WEBSITE ARCHITECTURE AND WEB PROGRAMMING CO-DESIGN FOR ACCELERATING HIGHLY PERSONALIZED DYNAMIC WEB PAGES GENERATION

By

YINGCUI QU

Bachelor of Science

Petroleum University of China

Shandong, P.R. China

1999

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
In partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2004

WEBSITE ARCHITECTURE AND WEB

PROGRAMMING CO-DESIGN FOR ACCELERATING

HIGHLY PERSONALIZED DYNAMIC

WEB PAGES GENERATION

Thesis Approved:

Thesis Advisor

Dean of the Graduate College

ACKNOWLEDGEMENT

I would like to express my deep gratitude to my thesis advisor, Dr. Blayne E. Mayfield, for providing me with this precious opportunity to work in the research area of dynamic Web content acceleration technology, for his expert guidance, and for his untiring support, endless patience, and inspiring encouragement throughout my thesis work. This thesis could not have been completed without his constructive comments that significantly improved the presentation and contents of my thesis. My thanks also go to the members of my committee, G. E. Hedrick and John P. Chandler for reading my thesis draft and providing their and invaluable suggestions. To my beloved husband, Zhenzhong, for always being there to help and inspire me whenever I felt frustrated towards my work. My parents, Guishan Qu and Peiyun Wang, receive my deepest gratitude and love for their unselfish dedication and the many years of support at all levels during my undergraduate and graduate studies that provided the foundation for this work.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

Dynamic Web pages have become increasingly popular in e-commerce Web sites because they enable a much wider range of interaction with customers than static HTML pages can provide. However, dynamic Web pages can reduce seriously the Web site performance by re-generating identical requested Web pages that involve a series of time-consuming server side processes such as database queries, data formatting and data output. A high-performance Web site can deliver several hundred static files per second. By contrast, the rate at which dynamic pages are delivered is often one or two orders of magnitude slower [1]. As the number of requests for dynamic Web pages increases, the overloaded Web site can experience the serious performance problems: slow response time, low Web site scalability and poor service quality [2].

Caching on the server side is an important approach for accelerating dynamic Web content delivery. Most existing solutions rely on caching entire pages of dynamically-generated content [3]. To increase flexibility in Web contents as well as to enrich customers' Web experience, most e-commerce Web sites are beginning to employ personalization technology to provide customized Web pages for specific users. The method of page-level caching is infeasible since it supports limited reusability. Some researchers have proposed caching Web page HTML fragments that generally are used in the whole Website, such as the site navigation bar and predefined contents, by introducing the "cache" tag in the scripting language [4]. But, for the highly personalized Web pages, it is hard to find considerable sharable fragments because they vary either in

information or in presentation. Therefore, Web site performance does not necessarily benefit from this approach.

This research presents a new solution to address this problem: *Web programming and Web site architecture co-design.* First, this solution employs a *Web page template* to lay out a whole Web page. A *Web Content Function* is used to define each part in the *Web template.* The *Web Content Function* is programmed as two layers: one is a *Web Information Block Generator*, which servers as an interface to the Web site database; the other is a *Web Information Block Stylizer*, which wraps the information from the *Web Information block Generator* to produce a personalized HTML segment. In the scripting language, two new cache tags are introduced: tags *<CacheInfo> and <CacheHtmLSegment>* respectively are used to trigger processes of caching the *Web Information Block* and the *Web Html Segment*; Second, the system architecture correspondingly has two main caches: cache one is to cache *Web Information Blocks*; cache two is to cache *Web Html Segments.*

The cache size for each cache can be specified by the Web site administrator according to the Web site feature such as a non-personalized Web site and a personalized Website.

The advantages of this solution are as follows:

1. *Web Content Functions* with the two-layer design not only enable flexible Web page structures and highly configurable Web contents, but also exploit the two data levels of a Web page. Two new cache tags introduced in the scripting language are supportive for the *Dual Caching Scheme.*
2. The *Dual Caching Scheme* increases the data reusability by caching two data levels respectively: *Web Information Blocks* and *Web Html Segments.*

CHAPTER II

LITERATURE REVIEW

There are four major fields related to this research: first is the database-driven Web site technology; second is about the dynamic Web content delivery; third is the Web site personalization technology; fourth is the M/M/1 queueing model.

2.1 Database-Driven Web Site Technology

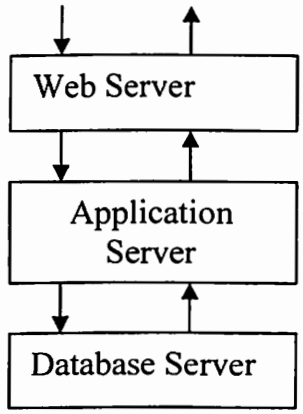


Figure 1. A typical database-driven Web site architecture

Figure 1 shows a typical database-driven Web site architecture. Arrows in this Figure describe the steps that a database-driven dynamic Web page request takes. The process starts when the Web server receives an HTTP request. The Web server uses the URL string to decide whether the application server is needed to process this request. Once the

application server receives the request passed from the Web server, it prepares the requested Web page by the database queries and other server side processes. The page dynamically generated by the application server is passed to the Web server. Finally, the Web server sends this page to the client side [5].

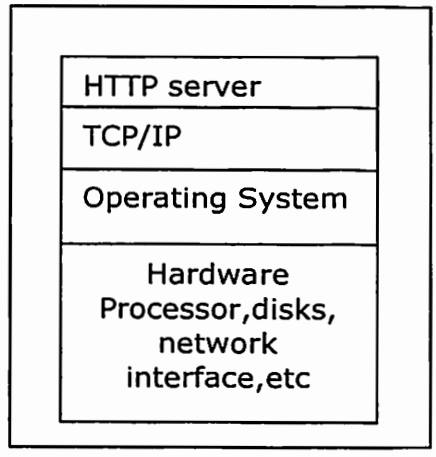


Figure 2. Web server elements

A Web server is a combination of a hardware platform, operating system, networking software, and an HTTP server (Figure 2). The HTTP server is a program that controls the flow of incoming and outgoing data on a computer connected to an intranet or to the Internet. Basically, a Web server listens for the HTTP requests coming from clients over the network and establishes a connection to send the requested file and returns to the listening function [6]. An application server is the software that usually runs on the Web server. It can handle all server side processes between browser-based customers and the back-end databases [6]. Application servers can be implemented in many different ways that are shown in Figure 3: compiled programs, server-side scripts, and combinations of these two technologies—hybrid technologies [7].

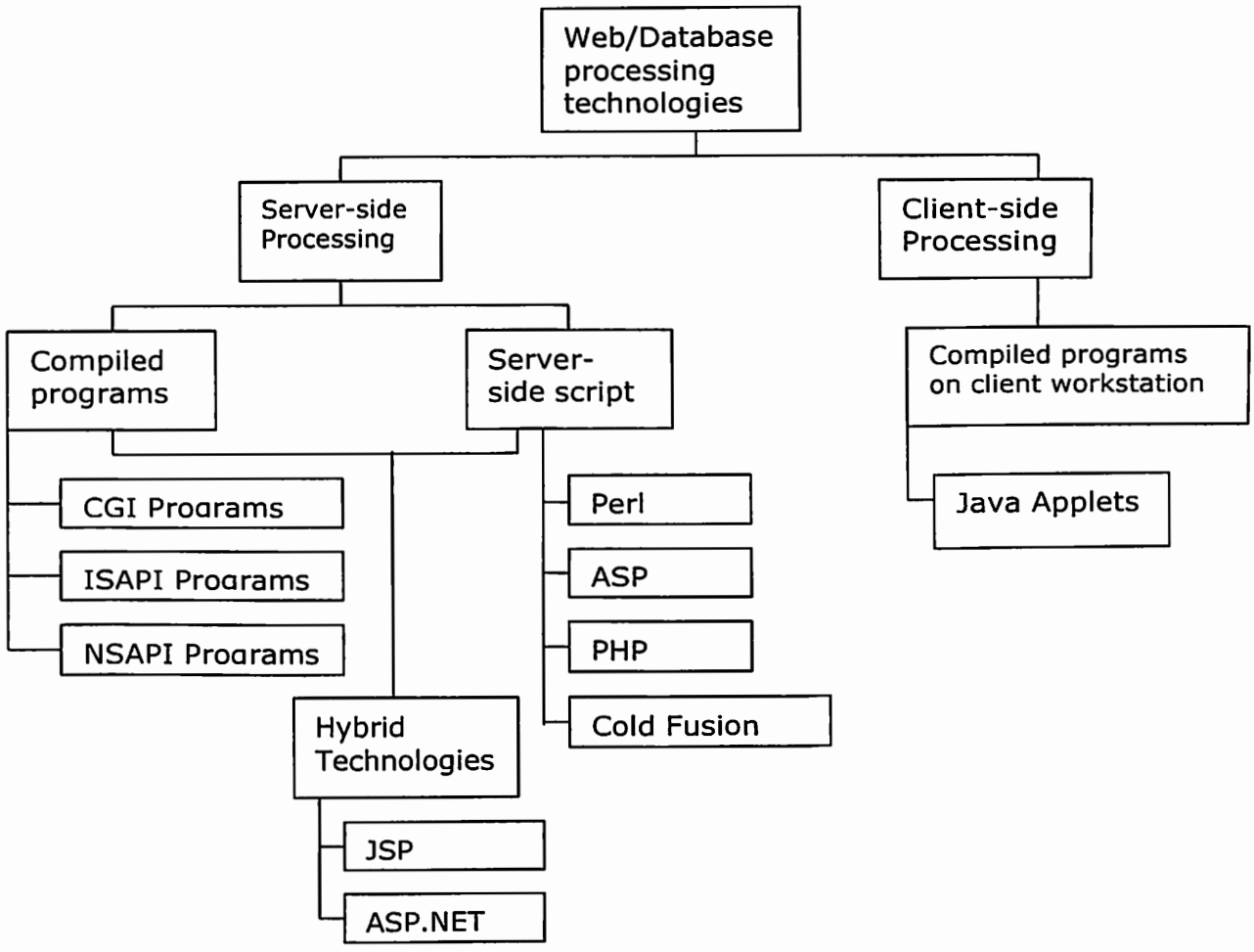


Figure 3. Technologies supporting database-driven Web site

2.2 Dynamic Web Content Delivery

For today's e-commerce Web sites, Web page contents are retrieved mainly from databases because of the demand for high content flexibility. However, the performance of a database-driven Web site can be impaired dramatically by re-generating Web pages from data in the database for every identical request. Caching on the server side is an important approach for accelerating the dynamic Web content delivery.

Most existing solutions rely on caching entire pages of dynamically-generated content [3]. Figure 4 shows a typical database-driven Web site architecture with a page-level cache. Usually, cache and the Web server communicate in order to retrieve Web pages or place new Web pages. The URL of a Web page not only can serve as the key to access cache, but also enables the Web server to determine whether the requested page is dynamic.

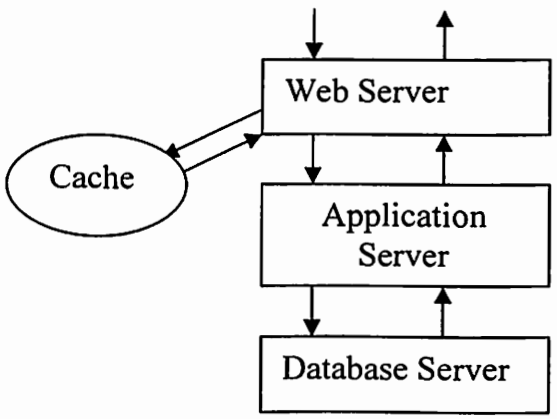


Figure 4. A typical database-driven Web site architecture with a page-level cache

However, caching dynamically-generated pages this way is not effective when dealing with the personalized Web pages. For example, Figure 5 shows a dynamically-generated Web page "flowers4u.php" displayed on an internet browser. For the page-level caching scheme, this page can be cached for the later request.

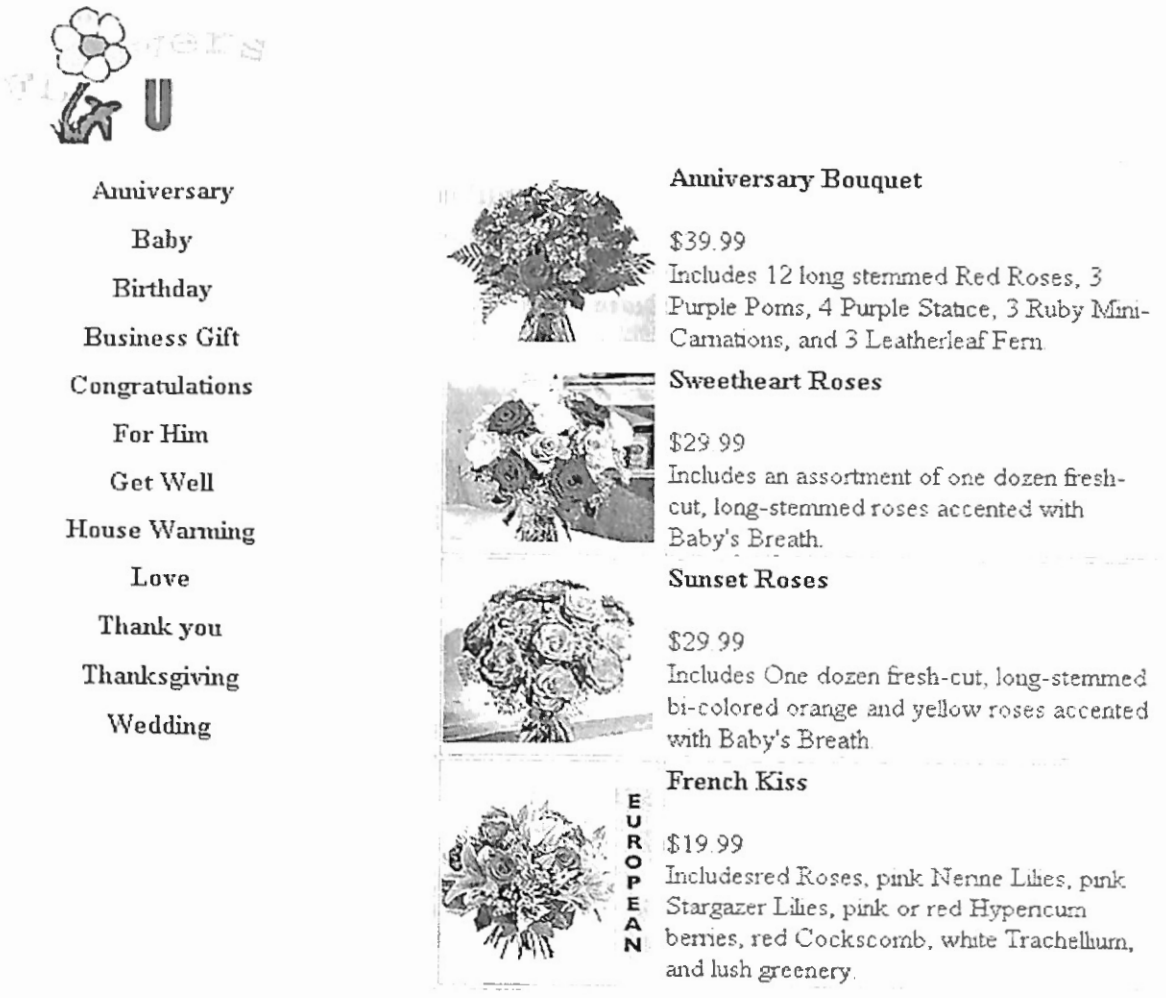


Figure 5. A dynamic Web page

When the this page is personalized by adding user greetings for a specific user, this Web page "flowers4u.php" can be displayed as follows (Figure 6).

