to the weblet protocol.

We illustrate the weblet processing in Figure 1. Client A sends a request to its proxy accessing http://www.abc.com/a.html. Since it is a conventional web request, it is handled in the conventional way. Client B sends a request to its proxy accessing http://www.abc.com/a.wlet, which is a weblet request. Upon receiving the request, the proxy server passes it to the weblet engine. The weblet engine checks if the requested weblet is cached locally. If not, then the request is forwarded to the web server. The web server, upon receiving the weblet request, decides whether to migrate the requested weblet. If not, then its weblet engine executes the weblet and returns the results to the proxy. If the decision is in favor of migration, then the requested weblet and associated data are sent to the proxy. Once the weblet is on the proxy (either it was already cached or just newly migrated), the proxy weblet engine executes the weblet and forwards the results to the client (Client B in this case).

Weblet Engine consists of three components: Uniform Execution Platform, Global Security Manager, and Local Security Manager. Uniform Execution Platform provides the execution environment for weblets. Global security manager interacts with Certification Authority (CA) servers and performs weblet authentication and authorization. Local security manager provides fine-grained access control to weblets' and local proxy resources and it provides privilege check before perform resource access actions requested by the weblets. Certification Authority server is used to authenticate various entities in the system. The certificate is used for both authentication and privilege control. Location Manager is used to keep track of weblet's location.

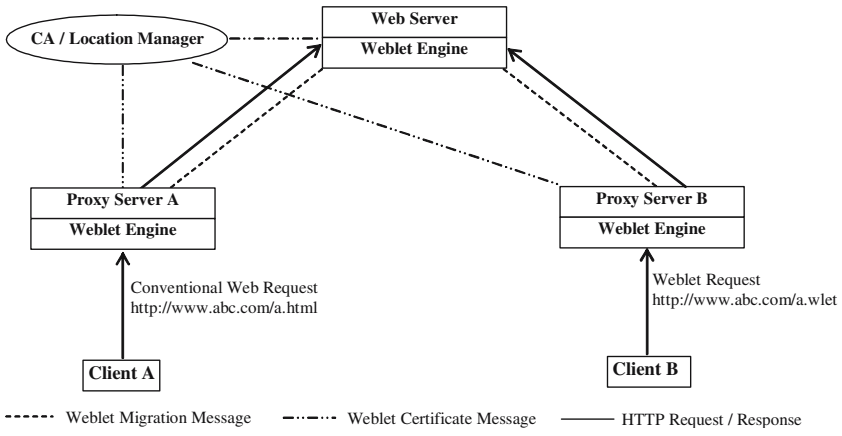


Figure 1 Weblet request processing.

2.2. Web server design and implementation

We designed the Web-Server Extension Module (shown in Figure 2) to enable web servers to support the weblet environment. The extension system consists of a Mod_Weblet module that lies in Apache web server, a Weblet Engine, and a Weblet Migration Manager.

Mod_Weblet Module is used to intercept the weblet request and forward it to the weblet engine. Weblet Migration Manager is used to decide whether a weblet and its data objects should be migrated. All the modules, except for the Mod_Weblet module, can be implemented externally to existing web server program; however, Mod_Weblet should be integrated in the web server. We use Apache as our base web server due to its popularity [2] and open source. The Mod_Weblet module is added to Apache. It consists of the weblet request handler and the weblet engine interface. Weblet request handler is responsible for receiving the weblet request from the Apache core module, generating the HTML response message, and sending it back to the Apache core module. Weblet engine interface component is used to forward the weblet request to the associated weblet engine and wait for the reply message. The other part of the Apache program stay intact.

2.3. Proxy server design and implementation

Similar to the case on web servers, we also developed a Proxy-Server Extension Module to enable the proxy servers to support the weblet environment. The design of Proxy-Server Extension Module is illustrated in Figure 3. The extension module consists of a Weblet Request Interceptor that lies in Squid proxy server, a Weblet Engine, and a Weblet Cache Manager.

We choose the Squid Proxy Server [20] as our proxy server since it is open source. In order to make Squid proxy support weblet, we add a new module called Weblet Request Interceptor. The architecture of Weblet Request Interceptor is the same as Mod_Weblet. It consists of two components, the weblet request handler and the weblet engine interface component. Weblet request handler is responsible for

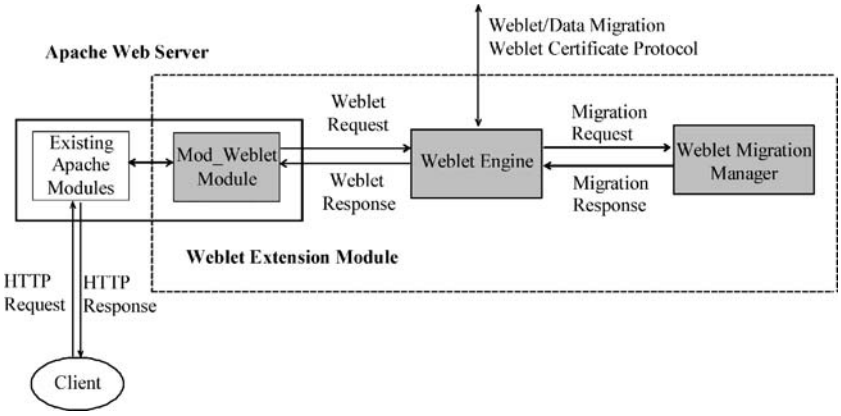


Figure 2 Weblet extension module for web server.

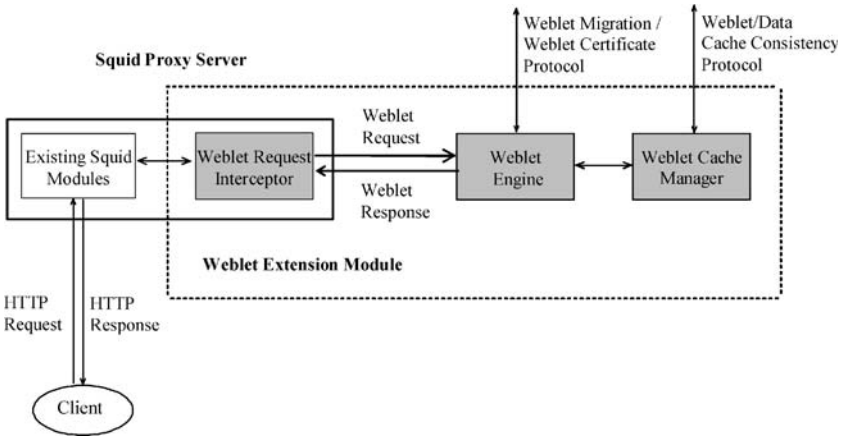


Figure 3 Weblet extension module for proxy server.

intercepting weblet requests from the processing flow of existing Squid modules. Weblet engine interface component is used to forward the weblet request to the associated weblet engine and wait for the reply message.