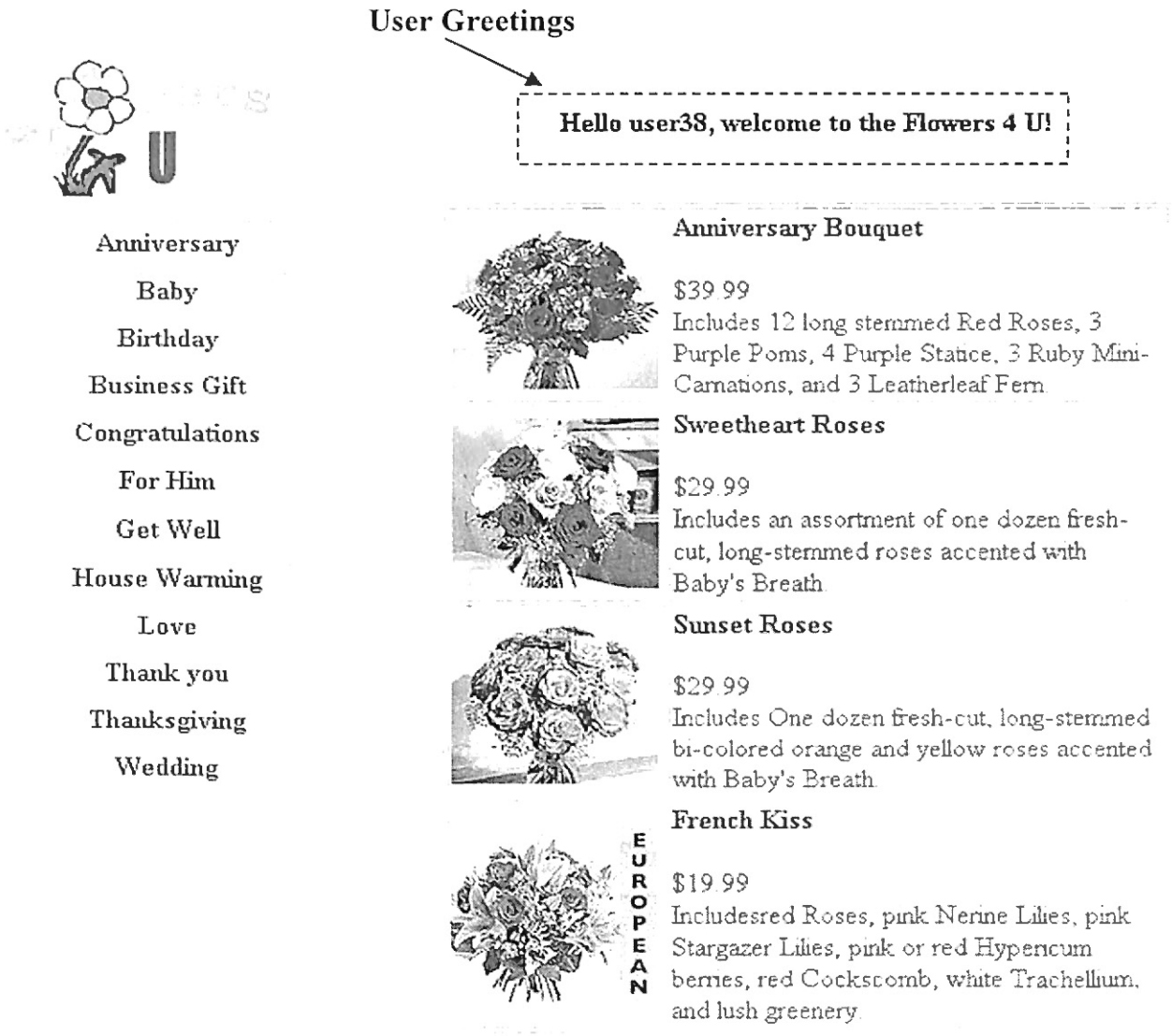


Figure 6. A dynamic Web page with user greetings

This page can be cached just for user38 instead of being shared by all users. Hence, the dramatically impaired cache performance can lead to poor performance at the Web site. To increase reusability of cached objects, another solution proposed is to cache Web HTML fragments that generally are used in the whole Web site, such as site navigation bar and predefined contents, by introducing the "cache" tag in the scripting language [4]. Figure 7 shows a typical database-driven Web site architecture with an HTML Segment-level cache.

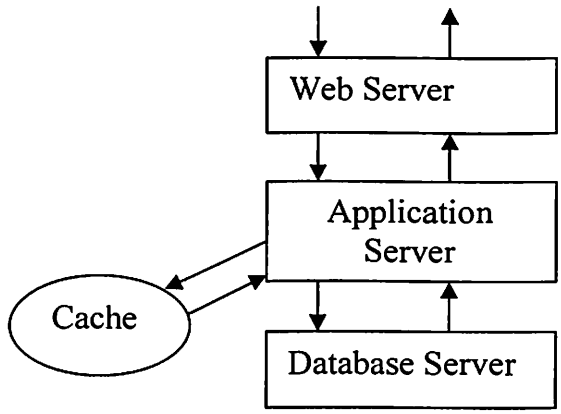


Figure 7. A typical database-driven Web site architecture with an HTML Segment-level cache

For example, the Web page "flowers4u.php?$userID=38" can be divided into several HTML segments: logo, navigation bar, user greetings and main content (Figure 8).

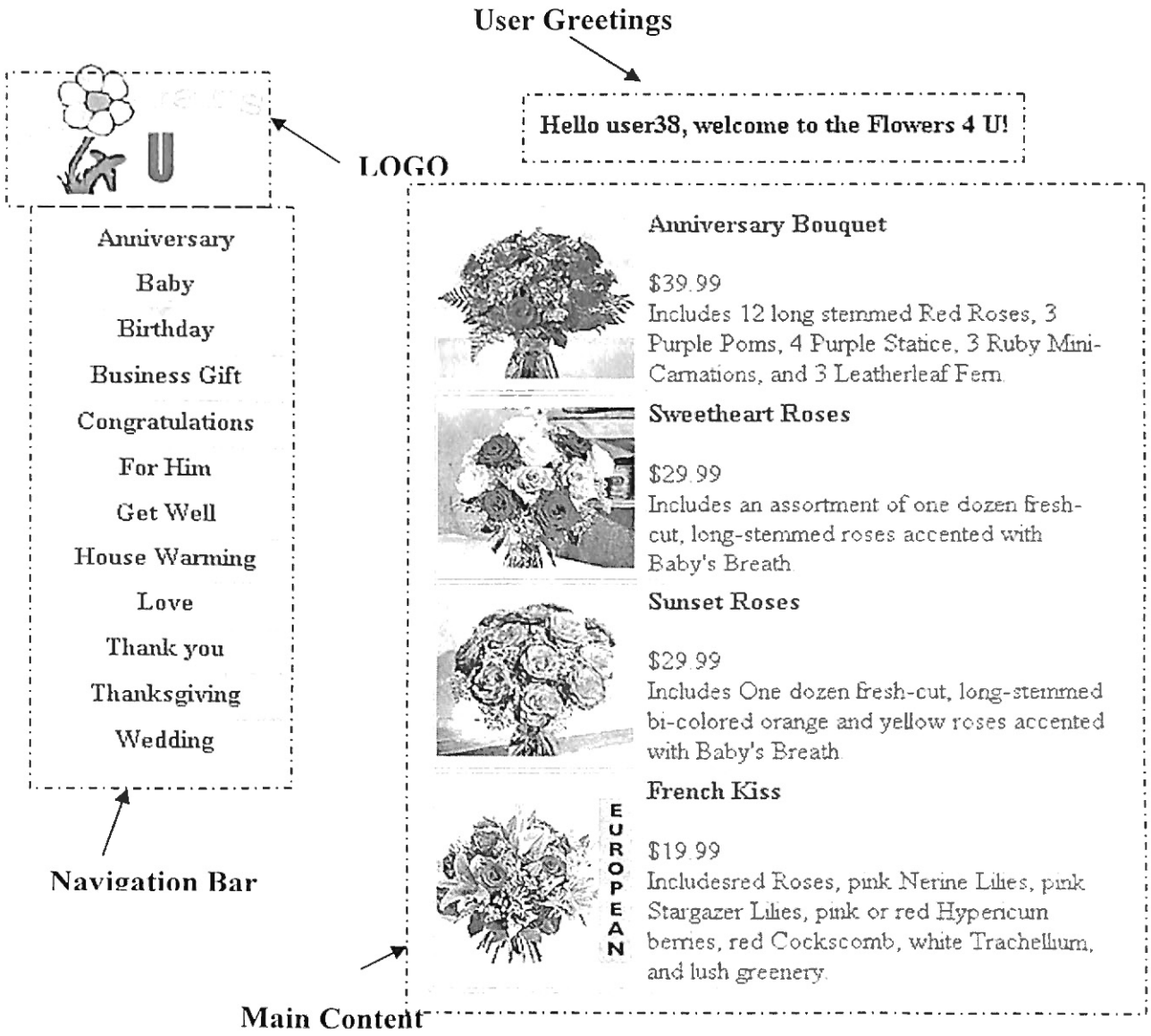


Figure 8. A dynamic Web page consisting of 4 Web content blocks

In addition, the HTML segments, such as logo, navigation bar and main content, can be cached individually and used for all users who visit the page "flower4u.php".

Unfortunately, for the highly personalized e-commerce Website, users even can customize their page presentational styles. All user-shared fragments also can become individualized. Hence, Web server performance doesn't obviously benefit from this approach. For example, the Web page "flowers4u.php?$userID=38" is visited by user38 whose profile shows that the purple theme is user38's preference (Figure 9).

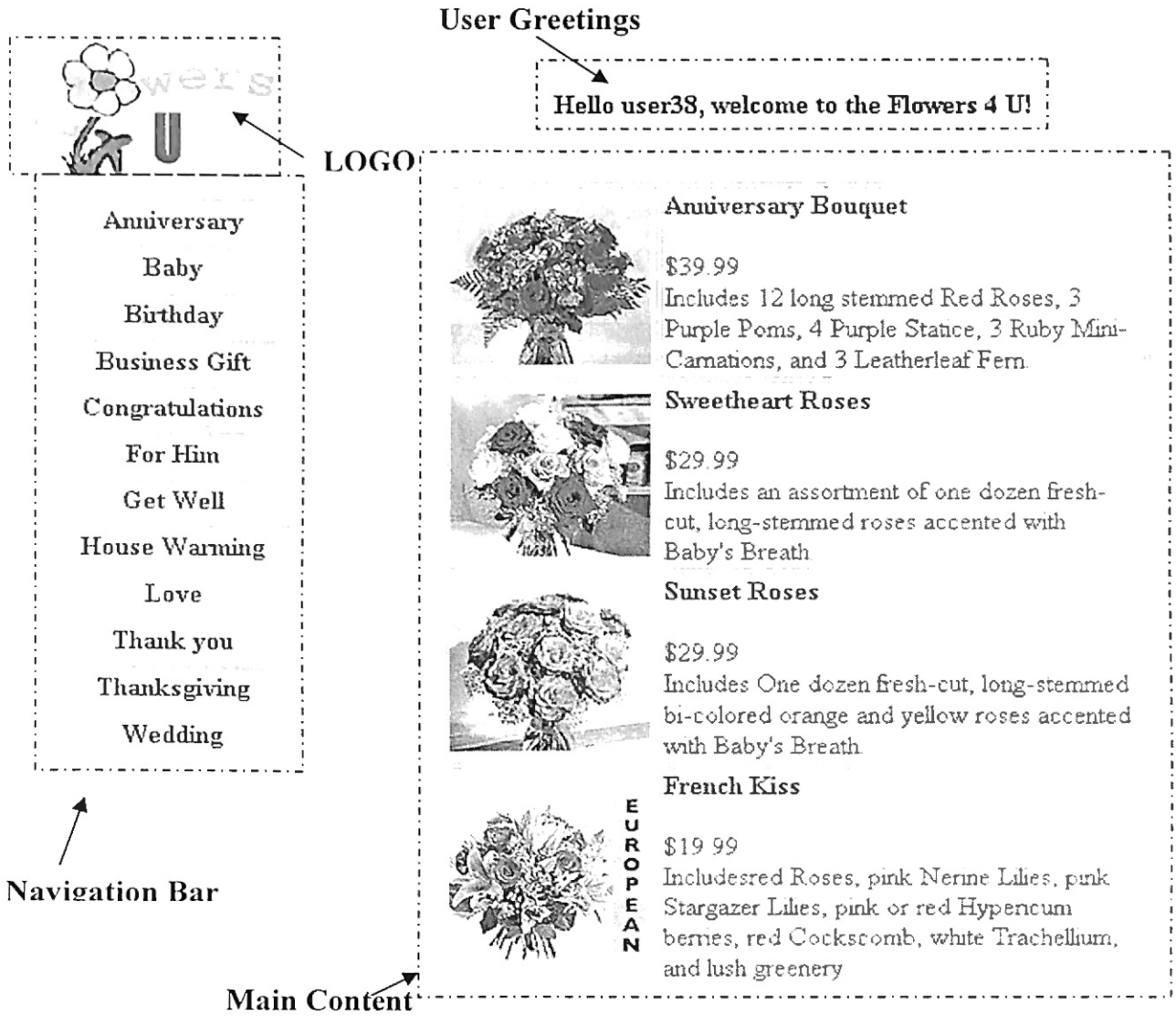


Figure 9. A dynamic Web page consisting of 4 Web content blocks in the purple theme

The Web page "flowers4u.php?$userID=9" is visited by user9 whose profile shows that the blue theme is user9's preference (Figure 10).

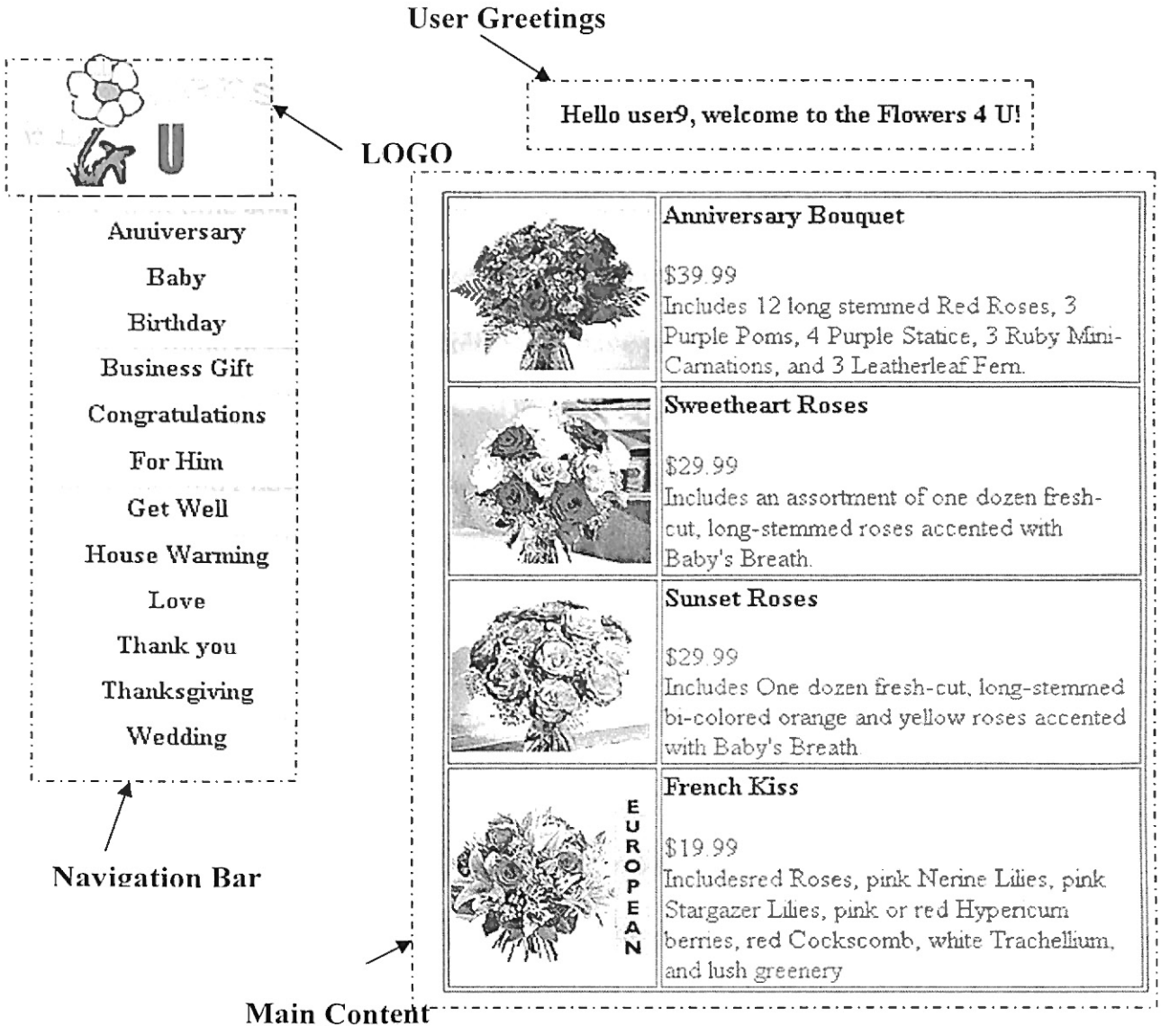


Figure 10. A dynamic Web page consisting of 4 Web content blocks in the blue theme

For the HTML segments, such navigation bar and main content, cannot be used widely for all users who visit the page "flower4u.php".

Associating with Web caching technologies, cache policies for managing cached Web objects have become an active research area in recent years. A comprehensive overview of the cache replacement policies was given in [8]. In [8], authors summarized the important factors of Web objects that can influence the replacement policies. These factors encompass the recency, the frequency, the size, the generation cost, the modification time and the expiration time of the referenced Web object. Some researchers sometimes incorporate several factors into one replacement policy. For example, pAnindya Datta and his colleagues have presented a prediction-based cache replacement called Least-Likely-to-be-Used (LLU). This cache replacement policy considers not only how recently a cached Web object has been referenced, but also whether any user is likely to need it in the near feature [5].

2.3 Personalization Technology

Web site personalization is defined as any action that adapts the information or services provided by a Web site to the needs of a particular user or a set of users [9]. The basic idea behind all personalization systems is simple: accumulate large amounts of Web data from customers' actions on the Website, turn this data into user profiles through the use of data mining techniques, and then use these profiles to deliver personalized Web contents for current on-site customers [9]. User profiles are important components in a personalization system. They can be static or dynamic. Static user profiles consist of information of users' demographics and preferences obtained through online registration processes [10]. They are easy to create, but cannot respond to the changes of users' visit

behaviors. Some researchers have presented dynamic user profiles that can be updated frequently according to the users' current preferences [11]. Both users' session data and Web log files are useful information sources for dynamic profiles creation. In this research, static user profiles are used to personalize users' Web pages.

2.4 M/M/1 Queueing Model

A Web site system together with a browser program (i.e., client) constitutes a client-server system. Many researchers have used queueing models to analyze client-server systems [12]. The basic queuing model is shown in Figure 11.

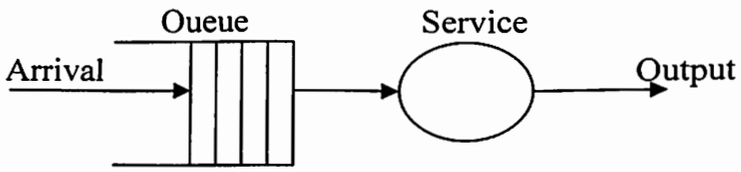


Figure 11. A basic queueing model

A queueing model is characterized by: (1) the probability distribution governing the arrivals (i.e., how often customers arrive), (2) the probability distribution governing the services (i.e., how long it takes to provide service), (3) the queue discipline, specifying how services are provided (e.g., the number of servers, priorities for choosing which customers are served first), (4) the service capacity (e.g., the number of servers) ,(5) queues can have finite capacity or unlimited capacity [13].

The Kendall classification of queuing systems (1953) exists in several modifications. The most comprehensive classification uses 6 symbols [14]:

A/B/s/q/c/p

where:

A is the arrival pattern (distribution of intervals between arrivals).

B is the service pattern (distribution of service duration).

s is the number of servers.

q is the queuing discipline (FIFO, LIFO, ...). Omitted for FIFO or if not specified.

c is the system capacity (Omitted for unlimited queues).

p is the population size (number of possible customers)(Omitted for open systems).

These symbols are used for arrival and service patterns:

M is the Poisson (Markovian) process with exponential distribution of intervals or service duration respectively.

Em is the Erlang distribution of intervals or service duration.

D is the symbol for deterministic (known) arrivals and constant service duration.

G is a general (any) distribution.

GI is a general (any) distribution with independent random values.

The M/M/1 Queuing Model is the simplest queueing model that has a single queue, Poisson arrivals, Poisson service times, unlimited queue size, and first-in first-out queue discipline (Figure 12).

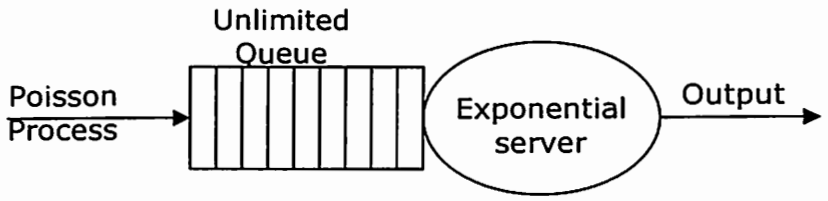


Figure 11. M/M/1 System Model

A Poisson process is a sequence of events "randomly spaced in time [15]." M/M/1 queueing systems assume a Poisson arrival process. This assumption is a very good approximation for arrival process in real systems that meet the following rules: (1) the number of customers in the system is very large, (2) impact of a single customer on the performance of the system is very small, i.e. a single customer consumes a very small percentage of the system resources, (3) all customers are independent, i.e. their decision to use the system are independent of other users. In an M/M/1 queueing system, we assume that service times for customers are also exponentially distributed (i.e. generated by a Poisson process). This assumption is not as general as the arrival time distribution. However, it could still be a reasonable assumption when service times are independent activities [16].

CHAPTER III

RESEARCH METHODOLOGY

To address the problems caused by dynamic Web content delivery in the highly personalized system, this research presents a new solution: *Web site architecture and Web programming co-design*. The research method is composed of three parts: the first part depicts how Web programming can support a *Dual Caching Scheme* and how it can be used to generate highly configurable dynamic Web pages; the second part shows the Web site architecture with a *Dual Caching Scheme*. The interaction among Web site functional components, such as the Web server, the application server, the database and the *Dual Caching Scheme,* is depicted; the last part is to evaluate the proposed solution by comparison with other approaches: the system with the page-level caching scheme and the system with the HTML segment-level caching scheme.

3.1 Web Programming

3.1.1 Web Content Functions with Two-layer Design

In the Web programming aspect of the proposed solution, *Web Content Functions* compose most of a Web page. A *Web Content Function* that can generate a *Web Content Block* is programmed as two layers: one is a *Web Information Block Generator*, which serves as an interface to the Web site database; the other is a *Web Information Block*

Stylizer, which wraps the information from the *Web Information block Generator* to produce a personalized HTML segment. In this research, the server-side scripting language PHP [17] is used to demonstrate how to create dynamic Web pages from *Web Content Functions*. For example, in the Web page shown in Figure 7, every Web page segment can be defined and generated from a Content Function. The example PHP scripts in appendix A show how the Web Content Block *navigation bar* is generated by the Web Content Function *naviBar()* , which includes two main sub-functions: *naviBarInfo()* and *naviBarStyle()*. To fulfill this Web page, a MySQL [18] database server is used to provide data needed in this Web page. Specifically, a database named *eShop* is established in the MySQL server. In this database, there are four tables: *theme*, *users*, *items* and *navibar*. Their table structures are described in appendix E. Besides the current PHP functionalities, two new cache tags need to be introduced to support the *Dual Caching Scheme*.

3.1.2 Two Caching Tags

In the PHP scripting language, two new tags are introduced: tag <*CacheInfo*> and tag <*CacheHtmlSegment*>. The outputs of tagged PHP scripts can be cached when they are generated for the first time. For the later reference, they can be retrieved from cache. The differences between the two tags are as follows:

(1) The output of <*CacheInfo*> tagged PHP scripts contains the unassigned variables that are used for personalization. The retrieved output can become a pure HTML segment and serve as a part of a Web page after the process of assigning. An example is given in Appendix B.

(2) The output of <*CacheHtmlSegment*> tagged PHP scripts is a pure HTML segment that contains personalization information. The retrieved output can serve as a part of a Web page directly. An example is given in Appendix C.

3.2 Web site Architecture with a Dual Caching Scheme

3.2.1 Web Site Architecture

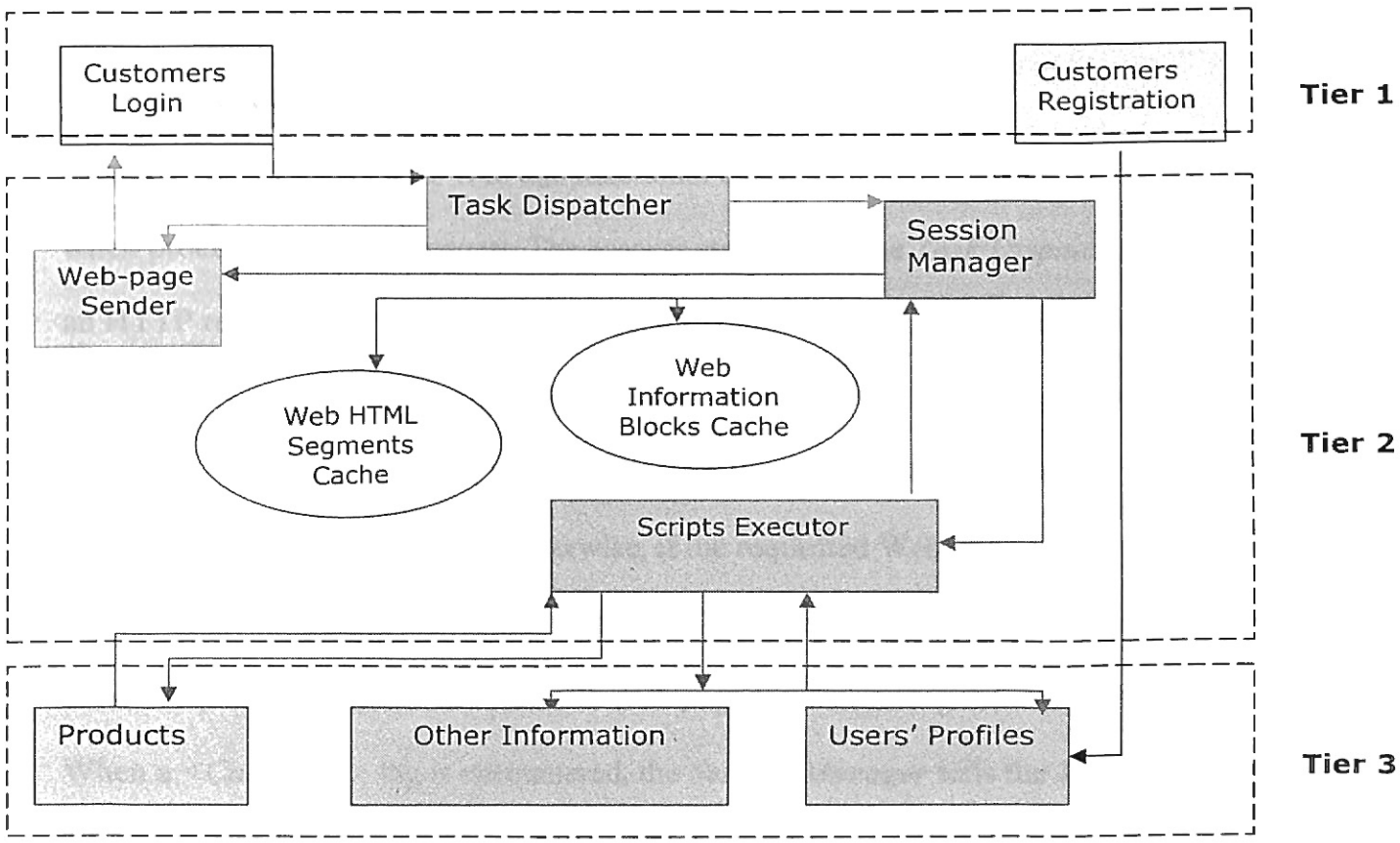


Figure 13. Overall architecture for highly personalized system

Figure 11 shows three-tier Web site architecture.