Weblet Cache Manager provides weblet caching mechanism. We use the Greedy-Dual-Size Popularity (GDSP) algorithm [15] as our weblet caching replacement policy. For weblet caching consistency, we use a TTL-based cache validation approach. Every cached weblet is assigned a TTL value. When a weblet with expired TTL is accessed, the Weblet Cache Manager checks with the original web server to validate the cached weblet. If there are no changes on the cached weblet, its TTL is renewed. Otherwise, the latest copy of weblet migrates from the original server to the proxy server. This mechanism is the same as TTL-based cache validation protocol for HTML documents [12].

3. Weblet migration manager

3.1. Data migration

Migration decision-making is a critical issue in the weblet environment. Migrating a weblet from a web server to a proxy server that is closer to the clients does not always yield performance improvement. Weblets, during execution, may need to access data from web servers. In an online shopping website, the browsing weblet needs to access the catalog information. The required data objects can be migrated to the proxy server with the browsing weblet. However, when the customer actually places an order, the ordering weblet needs to determine the availability of the desired merchandise. The quantity of a merchandise is a frequently updated data and may not be suitable to be migrated to the proxy site. The need for accessing data on the original web site results in communication overhead that could make weblet migration undesirable. We classify data objects into three types to assist the

migration decision making. The three types of data objects and corresponding access protocols are listed in the following.

Type 1: Many web applications involve data objects that are only read by the client and updated infrequently (or never updated) by the web server. For example, a data object storing the merchandise catalog in an online shopping system. This type of data objects may be migrated. Each migrated data object is assigned a TTL value by the web server. When a migrated data object is accessed by a weblet, the TTL of the data object will be checked. If the TTL expires, then the proxy server must validate the data object with the original web server.

Type 2: Some data objects are read and updated by clients, but are rarely shared. For example, a data object storing the personalization information of a client or a data object storing individual shopping cart information are of this nature. This type of data objects may be migrated too. Once it is migrated with a weblet, it can be cached at the proxy server with the weblet. We use a locking mechanism to ensure the consistency of the data object. Let w denote the weblet and d denote the data object that w needs to access. Also, assume that a copy of d, namely, d', is migrated with w. Whenever weblet w is activated at the proxy server and w knows that it needs to access d' during its computation, then w sends a request to the original web server to validate d' as well as to lock d. Locking d ensures that no concurrent updates to d would occur. Since d is rarely shared, the lock barely impacts the system performance. At the end of the execution, w sends all its updates on d' back to the web server to update and unlock d. When some other proxies or clients attempt to access d while it is locked, we let the web server reclaim its control over d. The proxy server sends the updates on d' to the web server and invalidates d'. All the remaining accesses to d by w will be directed to the web server.

Type 3: The data objects that are highly shared and need frequent updates are not suitable for migration.

Another factor to be considered for migration decisions is the structure of data. For example, database contains not only a collection of data objects, but also their semantic relations. Currently, we consider a table as a migration unit of databases. For migration decision-making, we consider data types, table size, and table access frequency. We can estimate the table access frequency from weblet logic and data access log files. Tables with small sizes and high access frequency are more suitable for migration. We define migration values for tables to indicate the suitabilities for migration. It is defined as the table access frequency divided by the table size. A table is migrated if its migration values is greater than or equal to a threshold.

We design a database protocol to migrate tables while capturing both the data objects in the table and the semantic relations among them. Figure 4 illustrates the database table migration flow. First, the migration value (MV) for a table is calculated. Then, the table is captured by an XML message. There are four elements in the message, pre-population, population, post-population, and dataObject elements. The pre-population, population, and dataObject elements are mandatory,

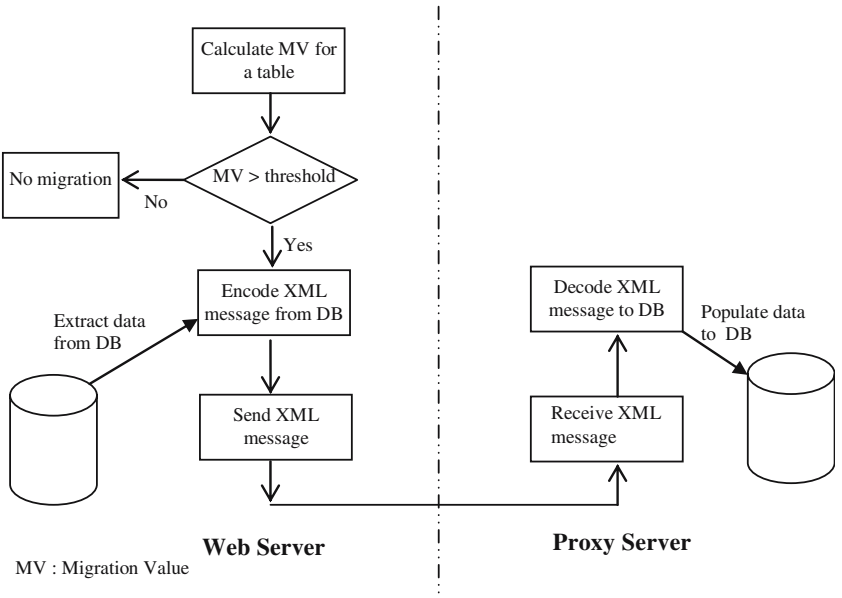


Figure 4 Table migration flow diagram.

and the post-population element is optional. When a web server decides to migrate a table, it first generates the SQL DDL (Data Definition Language) statements for creating the table structure and stores them in the pre-population element. Then, it generates the SQL DML (Data Manipulation Language) statements for populating the table with migrated data and stores them in the population element. Next, it generates the SQL DDL statements for adding index/constraints on the table, if necessary, and stores them in the post-population element. Finally, the web server exports the data from the table and stores the exported data in the dataObject element. If the exported data is binary, then it is encoded in base64 format. Next, the XML message representing the table is sent from the web server to the proxy server. When the proxy server receives a data migration message, it first creates the table based on the pre-population information. Then, the proxy server populates the table by using the population and dataObject elements. Finally, the proxy server adds index/constraints on the table if necessary by using the post-population element.

Using a table as a migration unit is somewhat coarse in granularity for large tables. We will consider using a partial table, e.g., a group of records or a group of fields, as a migration unit to improve the resource utilization. But this modification is beyond the scope of this paper.

3.2. Weblet migration

The weblet migration manager is responsible for weblet migration decision-making based on cost analysis. Some factors that should be considered for migration decision-making include the communication cost between the server and client

nodes, the dependency between a weblet and its required data objects, and the cost for maintaining weblet/data consistency, etc. Based on the data/server location and historical data access patterns, the cost incurred or benefits gained for a weblet to execute at a proxy server can be estimated. The general rules for weblet migration decision-making are as follows:

If a weblet does not need to access any data object, then it can be migrated easily.

If the data objects required by a weblet belong to data type 1 or data type 2, then we may migrate the weblet with its data objects.

If the data objects required by a weblet belong to data type 3, but the data objects are accessed infrequently, then we may migrate the weblet and leave the data objects on the web server.

If the data objects required by a weblet belong to data type 3, and the data objects are accessed frequently, then the weblet won't migrate.

If a weblet need perform operations on several data objects at the same time, such as executing a query of joining multiple tables. If some data objects can be migrated to a proxy server but the other data objects are not suitable for migration, then the weblet won't migrate.

4. TPC-W in weblet experimental study

4.1. TPC-W

TPC-W benchmark from the Transaction Processing Council is an industry-standard transactional web benchmark that models an online bookstore. The components of TPC-W can be logically divided into three parts: emulated browsers (EB), web server, and database. The EBs simulate the activities of multiple concurrent web browsing users, each making independently requests to a web server for web pages. TPC-W defines 14 web interactions (pages), which support user browsing, searching, customer registration, ordering, ordering inquiry, and administrative activities. Most of the 14 web interactions need dynamic content generation because they need to access information stored in the database. TPC-W specifies that the database consists of a minimum of eight tables: ITEM, CUSTOMER, ADDRESS, COUNTRY, ORDER, ORDER_LINE, CC_XACTS (credit card transaction data), and AUTHOR tables. TPC-W workloads are generated by EBs and the workload intensity can be controlled by number of EBs. An EB always starts with a browsing session at the bookstore's home page and then continues traversing the bookstore's web pages, following different links and entering user information with different probabilities. There is a random period of time spent sleeping between subsequent individual browser requests to simulate the user's think time. TPC-W specifies three different types of web interaction mixes by varying the ratio of browse to buy: browsing, shopping, and web-based ordering. Table 1 shows the web interaction frequencies for each type of interaction mix. The primary performance metric tested by TPC-W is the number of Web Interactions Processed per Second (WIPS), which measures the system throughput. Another metric is the Web Interaction Response Time (WIRT), which measures the system response latency.

Table 1 Web interaction mixes.

Web interaction	Browsing mix (%)	Shopping mix (%)	Ordering mix (%)	Weblet name
Browse	95	80	50	
Home	29.00	16.00	9.12	TPCW_home_interaction.wlet
New products	11.00	5.00	0.46	TPCW_new_products_weblet.wlet
Best sellers	11.00	5.00	0.46	TPCW_best_sellers_weblet.wlet
Product detail	21.00	17.00	12.35	TPCW_product_detail_weblet.wlet
Search request	12.00	20.00	14.53	TPCW_search_request_weblet.wlet
Search results	11.00	17.00	13.08	TPCW_execute_search.wlet
Order	5	20	50	
Shopping cart	2.00	11.60	13.53	TPCW_shopping_cart_interaction.wlet
Customer registration	0.82	3.00	12.86	TPCW_customer_registration_weblet.wlet
Buy request	0.75	2.60	12.73	TPCW_buy_request_weblet.wlet
Buy confirm	0.69	1.20	10.18	TPCW_buy_confirm_weblet.wlet
Order inquiry	0.30	0.75	0.25	TPCW_order_inquiry_weblet.wlet
Order display	0.25	0.66	0.22	TPCW_order_display_weblet.wlet
Admin request	0.10	0.10	0.12	TPCW_admin_request_weblet.wlet
Admin confirm	0.09	0.09	0.11	TPCW_admin_response_weblet.wlet

4.2. TPC-W for weblet environment

We use the java-based TPC-W source code developed by University of Wisconsin at Madison [7] as our basis. It is modified to fit our Weblet environment. This source code includes three parts: client side programs which implement an emulated browser, server side programs which implement TPC-W specified web logic by using 14 servlets, and a database population program which generates the database structure and populates the database. On the server side, we convert the 14 servlets to weblets, as indicated in the last column of Table 1. On the client side, the servlet requests are substituted with weblet requests. EBs send requests to a proxy server instead of directly to a web server, which is more accurate simulation of the current web systems. Also, the original source code uses DB2 database. We substitute it by MySQL database because MySQL is a popular open source database. Modifications are made to compensate the differences in DB2 and MySQL. Bug correction and performance improvement efforts are also made to the code. For example, we use a connection pool to access the database. We populate databases on two web servers with different sizes. The details of two populated databases are given in Table 2(a) and (b). The large database is approximately 1.65 GB and the small database is approximately 1 GB. We use the web server 1 with a small database to represent a small web site and use the web server 2 with a large database to represent a large web site.

4.3. Weblet and data migration

Based on the migration-decision rules for data objects and weblets discussed in Section 3, we decide to migrate six weblets and two tables. The data in COUNTRY

Table 2 Database tables characteristics.

Table name	Number of rows
(a) Tables in web server 1	
Customer	864,000
Address	1,728,000
Orders	777,600
Order_Line	2,332,800
CC_XACTS	777,600
Item	1,000
Author	250
Country	92
(b) Tables in web server 2	
Customer	1,440,000
Address	2,880,000
Orders	1,296,000
Order_Line	3,888,000
CC_XACTS	1,296,000
Item	10,000
Author	2,500
Country	92

table are read-only and can be migrated easily, so they belong to data type 1. The data in the ITEM and AUTHOR tables are read-only for clients and can be modified by administrators using Admin Confirm weblet. According to TPC-W specification, the probabilities for Admin Confirm weblet in three workload mixes are 0.09, 0.09, and 0.11%. Essentially, the data in ITEM and AUTHOR tables are seldom changed. Thus they also belong to data type 1. The data in remaining tables are highly shared and need frequent updates, so they belong to data type 3 and are not suitable for migration.