- The first tier is the customers' login and registration, which can be a front door to an e-commerce Website.
- The second tier consists of a Web server, an application server, and a *Dual Caching Scheme*. The Web server includes two modules: *Task Dispatcher* and *Web-page Sender*. The application server includes two modules: *Session Manager* and *Scripts Executor*.
- The third tier is *Database Management System* that stores and retrieves the products information, user profiles etc.

Figure 11 also reflects how Web site functional components interact among one another when processing a client request. The process starts when the *Task Dispatcher* receives an HTTP request. The *Task Dispatcher* uses the URL string to decide the type of the requested Web page. If the requested Web page is static, the *Task Dispatcher* passes the request to the *Web-page sender*. The *Web–page Sender* finds the requested Web page and sends it back to the client side. Otherwise, if the requested Web page is dynamic, the *Task Dispatcher* passes the request to the *Session Manager* in the application server. The *Session Manager* prepares the dynamic Web page by monitoring the script execution. When a <*CacheInfo*> tag is encountered, the *Session Manager* tells the *Web Information Blocks* cache manger to check if this block has been cached. If the block is available in cache, then the *Session Manager* retrieves the cached block, assigns the personalization-related variables and outputs the HTML result. When a <*CacheHtmlSegment*> tag is encountered, the *Session Manager* tells the *Web Html Segments* cache manger to check if

this block has been cached. If the block is available in cache, the *Session Manager* retrieves the cached block and outputs the HTML result. For both cases, if the block cannot be found in cache, the *Session Manager* continues the script execution and sends the execution result to the corresponding *Cache Manager*, which caches the result. When all scripts have been executed, the Session Manager sends the dynamically-generated Web page to the *Web-page Sender* in the Web server. The *Web-page Sender* returns the requested page to the client side.

3.3 Solution Evaluation

The average response time is one of the most important performance metrics for a Web site [19]. In the proposed solution, data reusability can be increased greatly because of flexible Web programming and the *dual caching scheme*. The *Average response time* is expected to be improved in comparison to a system that uses a page-level caching and a system that uses an HTML segment-level caching.

3.3.1 Evaluation Goal

The purpose of evaluation is to investigate the impact of the *dual caching scheme* on the system performance in terms of the average response time by comparison with the system with the page-level caching and the system with the HTML segment-level caching.

3.3.2 Evaluation Method

The evaluation method is to set up a hybrid (experiment+simulation) environment to experiment with realistic system components such as Web server, application server and database server as well as simulate the cache scheme.

To simplify the realistic Web site system complexity, an M/M/1 Queuing Model can be used to formulate system performance. This simplification is based on an assumption: a whole Web site can be viewed as a black box that receives the customers' requests and processes them one at a time and Poisson arrivals and Poisson service times can be applied to this Web site system. In the M/M/1 queuing model, the average response time, T, can be denoted by the following equation.

$$T=T_s/(1-AT_s)$$

Where T_s is the average service time and A is the average arrival rate. In this research, there are four steps designed to calculate the average service time.

Step 1: Model the functional components of a Web site using a Client/Server Interaction Diagram (CSID) [6].

A CSID represents all possible Client/Server interactions among all functional servers of a Web site such as Web server, application server, database server and caching scheme. A CSID has nodes (squares and circles) and directed arcs (arrows) connecting these nodes. Nodes represent requests from clients and the functional servers. Arrows indicate the possible paths to process a dynamic Web page.

The CSIDs of the Web site systems under this research are depicted in Figure 13, Figure 14 and Figure 15. Since all Web pages are dynamic, the diagram doesn't include the case: C (client)->WS (Web server)->C (client).

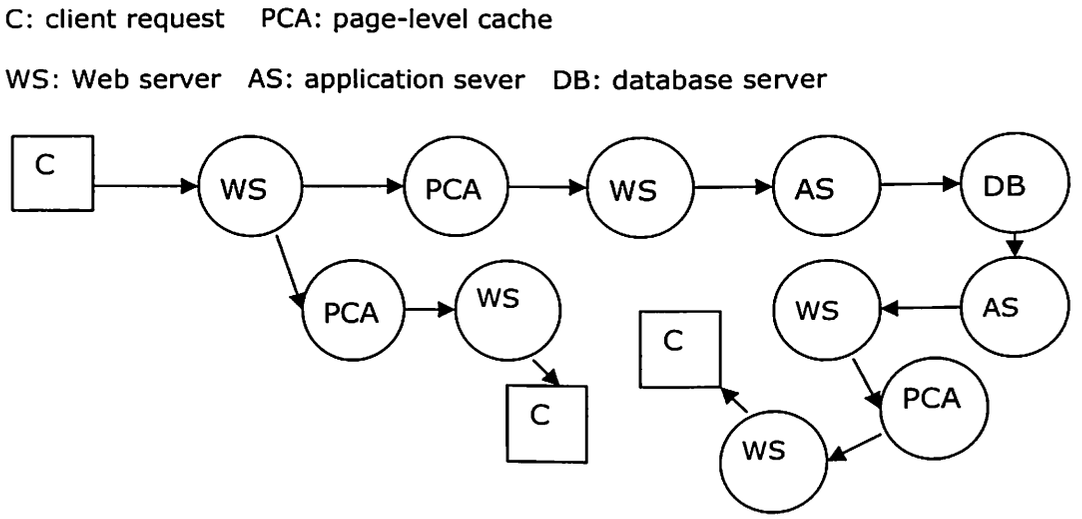


Figure 14. Web site CSID with a page-level cache

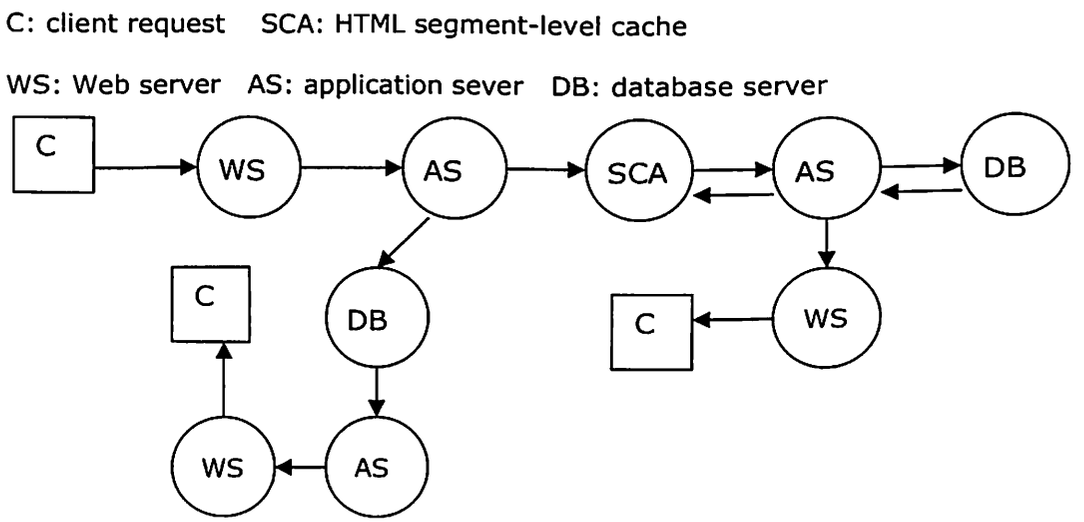


Figure 12. Web site CSID with an HTML segment-level cache

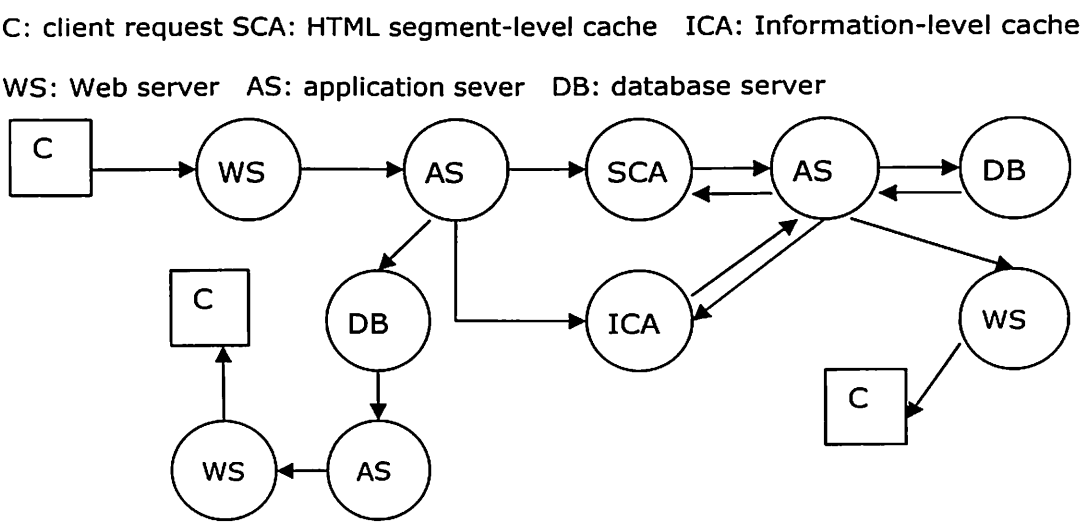


Figure 16. Web site CSID with a dual caching scheme

Step 2: Establish a realistic Web site system with the configuration shown in Table 1, and test a *Web Content Block* of a dynamic Web page *flowers4u.php* generated by a *Web Content Function* in this system. The information of the *Web Content Block* generation is shown in Table 2.

Platform	Operating System	Windows 2000
	CPU	667 MHz
	Size of Hard Drive	6.82 GB
	Size of RAM	256 MB
Servers	Web Server	Apache 2.0.39
	Application Server	PHP 4.2.2
	Database Server	MySQL

Table 1. Web site system configuration

Servers	Time Category	Duration of Time (seconds)
Database Server	Time taken for Database Connection	0.0020190
	Time taken for Database Query	0.0281260
Application Server	Time taken for the Data Formatting	0.0004471
	Time taken for the Formatted Data Personalization	0.0004710

Table 2. Server processing speed during Web content block generation

Step 3: Translate the three CSIDs under study into a system simulation that uses the times in Table 2. The simulation code is designed with the help of the Unified Modeling Language (UML) [20] and is coded in the C++ language [21]. The class diagram of the simulation code is described in Appendix F. The simulation code is in Appendix D.

The realistic times can be obtained for the Web server, the application server, and the database server to process a *Web Content block*. Since the cache architectures usually are implemented by software in realistic Web site systems, the simulation code of caches for different schemes in this research is designed and implemented in C++ language. The mainly execution times spent on the cache are shown in Table 3.

Time Category	Explanations
Time taken for Cache Object Placement	Place cache object from Web server memory or application server memory to cache memory
Time taken for Cache Object searching	Locate a requested cache object or find a location for a new cache object (replacement)
Time taken for Cache Object Output	Output a requested Cache Object to Web server memory or application server memory

Table 3. Time used on a cache

In the simulation code, the workload is represented by a workload class and generated at run time. The workload consists of a set of predefined Web site characteristics (Table 4) and a group of client requests carrying some information (Table 5).

Web site characteristics	Value
Average Web page size	200 KB
The average number of Web blocks in a Web page	4
Average Web Content Block size	50KB
Average Web Info. Block size	48KB
The number of total Web pages in a Web site	500
The number of total Web blocks in a Web site	300
The number of total users in a Web site	1000

Table 4. The predefined Web site characteristics in Workload

Information	Explanations	Representation in Simulation
pageID	The page name. e.g. Flower4u.php	Represented by an integer ranging from 1 to the number of total Web pages.
UserID	When a new user registers in the Web site, a unique userID will be assigned to him.	Represented by an integer ranging from 1 to the number of total users.
themeID	When a new user registers in the Web site, he is entitled to choose his own web page preferences. For a kind of combination, the Web site will assign a themeID to it. The number of total Web site themes reflects the extent to which the Web site can provide personalization for users.	Represented by an integer ranging from 1 to the number of total Web site themes.
webBlockIDs	A web page holds a set of Web content blocks. A Web content block has a unique name.	Represented by integers ranging from 1 to the number of total Web blocks.

Table 5. Information in a client request

Step 4: Design a set of experiments (Table 6), run the simulation code and output results.

Experiments	Purpose	Constants			Variables		
1	Impact of the personalization degree on the average response time	Average Arrival Rate	Cache Size (MB)	Number of Requests	Number of Themes		
		0.005	250	1000	1	10	20
					30	40	
2	Impact of cache size on the average response time	Average Arrival Rate	Number of Themes	Number of Requests	Cache Size(MB)		
		0.005	25	1000	5	10	15
					20	25	
3	Impact of average arrival rate on the average response time	Cache Size(MB)	Number of Themes	Number of Requests	Average Arrival Rate		
		250	25	1000	0.004	0.006	0.010
					0.014	0.016	

Table 6. Experiment design to study impact of various parameters on average response times

3.3.4 Results and Analysis

(1) Experiment 1:

Study the Web site performance by observing variations of the average response time (T) as the number of themes varies in three solutions when the cache size and the average arrival rate (A) are constants; and study the cache performance by observing variations of the cache byte hit rate as the number of themes varies in three solutions when the cache size and the average arrival rate (A) are constants. The cache byte hit rate can be expressed as the ratio of bytes that users obtain from the cache to the overall bytes that users request from the Web site system. A theme can be explained as a kind of the Web page presentational style that the Web site system can provide. When a new user registers in the Web site, he or she is entitled to choose his or her own Web page presentational

style, which is stored in his or her profile and used for the Web page personalization. The number of Web site themes reflects the extent to which the Web site can provide personalization for users.

Figures 17 and 18 graphically present the simulation results of experiment 1, which are shown in Table 7.