Though COUNTRY table is read-only, no weblets accessing this table are suitable for migration. So we do not migrate the COUNTRY table. To determine whether to migrate ITEM and AUTHOR tables, we consider their migration values. According to TPC-W specification, the total access frequencies of Home, New Products, Product Detail, Search Request, and Search Result weblets in three workload mixes are approximately 84, 75, and 49%, respectively. Their access frequencies are relatively high. Also, the sizes of ITEM and AUTHOR tables are approximately 700 and 100 KB for the small web site, and 7,000 and 1,000 KB for large web site, which are relatively small. So ITEM and AUTHOR tables can be easily migrated.

New Products, Product Detail, Search Request, and Search Result weblets only need to access ITEM and AUTHOR tables. Since ITEM and AUTHOR tables are suitable for migration, it is cost effective to migrate these four weblets. In addition to accessing ITEM and AUTHOR tables, Home weblet has a processing component which queries the CUSTOMER table via a customer ID to obtain the customer's first and last names. However, this processing component is only invoked at the first interaction of a user session requested by a return customer. If Home weblet remotely accesses the CUSTOMER table, then it only incurs one communication between the proxy and the web server. The communication cost can be further

reduced by caching query responses on the proxy server. So, migrating Home weblet is also cost-effective. Customer Registration weblet is used to provide a web page for customer registration and does not need to access any data in the database. So, it definitely can be migrated. The remaining weblets are not suitable for migration since the data objects required by these weblets are highly shared and need frequent updates.

At the beginning of each experiment, the proxy server only runs MySQL service and has an empty database and there are no weblets in the cache. All weblet requests will be forwarded to the web server, and the web server will check with Weblet Migration Manager to decide if the requested weblet should migrate. In our experimental study, all the weblets are predefined as migratable and non-migratable in the Weblet Migration Manager according to the analysis discussed above. So, when the web server gets requests accessing the six migratable weblets, it will migrate the weblets and the associated tables to the proxy server. The migration of data objects uses the data migration XML message discussed in Section 3. When a migrated weblet and its data objects come to the proxy server, the Weblet Cache Manager first caches the weblet, then creates and populates tables, and finally executes the migrated weblet. Proxy server returns the response back to the requesting client. Each migrated weblet and its carried data objects have TTL values assigned by the web server. The TTL values of weblet and its data objects may be different.

5. Experimental study

The goal of our experimental study is to compare the performance of weblet system with current Web system and dynamic content caching approach. We discuss the three cases in the following.

Current Web System: This is to simulate the conventional web systems. Clients send weblet requests to the proxy server, and the proxy server simply forwards the client requests to the web server and forwards the responses back to the corresponding clients. There is no weblet migration from the web server to the proxy server.

Dynamic Content Caching: Based on the design idea of [18], we implement the servlets with dynamic content caching on the Web server (Tomcat Server). The TPC-W specified web interactions are implemented as 14 servlets. Among them Home, New Products, Best Sellers, Product Detail, Search Request, Search Results, and Customer Registration are cacheable servlet requests. The rest are non-cacheable servlet requests. In this case, there is no proxy between clients and the web server. Clients directly send servlet requests to the Web server. Web server first checks if there are cached responses to the previous requests for the same content. If yes, then the requests are served from the cache. If not, the requests are served by corresponding servlets. TTL value of 5 min is assigned to each cached response. If the TTL values of cached responses expire, the requests are served by the corresponding servlets instead of returning the cached responses. In order to maximize the gain of this case, we use unlimited cache size.

Weblet case: In this case, clients send weblet requests to the proxy server and the proxy server first checks if the weblet is cached in the Weblet Cache Manager. If yes

and the cached weblet is valid, the request will be served on the local weblet engine. When the cached weblet needs to access its cached data objects, it checks the TTL values of the data objects. If the TTL value of a data object expires, then the weblet checks its validity with the web server. Otherwise, the data object is used directly.

In the following subsections, we discuss the experimental setup and performance results.

5.1. Experimental setup

The general Web infrastructure consists of a number of clients, proxy servers, and web servers. Clients generally have fixed proxy servers. Since the proxy servers usually work independently of each other, we consider a single proxy server in our experimental setups. We design two experimental setups to study the weblet system performance. In *Experiment 1*, we consider a single web server and one proxy server. The web server is configured as web server 1 with the small database as discussed in Section 4.2. Since a proxy server may cache web requests for multiple web sites, we design *Experiment 2* to study the performance impact on weblet system when the proxy caches weblets from two different web servers. Thus, the setup for Experiment 2 includes two web servers and one proxy server.

To measure the real impact of communication cost over the Internet, we setup the web servers at Virginia Tech Campus and the proxy server and client machines at UTD (University of Texas at Dallas) Campus. The web server platforms are Pentium IV 1.6 GHz single CPU notebooks with 256 MB RAM and 30 GB disk. The two web servers are set up as two different online book shopping web sites. For both experiments, the proxy server platform is a Pentium IV 2.2 GHz PC with 256 MB RAM and 40 GB disk. It runs Squid proxy server V2.5.stable4. Also, weblet extension module and MySQL V.4.1.0-alpha are installed on the proxy server. For weblet system and conventional web system, Apache V.2.0.44 web server with the weblet extension module and MySQL V.4.1.0-alpha database server are installed on the web server platform(s). For dynamic content caching mechanism, Tomcat Server 5.0.28 and MySQL V.4.1.0-alpha database server are installed on the web server.

In order to prevent the web server from becoming a bottleneck by the image requests, we do not populate the item images on web server and disable the “request images” parameter of EBs for all experiments. Disabling static image requests won’t impact the result comparisons for dynamic web requests.

For all experiments, we use eight client machines to generate web traffic. Eight Pentium III 800 MHz PCs with 256 MB RAM are used to simulate the clients. The TPC-W Emulated Browser (EB) software is installed on the client machines. All of machines run RedHat Linux 7.3. The client machines are connected to the proxy server through a 100 Mbps Ethernet switch. The average round trip time among the machines at UTD LAN is 0.35 ms. The average round trip time among the machines at Virginia Tech LAN is 0.43 ms. The average round trip time between UTD Campus and Virginia Tech Campus is 47 ms.

The client platforms emulate 50, 100, 150, 200, 250, 300, 350, 400, 450, and 500 simultaneous users and measure the system performance. To ensure that the clients do not become a bottleneck, each client machine at most emulates 100 users. The clients simulate the three different workload mixes defined in TPCW as discussed in Section 4.1.

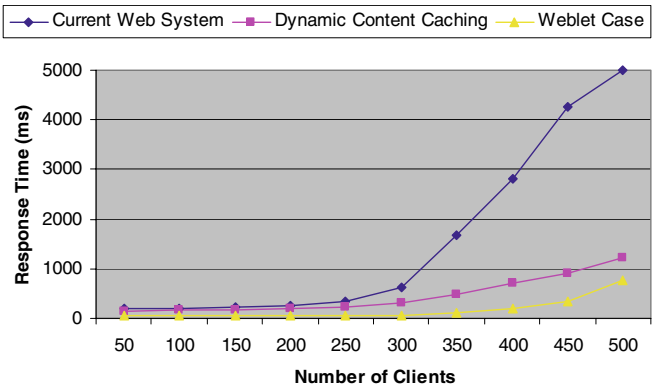
Each experiment is composed of three phases, warm-up phase, steady-state phase, and cool-down phase, which are of time duration 120, 1,200 and 20 s, respectively. During the warm-up phase, the migrations of weblets and their data objects are performed based on predefined migration rules in Weblet Migration Manager. In the steady-state phase, measurements are performed. System performance is measured in term of client response latency, web server's workload, and throughput. We use CPU utilization information as server load metric. The web servers use the sysstat utilities [21] to collect CPU utilization information from the Linux kernel every 1 min. Clients collect the response time and throughput of weblet requests. The TTL for migrated data objects is 5 min. We assume that weblets will not be updated in the duration of interest.

5.2. Experiment 1 results

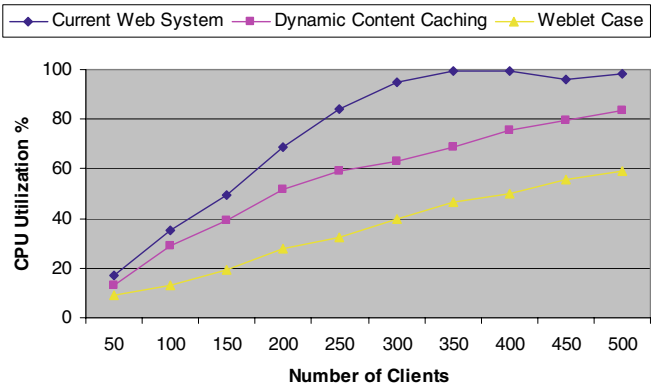
First, we consider Experiment 1 with browsing mix. Figure 5(a) shows the response times in three cases. As we can see, Weblet case achieves the best response latency. The response times in Weblet case and Dynamic Content Caching case are significantly lower than that in Current Web System case. Also, the response time increases drastically in Current Web System case after the number of clients exceeds 300, but increases smoothly in Weblet case and Dynamic Content Caching case. The response times in Weblet case and Dynamic Content Caching case are 660 and 400% shorter than that in Current Web System case when the number of clients reaches 500. In the Dynamic Content Caching case, all browsing Servlets are cacheable requests. According to TPC-W specification, the percentage of browsing requests is 95% in browsing mix. That means 95% requests might be served from cache. For the requests that are served from cache, their response latencies are significantly reduced because the responses come from memory and the database accesses are avoided. The time that dynamic requests spend on database accesses is a big part of the response latency. The number of cache hits increases as the number of clients increases. So this case achieves a better response time, especially when there are lots of clients. But this case has two disadvantages. First, each request has to be served from Web server. The physical network latency always exists regardless of cached responses or non-cached responses. Second, the responses are served from cache only when the clients request for exactly the same content as that in a previous cached request. This restricts the number of cache hits. In Weblet case, the migrated weblets are directly served on proxy, which is very close to clients. So the network latency is significantly reduced.

Figure 5(b) shows the CPU utilization in the three cases. The CPU utilization in Dynamic Content Caching case is lower than that in Current Web System case because the database accesses are avoided for cached responses and the database operations takes lots of CPU cycles. Although Dynamic Content Caching case can reduce the CPU utilization to some extent, all requests still need to be processed on the web server. In Weblet case, caching weblet on proxy servers can successfully offloads the web server load. So the CPU utilization in Weblet case is lowest one.

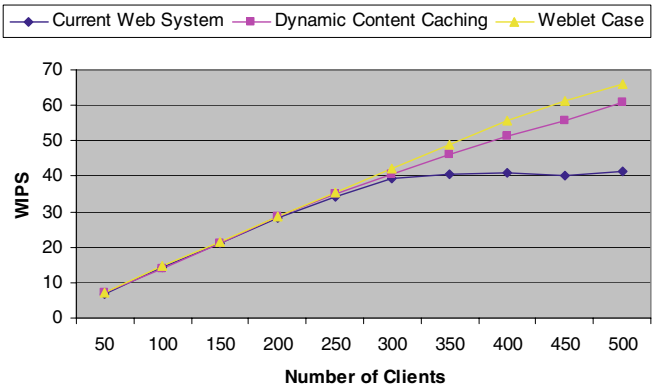
Figure 5(c) shows the throughput (number of interactions per second) comparisons. In Current Web System case, the web server becomes saturated when it reaches 350 clients. So it has the worst throughput. In the Dynamic Content Caching case, cached responses reduce the database access bottleneck to some extent. So it



(a) Response Time



(b) CPU Utilization



(c) Web Interaction per Second

Figure 5 Browsing mix in Experiment 1.

achieves the better system throughput. In Weblet case, the web server is offloaded significantly due to weblet migration. Thus, it achieves the best system throughput.