Figure 17 describes the relationship between the average response time and the number of themes under the three cache schemes. As the number of themes increases, the average response time of the Web site with a page-level cache fluctuates, ranging from 190ms to 240ms as well as falling above any other curve. This indicates that the average response time of the Web site with a page-level cache depends on some other parameters of the request sequences in the workload instead of the number of themes. For example, here is a request sequence:

Request1 (user1, page1 (WebBlk1,WebBlk4,WebBlk7,WebBlk10), theme1)->

Request2 (user1, page2 (WebBlk1,WebBlk5,WebBlk6,WebBlk10), theme1) ->

Request3 (user4, page1 (WebBlk1,WebBlk4,WebBlk7,WebBlk10), theme4) ->

Request4 (user5, page3 (WebBlk1,WebBlk6,WebBlk7,WebBlk8), theme5) ->

Request5 (user1, page1 (WebBlk1, WebBlk4, WebBlk7, WebBlk10), theme1)

In this request sequence, request 1 is identical with request 5, which can take advantage of the page-level cache to accelerate its generation since they have same userID and pageID. In this research, the Web site system has 1000 users and 500 Web pages. Random combinations of a userID out of the 1000 and a pageID out of the 500 in a request determine that the average response time of the Web site with a page-level cache fluctuates instead of tendency to steadily increase or decrease. From Figure 17, the

average response time of the Web site with an HTML segment-level cache rises with an increase in the number of themes. It is because the increase in the number of themes can lead to fewer sharable Web Content blocks, which impairs the cache performance, represented by a decreased byte hit rate in Figure 18. The tendency of the average response time of the Web site with a dual cache is similar to that of the Web site with a page-level cache in that both curves show minimal variance. This indicates that the average response time of the Web site with a dual cache is independent of the number of themes.

From experiment 1, a conclusion can be made that the Web site performance with a dual cache is superior to other caching schemes since it is independent of the degree of Web site personalization.

Arrival Rate: 0.005; Cache Size: 250MB; the Number of Requests: 1000

Cache Schemes	Number of themes	Web Site System Performance				Cache Performance			
		Total Time (ms)	T_s (ms)	$1/T_s$	T (ms)	Time (ms)	Hits	Byte Hit (KB)	Byte Hit Rate (%)
Page-Level Cache	1	107554	108	0.009	233	972	214	42800	21.4
	10	106710	107	0.009	228	942	220	44000	22.0
	20	98186	98	0.010	193	1232	285	57000	28.5
	30	106679	107	0.009	228	1182	222	44400	22.2
	40	107794	108	0.009	233	941	212	42400	21.2
HTML segment-level cache	1	11201	11	0.089	12	1200	3705	185250	92.6
	10	67913	68	0.015	103	3876	2111	105550	52.8
	20	70727	71	0.014	109	4215	2038	101900	51.0
	30	91511	91	0.011	168	4998	1448	72400	36.2
	40	98801	99	0.010	195	5949	1261	63050	31.5
Dual cache	1	12860	13	0.078	14	1192	3705	185250	92.6
	10	12722	13	0.079	14	920	3701	177648	88.8
	20	12446	12	0.080	13	912	3709	178032	89.0
	30	12762	13	0.078	14	960	3701	177648	88.9
	40	12755	13	0.078	14	920	3700	177600	88.8

Table 7. Results from Experiment 1

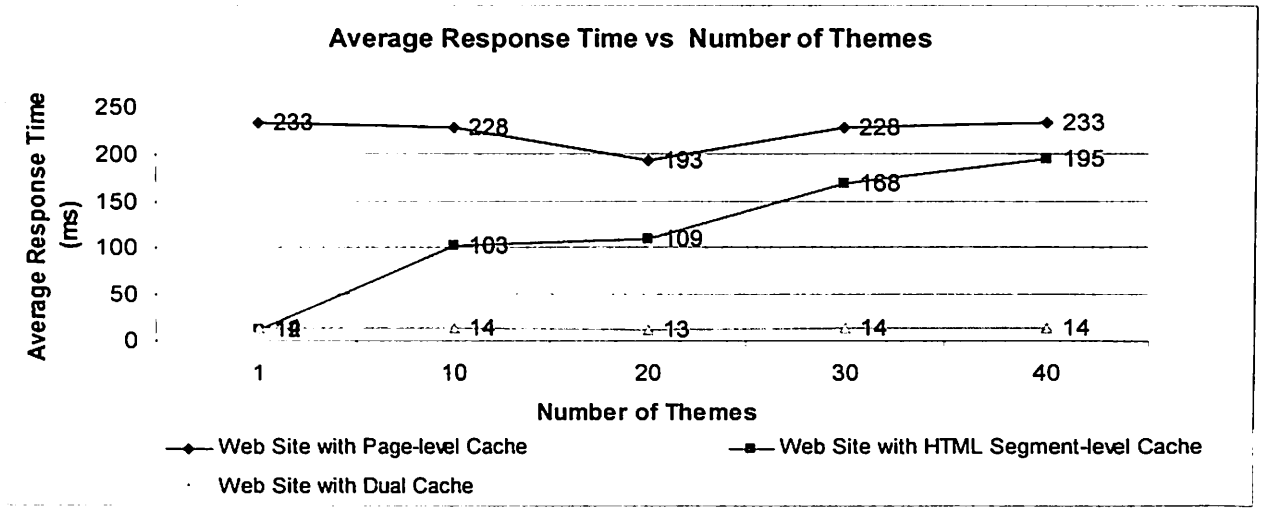


Figure 17. Relationship between Web site system average response time and the number of themes under three cache schemes in Experiment 1

The cache performance contributes to the Web site performance. From Figure 18, the relationship between the cache byte hit rate and the number of themes under the three cache schemes is shown.

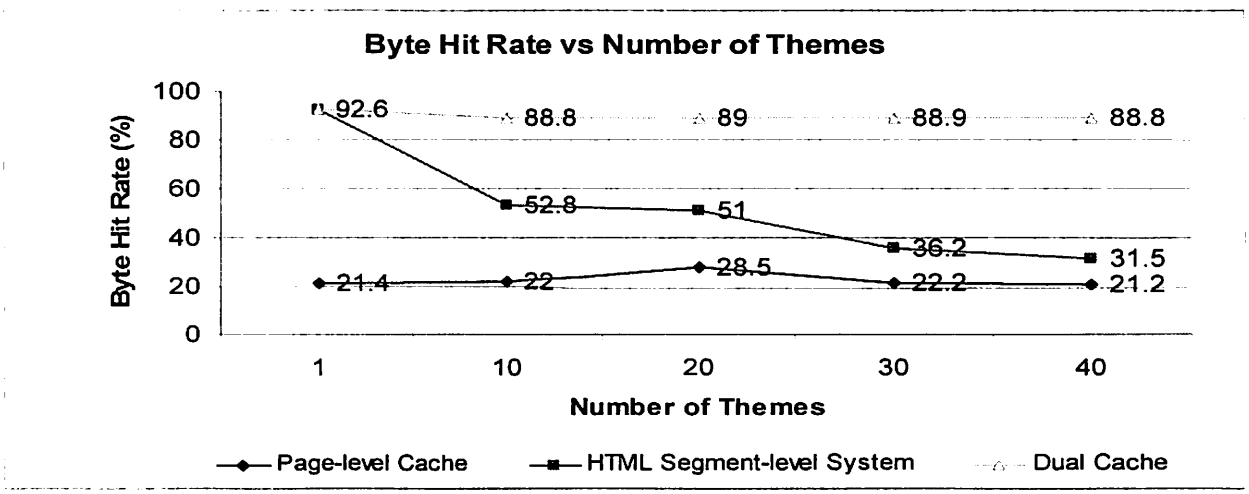


Figure 13. Relationship between byte hit rate and the number of themes under three cache schemes in Experiment 1

(2) Experiment 2:

Study the Web site performance by observing variations of the average response time (T) as the cache size varies in the three solutions when the average arrival rate (A) and the number of themes are constants; and study the cache performance by observing variations of the cache byte hit rate as the cache size varies in the three solutions when the average arrival rate (A) and the number of themes are constants. Figures 19 and 20 graphically present the simulation results of experiment 2, which are shown in Table 8.

Figure 19 describes the relationship between the average response time and the cache size under the three cache schemes. As the cache size increases, the average response time of the Web site with a page-level cache or an HTML segment-level cache keeps decreasing since the cache byte bit rate increases in both cache schemes shown in Figure 20.

As Figures 19 shows, the average response time of the Web site with a dual cache decreases as the cache size increases. Increasing cache size beyond 12 MB provides no significant decrease in average response time. A similar turning point can be observed on the curve of the Web site with a dual cache in Figure 20. This indicates that for this workload 12 MB cache size is enough for a dual cache to gain the best byte hit rate. However, to gain the same Web site and cache performances, the other two schemes need more cache space.

From experiment 2, a conclusion can be made that the Web site with a dual cache can gain better performance with less cache size.

Arrival Rate: 0.005; the Number of Themes: 25; the Number of Requests: 1000

Cache Schemes	Cache Size (MB)	Web Site System Performance				Cache Performance			
		Total Time (ms)	T_s (ms)	$1/T_s$	T (ms)	Time (ms)	Hits	Byte Hit (KB)	Byte Hit Rate (%)
Page-Level Cache	5	136281	136	0.007	427	681	0	0	0.0
	10	135165	135	0.007	417	931	10	2000	0.1
	15	134353	134	0.007	409	651	14	2800	0.1
	25	133508	134	0.007	402	891	22	4400	2.2
	50	96564	97	0.010	187	830	294	58800	29.4
HTML segment level cache	5	131954	131	0.008	388	1981	166	8300	4.2
	10	131359	131	0.008	383	2403	196	9800	4.9
	15	125954	126	0.008	340	3033	374	18700	9.4
	25	122662	123	0.008	317	4046	501	25050	12.5
	50	51241	51	0.020	69	3036	2578	128900	64.45
Dual cache	5	87592	88	0.011	156	963	1464	70272	35.1
	10	40034	40	0.025	50	1172	2892	138816	69.4
	15	12638	13	0.079	13	903	3703	177744	88.9
	25	12756	13	0.078	14	921	3700	177600	88.8
	50	11632	12	0.086	12	700	3727	178896	89.4

Table 8. Results from Experiment 2

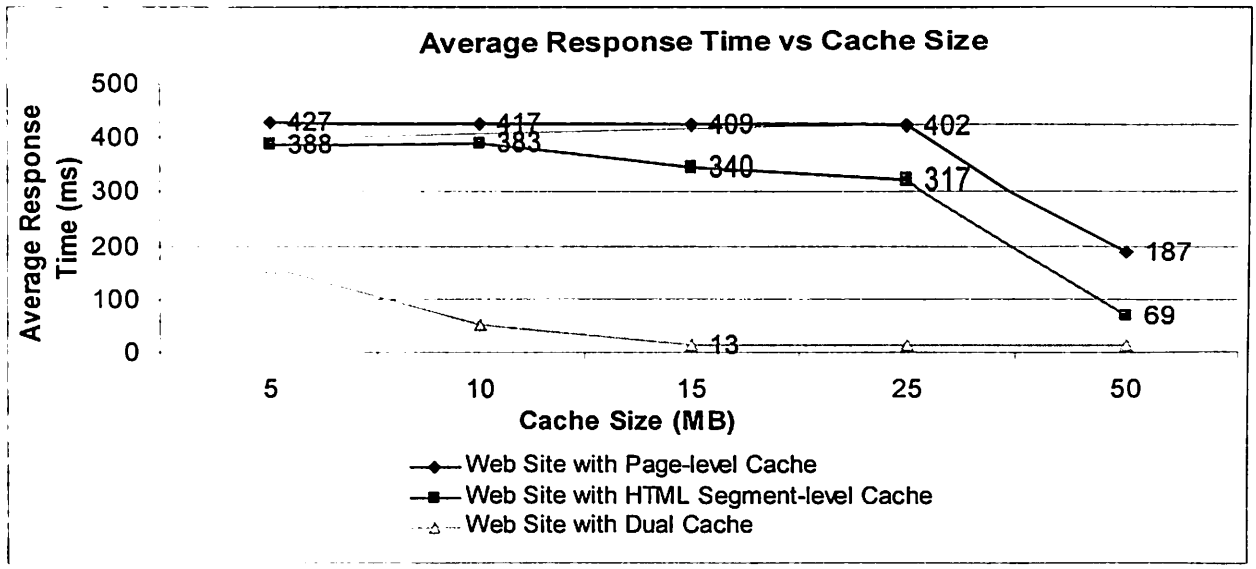


Figure 19. Relationship between Web site system average response time and cache size under three cache schemes in Experiment 2

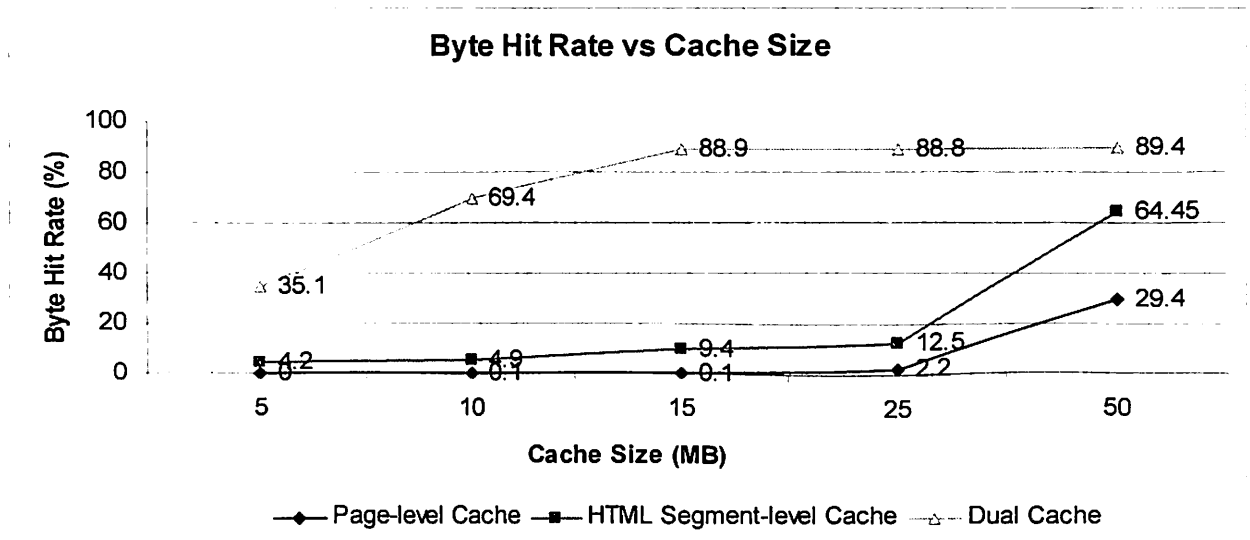


Figure 20. Relationship between byte hit rate and number of themes under three cache schemes in Experiment 2

(3) Experiment 3:

Study the Web site performance by observing variations of the average response time (T) as the average arrival rate (A) varies in the three solutions when the cache size and the number of themes are constants.