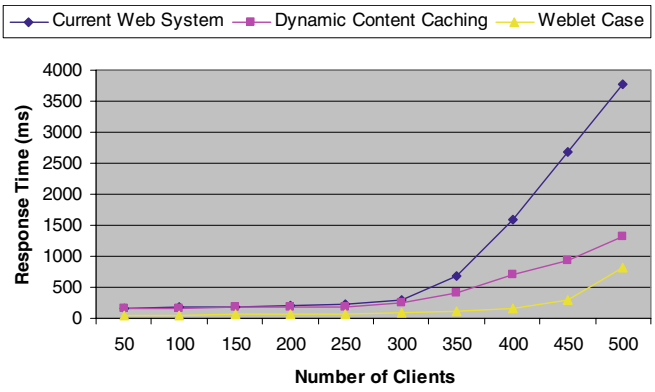
Next we consider shopping mix for Experiment 1. Figure 6 shows that the response time, CPU utilization, and system throughput under the shopping mix. Similar to browsing mix, caching weblet on proxy servers achieves the best results on the response time, CPU utilization, and system throughput metrics. The results of Dynamic Content Caching case are better than those in Current Web System case. Comparing Figure 6 with Figure 5, we can see that in Current Web System case, the response time of browsing mix is significantly higher than that of shopping mixes after the number of clients exceeds 300. The reason is that the browsing mix generates more "select" SQL queries than shopping mix, while the shopping mix generates more "update" SQL requests. The "select" requests usually take more time and resources than "update" requests. Thus database accesses become the bottleneck of the system. Also, we can see that in Dynamic Content Caching case, the improvement is less than that in browsing mix because the percentage of cacheable requests is reduced.

Finally, we consider ordering mix for Experiment 1. Figure 7 shows that the response time, CPU utilization, and system throughput for the ordering mix. Similar to browsing and shopping mixes, caching weblet on proxy servers achieves the best performance and the Dynamic Content Caching case has better results than those in Current Web System case. Comparing Figures 5–7, we can observe that in Current Web System case, the web server becomes saturated when it has 350, 450, and 500 clients for browsing, shopping, and ordering mixes, respectively, but the web server in Weblet case remains stable. The different saturation points reflect the different workload intensities imposed by different mixes. The browsing mix generates the heaviest workload because it generates more "select" SQL queries than the other mixes. We can also see that both Weblet and Dynamic Content Caching cases achieve different system throughput improvement under three mixes. For both cases, the system throughput under browsing mix has the most significant improvement and the system throughput under ordering mix has the least improvement. For Weblet case, the reason is that according to TPC-W specification, the six migrated weblets in Weblet case can offload the web server load by 85, 78, and 62% under browsing, shopping, and ordering mixes, respectively. The different level of offloading results in different system throughput improvement. For Dynamic Content Caching case, the percentages of cacheable requests are 95, 83, and 63% under browsing, shopping, and ordering mixes, respectively. Different percentages results in different system improvement.

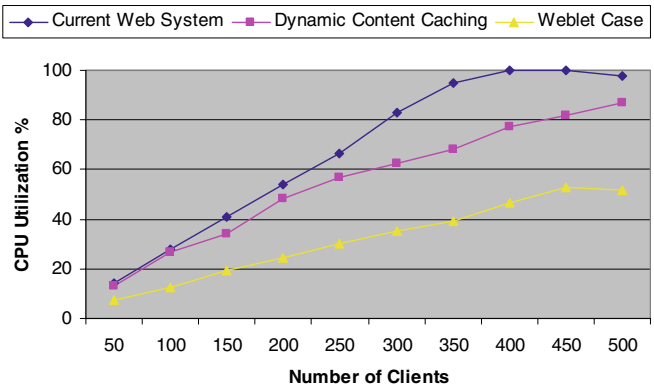
5.3. Experiment 2

Now, we consider Experiment 2 and study the performance of the weblet system with proxy caching weblets from two different web servers. Similar to Experiment 1, three different workload mixes are considered but we only compare the Current Web System and Weblet cases. We consider the performance measurement on one web server at a time. First, we fix the load on web server 2 at 100 clients and measure web server 1 behavior and response time. Then we reverse the setup and measure the behavior of web server 2.

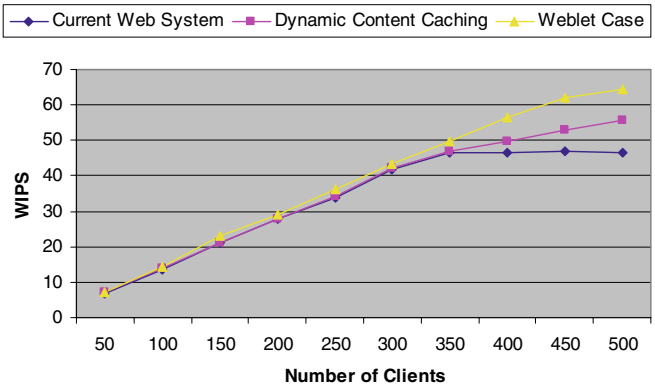
We measure the system performance under all three workload mixes and the results are similar to that in Experiment 1. Essentially, caching weblet on proxy



(a) Response Time

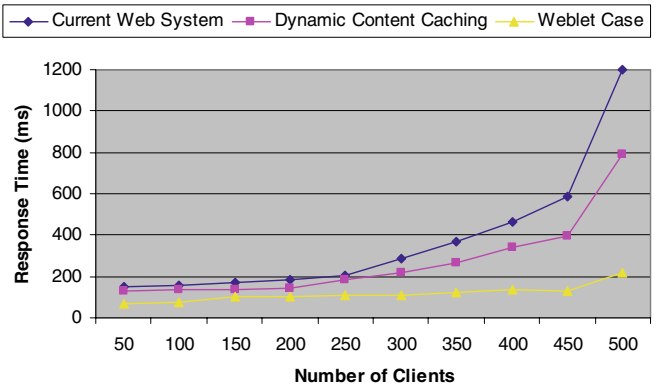


(b) CPU Utilization

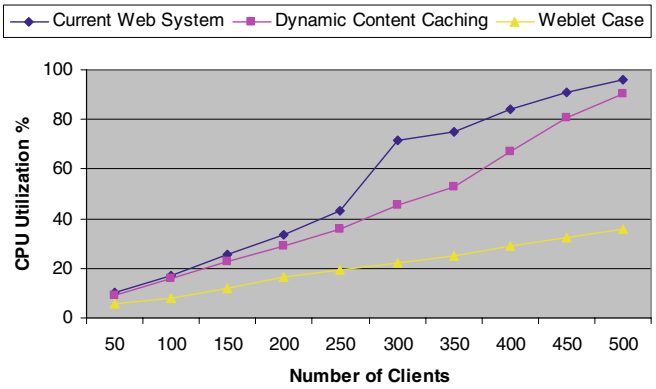


(c) Web Interaction per Second

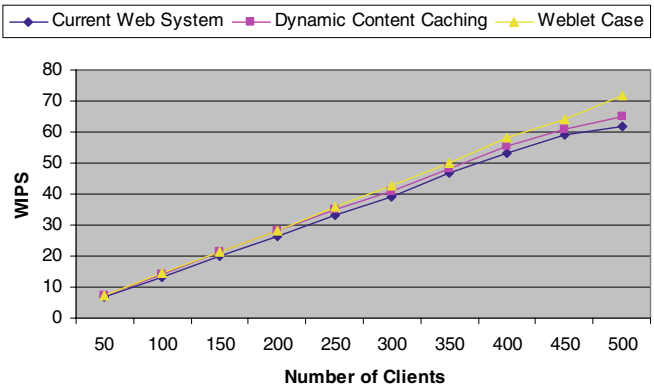
Figure 6 Shopping mix in Experiment 1.