Figure 21 presents simulation results of experiment 3 shown in Table 9.

Figure 21 describes the relationship between the average response time and the average arrival rate under the three cache schemes. As the average arrival rate increases, the curves of the average response times of the Web site with a page-level cache and an HTML segment-level cache become undefined at the points of the average arrival rate 0.009 and the average arrival rate 0.012 since the average response times calculated by the formula in M/M/1 queuing model are negatives. The physical meaning of the negative average response time is that the average arrival rate is greater than the average service time, which indicates that the Web site system becomes an unstable system because of the unbounded waiting queue length.

From experiment 3, a conclusion can be made that the Web site with a dual cache can handle more requests in a time period, which can be expressed by the Web site scalability.

Cache Size: 250 MB; the Number of Themes: 25; the Number of Requests: 1000

Cache Schemes	Arrival Rate	Web Site System Performance			
		Total Time (ms)	T_s (ms)	$1/T_s$	T (ms)
Page-Level Cache	0.004	108371	108	0.009	191
	0.006	99206	99	0.010	245
	0.010	104184	104	0.009	<0
	0.014	--	--	--	--
	0.016	--	--	--	--
HTML segment level cache	0.004	90238	90	0.011	141
	0.006	85274	85	0.012	176
	0.010	86036	86	0.012	616
	0.014	85538	86	0.012	<0
	0.016	--	--	--	--
Dual cache	0.004	12715	13	0.08	13
	0.006	12996	13	0.077	14
	0.010	12710	13	0.079	15
	0.012	12767	12	0.078	16
	0.014	12707	13	0.079	17

Table 9. Results from Experiment 3

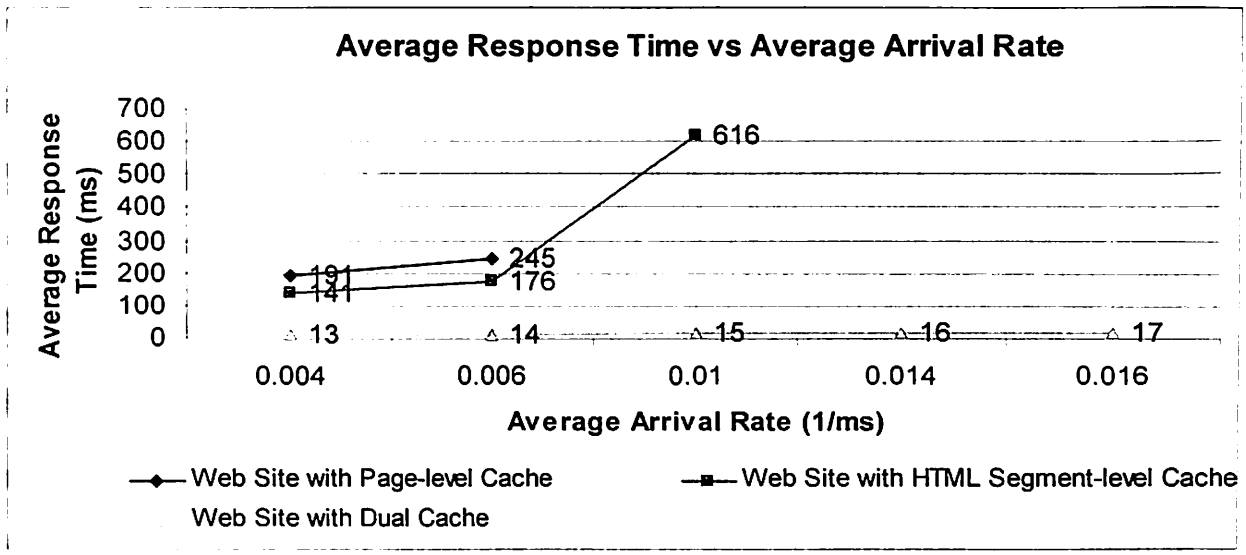


Figure 21. Relationship between Web site system average response time and average arrival rate under three cache schemes in Experiment 3

CHAPTER IV

CONCLUSION AND DISCUSSION

This research presents a new solution to address the problem that arises when Web site system serves highly personalized dynamic Web pages: *Web programming and Web site architecture co-design.* First, this solution employs a *Web page template* to lay out a whole Web page. A *Web Content Function* can be used to define each part in the *Web template.* A *Web Content Function* that can generate a *Web Content Block* is programmed as two layers: one is a *Web Information Block Generator*, which serves as an interface to the Web site database; the other is a *Web Information Block Stylizer*, which wraps the information from the *Web Information block Generator* to produce a personalized HTML segment. In the scripting language, two new cache tags are introduced: tags *<CacheInfo> and <CacheHtmLSegment>* respectively are used to trigger processes of caching the *Web Information Block* and the *Web Html Segment*; Second, the system architecture correspondingly has two main caches: cache one is to cache *Web Information Blocks*; cache two is to cache *Web Html Segments.*

The evaluation method presented in this paper is to set up a hybrid environment to experiment with realistic system components such as Web server, application server and database server as well as simulate the cache scheme. The purpose of evaluation is to investigate the impact of the proposed solution on the system performance in terms of the average response time by comparison with other solutions.

A Web site system under the proposed solution has been tested and shows better performance than other compared solutions dealing with dynamic Web content delivery, especially highly-personalized dynamic Web content delivery. Specifically speaking, the Web site system performs well independent of Web site personalization degree, gain better performance with less cache size requirement as well as handle more client requests in a time period.

In this paper, the research work focuses on how to increase the data reusability in cache through choosing proper Web objects for caching when dealing with highly personalized Web page delivery. This is only one essential aspect of caching dynamic contents. In fact, an efficient cache management policy for cached Web objects is also an important aspect. In addition, the evaluation method presented in this paper is not based on a completely realistic Web site system. Hence, several future works can be encouraged to do: to study the characteristics of cached Web objects and apply a better cache management policy to them instead of the simple LFU policy used in this research; to implement this proposed solution on a realistic Web site system and give the more accurate experimental results.

BIBLIOGRAPHY

1. Iyengar, A., J. Challenger, D. Dias, P. Dantzig, "High-Performance Web Design Techniques," *IEEE Internet Computing*, Vol. 4, No. 2, 17 (2000).

2. Mohapatra, A. and H. Chen, "WebGraph: A Framework for Managing and Improving Performance of Dynamic Web Content," *IEEE journal on selected areas in communications*, Vol. 20, NO. 7, 1414 (2002).

3. Iyengar, A. and J. Challenger, "Improving Web Server Performance by Caching Dynamic Data," *Proc.Usenix Symp.Internet Technologies and Systems*, Usenix Assoc., Berkeley, Calif. 1997.

4. Datta, A., K. Dutta, H. Thomas, D. VanderMeer and K. Ramamritham, "Accelerating dynamic Web content generation," *IEEE Internet Computing,* Vol. 6, Issue 5, 27 (2002).

5. K. Selçuk Candan, W. Li, Q. Luo, W. Hsiung, and D. Agrawal, "Enabling dynamic content caching for database-driven Web sites," *ACM SIGMOD Record , Proceedings of the 2001 ACM SIGMOD international conference on Management of data,* Vol. 30, Issue 2, 532 (2001).

6. Menasce, D., and V. Almeida, *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning,* Prentice Hall PTR, 2000.

7. Morrison, M. and J. Morrison, *Database-Driven Web Sites, Second Edition,* Thomson, 2003.

8. Podlipnig, S. and Laszlo B., "A Survey of Web Cache Replacement Strategies," *ACM Computing Surveys*, Vol. 35, No. 4, 374 (2001).

9. Eirinaki, M. and M. Vazirgiannis, "Web Mining for Web Personaliztion," *ACM Transactions on Internet Technology*, Vol. 3, No. 1, 1 (2003).

10. A. Datta, K. Dutta, D. VanderMeer, K. Ramamritham and S.B. Navathe, "An architecture to support scalable online personalization on the Web," *The VLDB Journal — The International Journal on Very Large Data Bases*, Vol. 10, Issue 1, 104 (2001).

11. Wu, K., C. Aggarwal, and P. Yu, "Personalization with Dynamic Profiler," *Advanced Issues of E-Commerce and Web-Based Information Systems, WECWIS 2001, Third International Workshop*, 12 (2001).

12. Slothouber, L., "A Model of Web Server Performance", http://www.geocities.com/webserverperformance/modelpaper.html, June, 1995.

13. Donald, G., C. Harris, *Fundamentals of queueing theory,* New York: Wiley, 1998.

14. Sklenar, J., "Elements of Queuing Systems," http://staff.um.edu.mt/jskl1/simweb/intro.htm.

15. Vastola, K., "The Poisson Arrival Model," http://networks.ecse.rpi.edu/~vastola/pslinks/perf/node30.html.

16. "M/M/1 Queueing System," http://www.eventhelix.com/RealtimeMantra/CongestionControl/m_m_1_queue.htm.

17. Castagnetto, J., H. Rawat, S. Schumann, C. Scollo, D. Veliath, *Professional PHP Programming,* Wrox Press, 2000.

18. Welling, L., L. Thomson, *PHP and MySQL Web Development*, SAMS, 2001.

19. Li, W. , W. Hsiung, O. Po, K. Candan and D. Agrawal, "Evaluations of architectural designs and implementation for database-driven web sites," *Data & Knowledge Engineering, Vol.* 43, 151 (2002).

20. Lee, R., W. Tepfenhart, *UML and C++: a practical guide to object-oriented development,* Prentice Hall, 2001.

21. Anderson, A., W. Heinze, *C++ programming and fundamental concepts*, Prentice-Hall, 1992.

APPENDIX A

```
/*************************************/

Web Content Functions for the Navigation bar

/*************************************/


function naviBarInfo($naviTable,$naviTd,$mainFont,$endFontToken)
  {

     $aSQL= "select * from naviBar";
     $aQResult=DBSearch($aSQL);
     if($aQResult)
          {
               <cacheInfo>      // cacheInfo tag begin or cacheHtmlSegment tag

                 echo "<Table ";
                 echo $naviTable;
                 echo ">";
                 while($row=mysql_fetch_array($aQResult))
                 {
                         echo "<tr>";
                         echo "<td ";
                         echo $naviTd;
                         echo ">";
                         echo "<div align=\"center\">";
                         echo $mainFont;
                         echo $row["buttonName"];
                         echo $endFontToken;
                         echo "</div>";
                         echo "</td>";
                        echo "</tr>";
                 }
               mysql_free_result($aQResult);
        echo "</Table>";

               </cacheInfo>     // cacheInfo tag end or cacheHtmlSegment tag


          }
```

```
        else
            {
                echo " query error in DB Table naviBar";
            }

    }


function naviBarStyle($themeID)
{
    $aSQL= "select * from theme where themeID=$themeID";
    $aQResult=DBSearch($aSQL);
    if($aQResult)
        {
            while($row=mysql_fetch_array($aQResult))
            {
                $naviTable=$row["naviTable"];
                $naviTd=$row["naviTd"];
                $mainFont=$row["mainFont"];
                $endFontToken=$row["thirdFont"];
            }
            mysql_free_result($aQResult);


            naviBarInfo($naviTable,$naviTd,$mainFont,$endFontToken);
        }

    else
        {
            echo " query error in BD Table theme";
        }
}

function naviBar($themeID)
{

        naviBarStyle($themeID);
}
```

APPENDIX B

```
/*************************************/
PHP Scripts with the tag <cacheInfo>:

/*************************************/


  function naviBarInfo($naviTable,$naviTd,$mainFont,$endFontToken)
  {

     $aSQL= "select * from naviBar";
     $aQResult=DBSearch($aSQL);
     if($aQResult)
          {
               <cacheInfo>      // cacheInfo tag begin or cacheHtmlSegment tag

                 echo "<Table ";
                 echo $naviTable;
                 echo ">";
                 while($row=mysql_fetch_array($aQResult))
                 {
                          echo "<tr>";
                          echo "<td ";
                          echo $naviTd;
                          echo ">";
                          echo "<div align=\"center\">";
                          echo $mainFont;
                          echo $row["buttonName"];
                          echo $endFontToken;
                          echo "</div>";
                          echo "</td>";
                         echo "</tr>";
                 }
               mysql_free_result($aQResult);
       echo "</Table>";

               </cacheInfo>      // cacheInfo tag end or cacheHtmlSegment tag


          }
      else
          {
```

```
                echo " query error in DB Table naviBar";
            }

    }
```

/**************************************/

Output of the <CacheInfo> tagged PHP scripts

/**************************************/

```
        <Table $naviTable >
       <tr><td $naviTd >
            <div align="center">$mainFont  Anniversary $EndFontToken</div>
             </td>
         </tr>
        <tr><td $naviTd >
            <div align="center">$mainFont  Baby $EndFontToken</div>
             </td>
         </tr>
       <tr><td $naviTd >
            <div align="center">$mainFont  Birthday $EndFontToken</div>
             </td>
         </tr>
       <tr><td $naviTd >
            <div align="center">$mainFont  Business Gift $EndFontToken</div>
             </td>
         </tr>
       <tr><td $naviTd >
            <div align="center">$mainFont  Congtatulations $EndFontToken</div>
             </td>
         </tr>
       <tr><td $naviTd >
            <div align="center">$mainFont  For Him $EndFontToken</div>
             </td>
         </tr>
       <tr><td $naviTd >
            <div align="center">$mainFont  Get Well $EndFontToken</div>
             </td>
         </tr>
       <tr><td $naviTd >
            <div align="center">$mainFont  House Warming $EndFontToken</div>
             </td>
         </tr>
```

```
<tr><td $naviTd >
    <div align="center">$mainFont  Love $EndFontToken</div>
     </td>
  </tr>
<tr><td $naviTd >
    <div align="center">$mainFont  Thank you $EndFontToken</div>
     </td>
  </tr>
<tr><td $naviTd >
    <div align="center">$mainFont  Thanksgiving $EndFontToken</div>
     </td>
  </tr>
<tr><td $naviTd >
    <div align="center">$mainFont  Wedding $EndFontToken</div>
     </td>
  </tr>
 </Table>
```

APPENDIX C

```
/*************************************/
```

PHP Scripts with the tag <CacheHtmlSegment>:

```
/*************************************/

  function naviBarInfo($naviTable,$naviTd,$mainFont,$endFontToken)
  {

     $aSQL= "select * from naviBar";
     $aQResult=DBSearch($aSQL);
     if($aQResult)
          {
               <cacheHtmlSegment>

                 echo "<Table ";
                 echo $naviTable;
                 echo ">";
                 while($row=mysql_fetch_array($aQResult))
                 {
                          echo "<tr>";
                          echo "<td ";
                          echo $naviTd;
                          echo ">";
                          echo "<div align=\"center\">";
                          echo $mainFont;
                          echo $row["buttonName"];
                          echo $endFontToken;
                          echo "</div>";
                          echo "</td>";
                         echo "</tr>";
                 }
               mysql_free_result($aQResult);
       echo "</Table>";

               </cacheHtmlSegment>

          }
     else
          {
               echo " query error in DB Table naviBar";
```

```
        }
    }
```

```
/*********************************************/
```

Output of the <CacheHtmlSegment> tagged PHP scripts:

```
/*********************************************/

<Table  width="77%" border="0" cellspacing="0" cellpadding="0" align="center">
      <tr><td background="theme/naviBarBg/purpleBT.jpg" height="29">
          <div align="center">
          <b><font color="#FF00FF">Anniversary </font></b>
          </div>
           </td>
        </tr>
      <tr><td background="theme/naviBarBg/purpleBT.jpg" height="29">
          <div align="center">
          <b><font color="#FF00FF">Baby</font></b>
          </div>
           </td>
        </tr>
      <tr><td background="theme/naviBarBg/purpleBT.jpg" height="29">
          <div align="center">
          <b><font color="#FF00FF">Birthday</font></b>
          </div>
           </td>
        </tr>
      <tr><td background="theme/naviBarBg/purpleBT.jpg" height="29">
          <div align="center">
          <b><font color="#FF00FF">Business Gift</font></b>
          </div>
           </td>
        </tr>
      <tr><td background="theme/naviBarBg/purpleBT.jpg" height="29">
          <div align="center">
          <b><font color="#FF00FF">Congratulations</font></b>
          </div>
           </td>
        </tr>
      <tr><td background="theme/naviBarBg/purpleBT.jpg" height="29">
```

```
      <div align="center">
      <b><font color="#FF00FF">For Him</font></b>
      </div>
       </td>
  </tr>
<tr><td background="theme/naviBarBg/purpleBT.jpg" height="29">
      <div align="center">
      <b><font color="#FF00FF">Get Well</font></b>
      </div>
       </td>
  </tr>
<tr><td background="theme/naviBarBg/purpleBT.jpg" height="29">
      <div align="center">
      <b><font color="#FF00FF">House Warming</font></b>
      </div>
       </td>
  </tr>
<tr><td background="theme/naviBarBg/purpleBT.jpg" height="29">
      <div align="center">
      <b><font color="#FF00FF">Love</font></b>
      </div>
       </td>
  </tr>
<tr><td background="theme/naviBarBg/purpleBT.jpg" height="29">
      <div align="center">
      <b><font color="#FF00FF">Thank you </font></b>
      </div>
       </td>
  </tr>
<tr><td background="theme/naviBarBg/purpleBT.jpg" height="29">
      <div align="center">
      <b><font color="#FF00FF">Thanksgiving</font></b>
      </div>
       </td>
  </tr>
<tr><td background="theme/naviBarBg/purpleBT.jpg" height="29">
      <div align="center">
      <b><font color="#FF00FF">Wedding</font></b>
      </div>
       </td>
  </tr>

 </Table>
```