(a) Response Time



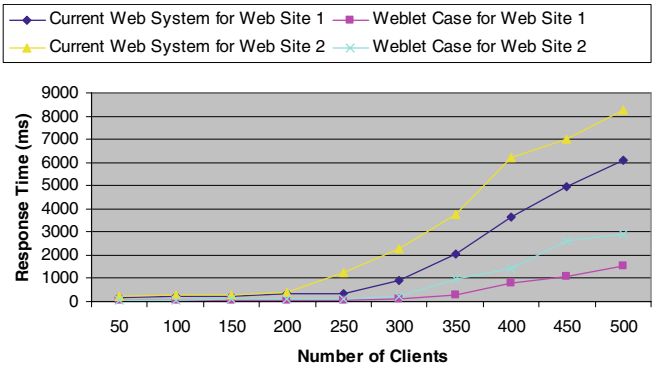
(b) CPU Utilization



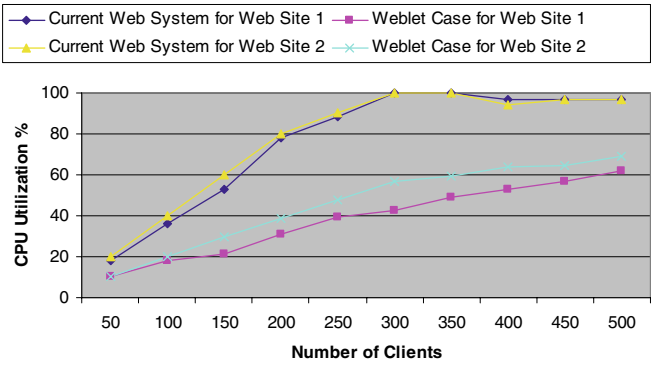
(c) Web Interaction per Second

Figure 7 Ordering mix in Experiment 1.

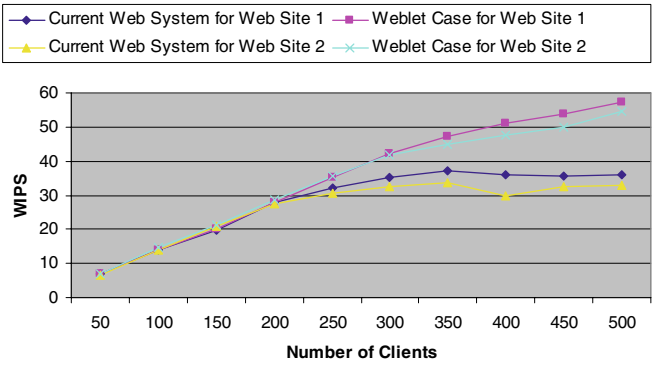
server can significantly reduce response time, offload the web server load, and improve the system throughput for both web servers. Here we only present the response time, CPU utilization, and system throughput of two web servers under browsing mix, as shown in Figure 8. From this figure, we can observe that the response time of web server 2 is significantly higher than that of web server 1. The



(a) Response Time



(b) CPU Utilization



(c) Web Interaction per Second

Figure 8 Browsing mix in Experiment 2.

reason is that web server 2 has a bigger database than that of web server 1. The SQL requests usually take more time and resources in a bigger database than in a smaller database. Compare the response time and the system throughput of web server 1 in Weblet case in Experiment 2 with that in Experiment 1, we can see that the Experiment 2 yields worse performance. The response time in Experiment 2 is significantly longer than that in Experiment 1 when the number of clients exceeds 300. The system throughput is significantly lower in Experiment 2 than that in Experiment 1 when the number of clients exceeds 400. The reason of degraded performance is obvious. In Experiment 2, the proxy server serves two different web servers, and it caches and runs more weblets locally. The problem can be solved easily by adding extra proxy servers to share the load.

6. Remarks

The weblet environment significantly improves system performance in terms of client response latency, web server throughput, and workload. Also, it is very flexible for weblet designers. The weblet designers can specify application specific migration rules based on required data consistency level and system environment to obtain the desired performance. Currently we only consider migrating read-only data objects with the weblets. Actually, we are also developing full scale migration decision policies to determine whether to migrate weblet and data objects at a fine grained level considering various access patterns. For example, Shopping Cart weblet can also be migrated to proxy servers even though the shopping cart is a frequently updated data object. In real life, users usually go through a fixed proxy server to issue shopping requests for an extended duration. Also, a user always reads and updates his/her own shopping cart and his/her shopping cart is never updated by other users. The Shopping Cart weblet can be migrated to a proxy server together with its data objects. The proxy server can locally process shopping requests and update the shopping cart. Updates can be sent back to the web server after an extended time period. According to TPC-W, the percentages of shopping cart interaction in three workload mixes are 2, 11.6, and 13.53%, respectively. Thus, migration of Shopping Cart weblet can still result in performance gains. We can go even further and migrate the ordering weblets to proxy servers. Ordering weblets need access user personal information, such as name, address, credit card, etc. These information can be migrated to the proxy server for ordering check. However, security issues on these critical data should be considered. In [14], we discussed how to protect weblet code and weblet carried data objects. A simpler approach is to use one-way hash function to store the information. The proxy server only has hashed values since only comparison is needed. Another issue to be considered for migrating the ordering weblets is that it needs to verify the available quantities of merchandises. Several approaches can be used for this step. First, the ordering weblets can interact with back-end servers to perform the verification. Also, we can let the ordering weblets estimate the remaining quantity based on historical selling rate. The ordering weblets go back to the server only when the potential quantity is low. Another way is to let the web servers partition the remaining quantity of merchandise to let ordering weblet "own" a number of merchandise and periodically update their own quantities.

Weblet environment can be integrated not only with existing HTTP protocol, but also many newly proposed frameworks. For example, fragment caching is an effective technique to improve the web performance. This feature is supported by Microsoft ASP.NET, IBM WebSphere Application Server, and other products. ESI implements the fragment cache at edge servers. We can easily integrate the weblet environment with ESI. In weblet environment, the web server decomposes a web page into fragments. If the page only contains static fragments, then the assembly can be done using usual fragment caching technique. If some fragments of the page involve dynamic processing, then the processing logic can be implemented as weblets and the weblets can be marked as dynamic fragments in the page template. The web server not only provides templates to a proxy server but also migrates the corresponding weblets to the proxy server. When a client issues a request, the proxy server dynamically assemblies the response based on the corresponding template and page fragments. If there are dynamic fragments in the template, then the corresponding weblets are invoked before assembly. This combined approach can have the benefits of fragment caching and weblet caching and yield better performance for web system.

7. Summary

We discuss an approach to enable the proxy servers to cache web processing components such that conventionally noncachable objects can now be cached on proxy servers to increase the cache hit rate. In our approach, we implement web processing components as weblets that can migrate to client side proxies to provide required services. When a client requests a certain service and the corresponding weblet is cached at the proxy, it can be activated locally without going to the original web server. To support this approach, we developed a weblet supporting environment to facilitate robust and secure weblet migration and execution. We have conducted experiments to study the performance of the weblet environment by using the industrial standard e-commerce benchmark TPC-W. The experimental results show that the weblet environment significantly improves system performance in terms of client response latency, web server throughput, and workload. The weblet environment also provides high security and considers both the protection of proxy servers and the weblets. We have implemented a prototype weblet system including the weblet environment and system modules discussed in this paper.

The weblet approach can yield a significant impact on the web system infrastructure. With the current trend, we believe that more and more web applications will be computation oriented. With existing proxy protocol, cache hit rate will drop due to these processing units. With the weblet approach, these processing units can be migrated and cached and a higher proxy cache hit rate can be achieved. Currently, the migration decision making module implemented on the web server is still primitive. Research work needs to be conducted in several directions, including the design of an integrated data migration and sharing protocol for weblets and the development of cost-effective security algorithms for weblet code and data protection.

In addition to supporting weblet execution, the weblet engine can also be used to provide a uniform execution environment for many other applications. For example, content adaptation became an important task recently due to the growing diversity

on end-user devices, such as PDA, mobile phone, and WebPad. Proxy servers are the ideal platforms to perform content adaptation and information filtering for end users. Various content adaptation and filtering programs can be implemented as weblets and migrated to proxies to perform the desired task. Web service is another application area that can make use of the weblet concept. Though currently not considered, web service units can be implemented as weblets and migrated to proxy servers to provide services close to the end users. We will investigate possible extensions to Weblet Engine to allow proxy servers to form a widely distributed infrastructure to support various applications.

References

1. Aglet: http://www.trl.ibm.co.jp/aglets
2. Apache Web Server: http://httpd.apache.org
3. Arlitt, M., Krishnamurthy, D., Rolia, J.: Characterizing the scalability of a large web-based shopping system. ACM Transactions on Internet Technology **1**(1), 44–69 (2001)
4. Cache Array Routing Protocol (CARP) and Microsoft Proxy Server 2.0: http://www.msdn.microsoft.com/library/backgrnd/html/carp.html
5. Cao, P., Irani, S.: Cost-aware WWW proxy caching algorithms. In: Proceedings of the USENIX Symp. on Internet Technologies and Systems, December, 193–206 (1997)
6. Cao, P., Zhang, J., Beach, K.: Active cache: caching dynamic contents on the web. In: Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98), 373–388 (1998)
7. Crain, H.W., Rajwar, R., Marden, M., Lipasti, M.H.: An architecture evaluation of Java TPC-W. In: Proceedings of the Seventh IEEE Symposium on High-Performance Computer Architecture, January, 229–240 (2001)
8. Datta, A., Dutta, K., Thomas, H., VanderMeer, D., Ramamritham, K.: Accelerating dynamic web content generation. IEEE Internet Computing **6**(5), 26–35 September (2002)
9. Edge Side Includes: http://www.esi.org
10. Fan, L., Cao, P., Almeida, J., Broder, A.Z.: Summary cache: a scalable wide-area web cache sharing protocol. IEEE/ACM Trans. Netw. **8**(3), 281–293 (2000)
11. Feldmannm, A., Caceres, R., Douglis, F., Rabinovich, M.: Performance of web proxy caching in heterogeneous bandwidth environments. In: Proceedings of INFOCOM, 107–116 (1999)
12. Fielding, R., Gettys, J., Mogul, J.C., Frystyk, J., Masinter, H., Leach, L., Berners-Lee: RFC2616: HyperText Transfer Protocol, HTTP/1.1- http://www.ietf.org/rfc/rfc2616.txt
13. Gao, L., Dahlin, M., Nayate, A., Zheng, J., Iyengar, A.: Application specific data replication for edge services. WWW2003, May, 449–460 (2003)
14. Hao, W., Ma, Q.K., Yen, I.-L., Chen, I.: A Weblet environment to facilitate proxy caching of web. In: Proceedings of Parallel & Distributed Computing and Systems, Marina del Rey, California, November, 797–802 (2003)
15. Jin, S., Bestavros, A.: Popularity-aware greedy-dual-size web proxy caching algorithms. In: Proceeding of the 20th Intl. Conf. on Distributed Computing Systems, April, 254–261 (2000)
16. Ma, Q.K., Hao, W., Li, W., Tu, M.H., Yen, I.-L: A mobile agent system to support secure Internet and web applications. COMPSAC 2003 Workshop, Dallas, Texas, November, 2003
17. Mohapatra, P., Chen, H.: WebGraph: a framework for managing and improving performance of dynamic web content. Special Issue of Proxy Servers in the IEEE J Sel Areas Commun **20**(7), September, 1414–1425 (2002)
18. Rajamani, K., Cox, A.: A simple and effective caching scheme for dynamic content. Tech. Report TR 00-371, Computer Sc. Dept. at Rice Univ. (2000)
19. Roy, J., Ramanujan, A.: Understanding web services. IT Professional **3**(6), 69–73, November (2001)
20. Squid Proxy Server: http://www.squid-cache.org/
21. Sysstat: http://freshmeat.net/projects/sysstat/
22. TPC-W: http://www.tpc.org/tpcw/
23. WebSphere Edge Server: http://www-306.ibm.com/software/webservers/edgeserver/
24. Wessels, D., Claffy, K.: ICP and the Squid web cache. IEEE J. Sel. Areas Commun. **16**(3), 345–357, April (1998)