APPENDIX D

SIMULATION CODE FOR WEB SITE SYSTEM UNDER THREE CACHE SCHEMES

```
//=============================================
// main.cpp
//=============================================
#pragma once
#include <iostream>
using namespace std;

#include "WorkLoad.h"
#include "WebServer.h"
#include "PageCache.h"
#include "HTMLSegmentCache.h"
#include "DualCache.h"
#include "DBServer.h"
#include "AppServer.h"
#include <time.h>

void main()
{

        //***********WORKLOAD PARAMETERS**********
        int avrgInfoBlkSize=48,avrgContBlkSize=50,avrgPageSize=200;
        int avrgBlkNum=4;
        int numTheme,numUser=1000,numPage=500,numWebBlk=300;
    int numRefer=1000;
        int workLoadType=2;


        //***********SYSTEM PARAMTERS*************
    double DBTimeOneBlk=33.0;        //DBServer (clocks)
       double
       infoBlkTime=0.45,contBlkTime=0.45,getScriptFile=0;//AppServer
        double disPatchTime=0,sendFileTime=0; //(WebServer)
        int cacheSize=250000;  // cacheSize(KB)
        int infoCacheSize,contCacheSize;

        //************M/M/1 model*****************
    float ArrivalRate=0.005;
```

```
        //***********EVALUATION VARIABLES**********
        double
    totalTicksCSID1=0.0,totalTicksCSID2=0.0,totalTicksCSID3=0.0;
        float
    ByteHit,ByteHitRate,ArgServiceTime,ServiceRate,ArgResponseTime;

        //***********TEMPORARY VARIABLES***********

        int requestID,userID,themeID,webBlkID,pageID;
        int pageSize,inforBlkSize,contBlkSize,sizeTemp;
        int loop;
        int tempInt,i;
        int CSID1=1,CSID2=2,CSID3=3;

  //***********WORKLOAD TYPE OPTIONS***********
        switch(workLoadType)
        {
        case 1: numTheme=1;
                    contCacheSize=cacheSize;
                        infoCacheSize=0;
                break;

        case 2: numTheme=20;
              infoCacheSize=cacheSize;
                        contCacheSize=0;
           break;
        }

  //***********DECLARATION**********************
  CWorkLoad
 WL(avrgInfoBlkSize,avrgContBlkSize,numTheme,numUser,numWebBlk,numPage);

  CDBServer DBServer(DBTimeOneBlk);
  CAppServer AppServer(infoBlkTime,contBlkTime,getScriptFile);

  CWebServer WebServer(disPatchTime,sendFileTime);

  CPageCache PageCache(cacheSize);
  CHTMLSegmentCache HTMLSegmentCache(cacheSize);
  CDualCache DualCache(contCacheSize,infoCacheSize);

 //***********WORKLOAD INITIALIZATON***************
  WL.WLInitiation();

  //**********WORKLOAD PROCESS*********************
```

```
loop=numRefer;
while(loop)
{

 requestID=WL.getARequest(loop);

 //---------WEB SITE WITH PAGE CACHE--------

        userID=WL.getUserID(requestID);
        pageID=WL.getPageID(requestID);
        pageSize=WL.getPageSize(requestID);
        totalTicksCSID1+=WebServer.getDispatchTime();

                if(PageCache.cacheTimeCalculation(2,pageID,userID,pageSize)==2)
    //retrieve && miss
        {
        totalTicksCSID1+=PageCache.getCacheTime();
        totalTicksCSID1+=AppServer.getDiskAccessTime();
        totalTicksCSID1+=avrgBlkNum*DBServer.getDBTime();

totalTicksCSID1+=avrgBlkNum*(AppServer.getInfoBlockTime()+AppServer.getConten
tBlockTime());

tempInt=PageCache.cacheTimeCalculation(1,pageID,userID,pageSize);
 // placement
totalTicksCSID1+=PageCache.getCacheTime();
}
else  // retrieve && hit
{
totalTicksCSID1+=PageCache.getCacheTime();
}
totalTicksCSID1+=WebServer.getSendTime();


//---------WEB SITE WITH HTML SEGMENT CACHE--------

for(i=1;i<=4;i++)
{
webBlkID=WL.getWebBlkID(requestID,i);
themeID=WL.getThemeID(requestID);
contBlkSize=WL.getContBlkSize(requestID);
totalTicksCSID2+=WebServer.getDispatchTime();

if(HTMLSegmentCache.cacheTimeCalculation(2,webBlkID,themeID,contBlkSize)==2)
//retrieve && miss
```

```
{
totalTicksCSID2+=HTMLSegmentCache.getCacheTime();
totalTicksCSID2+=AppServer.getDiskAccessTime();
totalTicksCSID2+=DBServer.getDBTime();
totalTicksCSID2+=AppServer.getInfoBlockTime()+AppServer.getContentBlockTime();
tempInt=HTMLSegmentCache.cacheTimeCalculation(1,webBlkID,themeID,contBlkSize
); // placement
totalTicksCSID2+=HTMLSegmentCache.getCacheTime();
}
else  // retrieve && hit
{
totalTicksCSID2+=HTMLSegmentCache.getCacheTime();
}

}

totalTicksCSID2+=WebServer.getSendTime();

//---------WEB SITE WITH DUAL CACHE--------

for(i=1;i<=4;i++)
{
webBlkID=WL.getWebBlkID(requestID,i);
themeID=WL.getThemeID(requestID);
switch(workLoadType)
{
case 1:
contBlkSize=WL.getContBlkSize(requestID);
sizeTemp=contBlkSize;
break;
case 2:
inforBlkSize=WL.getInfoBlkSize(requestID);
sizeTemp=inforBlkSize;
break;
}

totalTicksCSID3+=WebServer.getDispatchTime();
if(DualCache.cacheTimeCalculation(workLoadType,2,webBlkID,themeID,sizeTemp)==
2) //retrieve && miss
{
totalTicksCSID3+=DualCache.getCacheTime();
totalTicksCSID3+=AppServer.getDiskAccessTime();
totalTicksCSID3+=DBServer.getDBTime();
totalTicksCSID3+=AppServer.getInfoBlockTime()+AppServer.getContentBlockTime();
tempInt=DualCache.cacheTimeCalculation(workLoadType,1,webBlkID,themeID,sizeTe
mp); // placement
```

```
totalTicksCSID3+=DualCache.getCacheTime();
}
else  // retrieve && hit
{
totalTicksCSID3+=DualCache.getCacheTime();
totalTicksCSID3+=AppServer.getContentBlockTime();
}
}
totalTicksCSID3+=WebServer.getSendTime();
WL.updataWLTableEntry(loop);
loop--;

}



//**********EVALUATION OUTPUT**************

cout<<"****EVALUATION*****"<<endl;
cout<<"----------Page Cache----------"<<endl;
cout<<"totalTicksCSID1="<<totalTicksCSID1<<endl;
cout<<"total m seconds="<<totalTicksCSID1<<"ms"<<endl;
ArgServiceTime=totalTicksCSID1/numRefer;
ServiceRate=numRefer/totalTicksCSID1;
cout<<"Average Service Time="<<ArgServiceTime<<"ms"<<endl;
cout<<"Average Service Rate="<<ServiceRate<<endl;
ArgResponseTime=ArgServiceTime/(1-ArrivalRate*ArgServiceTime);
cout<<"Average Response Time="<<ArgResponseTime<<"ms"<<endl<<endl;
cout<<"Time Spent on Cache="<<PageCache.getOverAllCacheTime()<<endl;
cout<<"hitCount="<<PageCache.getHitCount()<<endl;
ByteHit=PageCache.getHitCount()*avrgPageSize;
ByteHitRate=ByteHit/(numRefer*avrgPageSize);
cout<<"ByteHit="<<ByteHit<<"KB"<<endl;
cout<<"ByteHitRate="<<ByteHitRate<<endl<<endl;




cout<<"----------HTML Segment Cache----------"<<endl;
cout<<"totalTicksCSID2="<<totalTicksCSID2<<endl;
cout<<"total m seconds="<<totalTicksCSID2<<"ms"<<endl;
ArgServiceTime=totalTicksCSID2/numRefer;
ServiceRate=numRefer/totalTicksCSID2;
cout<<"Average Service Time="<<ArgServiceTime<<"ms"<<endl;
cout<<"Average Service Rate="<<ServiceRate<<endl;
ArgResponseTime=ArgServiceTime/(1-ArrivalRate*ArgServiceTime);
```

```
cout<<"Average Response Time="<<ArgResponseTime<<"ms"<<endl<<endl;
cout<<"Time Spent on Cache="<<HTMLSegmentCache.getOverAllCacheTime()<<endl;
cout<<"hitCount="<<HTMLSegmentCache.getHitCount()<<endl;
ByteHit=HTMLSegmentCache.getHitCount()*avrgContBlkSize;
ByteHitRate=ByteHit/(numRefer*avrgPageSize);
cout<<"ByteHit="<<ByteHit<<"KB"<<endl;
cout<<"ByteHitRate="<<ByteHitRate<<endl<<endl;


cout<<"----------Dual Cache----------"<<endl;
cout<<"totalTicksCSID3="<<totalTicksCSID3<<endl;
cout<<"total m seconds="<<totalTicksCSID3<<"ms"<<endl;
ArgServiceTime=totalTicksCSID3/numRefer;
ServiceRate=numRefer/totalTicksCSID3;
cout<<"Average Service Time="<<ArgServiceTime<<"ms"<<endl;
cout<<"Average Service Rate="<<ServiceRate<<endl;
ArgResponseTime=ArgServiceTime/(1-ArrivalRate*ArgServiceTime);
cout<<"Average Response Time="<<ArgResponseTime<<"ms"<<endl<<endl;
cout<<"Time Spent on Cache="<<DualCache.getOverAllCacheTime()<<endl;
cout<<"hitCount="<<DualCache.getHitCount()<<endl;
if(workLoadType==1)
ByteHit=DualCache.getHitCount()*avrgContBlkSize;
else
ByteHit=DualCache.getHitCount()*avrgInfoBlkSize;
ByteHitRate=ByteHit/(numRefer*avrgPageSize);
cout<<"ByteHit="<<ByteHit<<"KB"<<endl;
cout<<"ByteHitRate="<<ByteHitRate<<endl<<endl;

}

//=============================================
// WebServer.cpp
//=============================================

#include "WebServer.h"

CWebServer::CWebServer()
{}
CWebServer::CWebServer(double dTime,double sTime)
{
 dispatchTime=dTime;
 sendTime=sTime;
}

CWebServer::~CWebServer()
{}
```

```
double CWebServer::getDispatchTime()
{
  return dispatchTime;
}

double CWebServer::getSendTime()
{
        return sendTime;
}

//=============================================
// AppServer.cpp
//=============================================

#include "AppServer.h"

CAppServer::CAppServer()
{}
CAppServer::CAppServer(double infoTime,double contTime,double diskTime)
{
        webInfoBlockTime=infoTime;
        webContentBlockTime=contTime;
        diskAccessTime=diskTime;
}

CAppServer::~CAppServer()
{}

double CAppServer::getInfoBlockTime()
{
   return webInfoBlockTime;
}

double CAppServer::getContentBlockTime()
{
   return webContentBlockTime;
}

double CAppServer::getDiskAccessTime()
{
   return diskAccessTime;
```

```
}

//=============================================
// DBServer.cpp
//=============================================
#include "DBServer.h"

CDBServer::CDBServer()
{

}
CDBServer::CDBServer(double time)
{
 DBTime=time;
}

CDBServer::~CDBServer()
{

}

double CDBServer::getDBTime()
{
  return DBTime;
}

//=============================================
// PageCache.cpp
//=============================================
#include <iostream>
using namespace std;
#include "PageCache.h"
#include <time.h>
#include <stdlib.h>

CPageCache::CPageCache()
{

}
CPageCache::CPageCache(int cSize)
{
        hitCount=0;
        cacheSize=cSize;
        cacheSizeUsed=0;
        cacheTime=0;
        overAllCacheTime=0;
```

```
}

CPageCache::~CPageCache()
{

}

int CPageCache::cacheTimeCalculation(int flag,int pageID,int userID,int size)
{

 clock_t ticks1, ticks2,ticks;
 int minLRU;
 int i;
 bool found=false;
 PCEPointer PCEPtr;
 list<PCEPointer>::iterator tempIter;


 // get starting time
   cacheTime=0;
  ticks1=clock();

 switch(flag)
 {

 case 1: // placement ( may be replace)

            PCEPtr=new PCacheElement;
            PCEPtr->numLRU=1;
       PCEPtr->objectID=pageID;
            PCEPtr->userID=userID;
            PCEPtr->objectSize=size;  //page size average=200K

          if(size<=(cacheSize-cacheSizeUsed))
          {
                  cacheList.push_back(PCEPtr);
                  cacheSizeUsed=cacheSizeUsed+PCEPtr->objectSize;

          }
          else
          {

                   cacheIter=cacheList.begin();
          minLRU=(*cacheIter)->numLRU;
                  tempIter=cacheIter;
                  while(cacheIter!=cacheList.end())
```

```
                {
        if((*cacheIter)->numLRU<minLRU)

                        {
                                minLRU=(*cacheIter)->numLRU;
                                tempIter=cacheIter;

                        }

                                ++cacheIter;

                }

                cacheSizeUsed=cacheSizeUsed-(*tempIter)->objectSize;
                cacheList.erase(tempIter);
            cacheList.push_back(PCEPtr);
                cacheSizeUsed=cacheSizeUsed+PCEPtr->objectSize;
              }
   // placement time

        for(i=1;i<=200000;i++)
        {
        }
       ticks2=clock();
    ticks=ticks2-ticks1;
    cacheTime=ticks;
       overAllCacheTime=overAllCacheTime+cacheTime;
       return 0;

       break;

case 2: // retrieve


        if(cacheList.size()>0)
        {
                cacheIter=cacheList.begin();

                while(!found && cacheIter!=cacheList.end())
                {
       if((*cacheIter)->objectID==pageID && (*cacheIter)-
         >userID==userID  )
                        {
                                found=true;
                                tempIter=cacheIter;
```

```
                    }
                    else

                            ++cacheIter;
                }

                if(found)
                    {
                      (*tempIter)->numLRU++;
                            for(i=1;i<=200000;i++)
                            {
                            }
                            ticks2=clock();
                ticks=ticks2-ticks1;
                cacheTime=ticks;
                            overAllCacheTime=overAllCacheTime+cacheTime;
                            hitCount++;
                            return 1;
                    }//hit
                    else
                    {
                            ticks2=clock();
                ticks=ticks2-ticks1;
                cacheTime=ticks;
                            overAllCacheTime=overAllCacheTime+cacheTime;

                            return 2;
                    }// not found miss

        }
        else
        {
    ticks2=clock();
    ticks=ticks2-ticks1;
    cacheTime=ticks;
        overAllCacheTime=overAllCacheTime+cacheTime;
    return 2;
        }// empty miss

        break;
 }


}

long CPageCache::getCacheTime()
```

```
{
  return cacheTime;
}


void CPageCache::print(ostream& outStream)
{

 cacheIter=cacheList.begin();
 outStream<<endl<<"Cache Snapshot:"<<endl;
 while(cacheIter!=cacheList.end())
 {
  outStream<<"pageID="<<(*cacheIter)->objectID <<"  ";
  ++cacheIter;
 }
  outStream<<endl<<"hitCount="<<hitCount;
  outStream<<endl<<endl;
}

int CPageCache::getHitCount()
{
  return hitCount;
}

long CPageCache::getOverAllCacheTime()
{
 return overAllCacheTime;
}

//=============================================
// HTMLSegmentCache.cpp
//=============================================
#include <iostream>
using namespace std;
#include "HTMLSegmentCache.h"
#include <time.h>
#include <stdlib.h>

CHTMLSegmentCache::CHTMLSegmentCache()
{
}
CHTMLSegmentCache::CHTMLSegmentCache(int cSize)
{
        cacheSize=cSize;
        cacheSizeUsed=0;
        cacheTime=0;
```

```
        overAllCacheTime=0;
        hitCount=0;
}
CHTMLSegmentCache::~CHTMLSegmentCache()
{

}


int CHTMLSegmentCache::cacheTimeCalculation(int flag,int segmentID,int themeID,int
size)
{

 clock_t ticks1, ticks2,ticks;
 int minLRU;
 int i;
 bool found=false;
 SCEPointer SCEPtr;
 list<SCEPointer>::iterator tempIter;
 cacheTime=0;
 ticks1=clock();

 switch(flag)
 {

 case 1: // placement ( may be replace)

            SCEPtr=new SCacheElement;
            SCEPtr->numLRU=1;
        SCEPtr->objectID=segmentID;
            SCEPtr->themeID=themeID;
            SCEPtr->objectSize=size; //segment average=40K

          if(size<=(cacheSize-cacheSizeUsed))

          {
                  cacheList.push_back(SCEPtr);
                  cacheSizeUsed=cacheSizeUsed+SCEPtr->objectSize;
          }
          Else
     {
                 cacheIter=cacheList.begin();
        minLRU=(*cacheIter)->numLRU;
            tempIter=cacheIter;
            while(cacheIter!=cacheList.end())
                  {
```

```
        if((*cacheIter)->numLRU<minLRU)

                        {
                                minLRU=(*cacheIter)->numLRU;
                                tempIter=cacheIter;

                        }

                                ++cacheIter;

                }

                    cacheSizeUsed=cacheSizeUsed-(*tempIter)->objectSize;
                  cacheList.erase(tempIter);
                    cacheList.push_back(SCEPtr);
                  cacheSizeUsed=cacheSizeUsed+SCEPtr->objectSize;


                }
        for(i=1;i<=40000;i++)
        {
        }
        ticks2=clock();
    ticks=ticks2-ticks1;
    cacheTime=ticks;
        overAllCacheTime+=cacheTime;

        return 0;
        break;

case 2: // retrieve


        if(cacheList.size()>0)
        {
                cacheIter=cacheList.begin();

                while(!found && cacheIter!=cacheList.end())
                {
        if((*cacheIter)->objectID==segmentID && (*cacheIter)-
          >themeID==themeID )

                        {
                                found=true;
                                tempIter=cacheIter;
```

```
                    }
                    else

                            ++cacheIter;
                }

                if(found)
                    {
                      (*tempIter)->numLRU++;
                        for(i=1;i<=40000;i++)
                        {
                        }
                        ticks2=clock();
                        ticks=ticks2-ticks1;
                        cacheTime=ticks;
                        overAllCacheTime+=cacheTime;

                        hitCount++;

                        return 1;
                    }//hit
                    else
                    {
                            ticks2=clock();
            ticks=ticks2-ticks1;
            cacheTime=ticks;
                        overAllCacheTime+=cacheTime;

                            return 2;
                    } //not found miss

        }
        else
        {
    ticks2=clock();
    ticks=ticks2-ticks1;
    cacheTime=ticks;
        overAllCacheTime+=cacheTime;

            return 2;

        }//empty miss

        break;
}
```

```
}

long CHTMLSegmentCache::getCacheTime()
{
  return cacheTime;
}



void CHTMLSegmentCache::print(ostream &outStream)
{
  cacheIter=cacheList.begin();
  outStream<<endl<<"Segment Cache Snapshot:"<<endl;
  while(cacheIter!=cacheList.end())
  {
    outStream<<"segmentID="<<(*cacheIter)->objectID<<" ";
        outStream<<"themeID="<<(*cacheIter)->themeID<<"  ";
        ++cacheIter;

 }
    outStream<<endl<<"hitCount="<<hitCount;
    outStream<<endl<<endl;
}

int CHTMLSegmentCache::getHitCount()
{
  return hitCount;
}


long CHTMLSegmentCache::getOverAllCacheTime()
{
 return overAllCacheTime;
}



//=============================================
//InfoCache.cpp
//=============================================
#include <iostream>
using namespace std;
#include "InfoCache.h"
#include <time.h>
```

```
CInfoCache::CInfoCache()
{

}

CInfoCache::CInfoCache(int cSize)
{
        hitCount=0;
        cacheSize=cSize;
        cacheSizeUsed=0;
        cacheTime=0;
        overAllCacheTime=0;
}
CInfoCache::~CInfoCache()
{

}

int CInfoCache::cacheTimeCalculation(int flag, int infoID, int size)
{


 clock_t ticks1, ticks2,ticks;
 int minLRU;
 int i;
 bool found=false;
 ICEPointer ICEPtr;
 list<ICEPointer>::iterator tempIter;

 // get starting time
  cacheTime=0;
   ticks1=clock();

 switch(flag)
 {

 case 1: // placement ( may be replace)

            ICEPtr=new ICacheElement;
            ICEPtr->numLRU=1;
        ICEPtr->objectID=infoID;
            ICEPtr->objectSize=size;  //page size average=200K

          if(size<=(cacheSize-cacheSizeUsed))
```

```
        {
                cacheList.push_back(ICEPtr);
                cacheSizeUsed=cacheSizeUsed+ICEPtr->objectSize;

        }
        else
        {

                cacheIter=cacheList.begin();
        minLRU=(*cacheIter)->numLRU;
                tempIter=cacheIter;
                while(cacheIter!=cacheList.end())
                {
          if((*cacheIter)->numLRU<minLRU)

                        {
                                minLRU=(*cacheIter)->numLRU;
                                tempIter=cacheIter;

                        }

                                ++cacheIter;

                }

                    cacheSizeUsed=cacheSizeUsed-(*tempIter)->objectSize;
                  cacheList.erase(tempIter);
                    cacheList.push_back(ICEPtr);
                  cacheSizeUsed=cacheSizeUsed+ICEPtr->objectSize;


              }
        for(i=1;i<=35000;i++)
        {
        }
        ticks2=clock();
        ticks=ticks2-ticks1;
     cacheTime=ticks;
        return 0;
        break;

case 2: // retrieve


        if(cacheList.size()>0)
        {
```

```
        cacheIter=cacheList.begin();

        while(!found && cacheIter!=cacheList.end())
        {
    if((*cacheIter)->objectID==infoID)

                {
                        found=true;
                        tempIter=cacheIter;

                }
                else

                        ++cacheIter;
        }

         if(found)
                {
                  (*tempIter)->numLRU++;
            for( i=1;i<=35000;i++)
                        {
                        }
                        ticks2=clock();
            ticks=ticks2-ticks1;
            cacheTime=ticks;

                        hitCount++;

                        return 1;
                }//hit
                else
                {
                        ticks2=clock();
            ticks=ticks2-ticks1;
            cacheTime=ticks;
                        return 2;
                } // not found miss

    }
    else
    {
ticks2=clock();
ticks=ticks2-ticks1;
cacheTime=ticks;
            return 2;
```

```
            }//empty miss;

            break;
  }
  overAllCacheTime+=cacheTime;
}

long CInfoCache::getCacheTime()
{
  return cacheTime;
}


void CInfoCache::print(ostream &outStream)
{

  cacheIter=cacheList.begin();
  outStream<<endl<<"info Cache Snapshot:"<<endl;
  while(cacheIter!=cacheList.end())
 {
    outStream<<"infoID="<<(*cacheIter)->objectID <<"  ";
    ++cacheIter;

}

  outStream<<endl<<"hitCount="<<hitCount;
   outStream<<endl<<endl;
}


int CInfoCache::getHitCount()
{
   return hitCount;
}

//=========================================
//DualCache.cpp
//=========================================

#include "DualCache.h"
#include <time.h>
#include <stdlib.h>
#include <iostream>
using namespace std;

CDualCache::CDualCache()
```

```
{

}
CDualCache::CDualCache(int contCacheSize,int infoCacheSize)
{
   cacheTime=0;
   hitCount=0;
   HTMLSegmentCache=new CHTMLSegmentCache(contCacheSize);
   InfoCache=new CInfoCache(infoCacheSize);
   overAllCacheTime=0;
}

CDualCache::~CDualCache()
{

}

int CDualCache::cacheTimeCalculation(int workLoadType,int flag,int webBlkID,int
themeID,int size)
{
        int tempInt;
        clock_t ticks1, ticks2,ticks;
    ticks1=clock();
    cacheTime=0;
        switch(workLoadType)
        {
        case 1:

      tempInt=HTMLSegmentCache-
>cacheTimeCalculation(flag,webBlkID,themeID,size);
          hitCount=HTMLSegmentCache->getHitCount();
          break;

        case 2:
      tempInt=InfoCache->cacheTimeCalculation(flag,webBlkID,size);
          hitCount=InfoCache->getHitCount();
          break;

        }


        ticks2=clock();
    ticks=ticks2-ticks1;
    cacheTime=ticks;
        overAllCacheTime+=cacheTime;
        return tempInt;
```

```
}

long CDualCache::getCacheTime()
{
  return cacheTime;
}

int CDualCache::getHitCount()
{
 return hitCount;
}

long CDualCache::getOverAllCacheTime()
{
 return overAllCacheTime;
}
//=========================================
// PageObject.cpp
//=========================================

#include "PageObject.h"

CPageObject::CPageObject()
{

}

CPageObject::~CPageObject()
{

}

CPageObject::CPageObject(int infoSize,int contSize)
{
        for(int i=1;i<=4;i++)

        {
                webBlkID[i]=-1;
        }

                WebInfoBlkSize=infoSize;
                WebConBlkSize=contSize;
                pageSize=5*WebConBlkSize;
                pageID=-1;

}
```

```
void CPageObject::assignWebBlkID(int index,int num)
{
    webBlkID[index]=num;
}

int CPageObject::getPageSize()
{
  return pageSize;
}

int CPageObject::getWebBlkID(int index)
{
  return webBlkID[index];
}

int CPageObject::getContBlkSize()
{
        return WebConBlkSize;
}

int CPageObject::getInfoBlkSize()
{
  return WebInfoBlkSize;
}

void CPageObject::print(ostream &outStream)
{
   int i;
   outStream<<" PageID="<<pageID;
   outStream<<endl;
   outStream<<endl;
   for(i=1;i<=5;i++)
   {
    outStream<<" "<<webBlkID[i];
   }
   outStream<<endl;

}


int CPageObject::getPageID()
{
```

```
 return pageID;
}

void CPageObject::assignPageID(int ID)
{
 pageID=ID;
}
```

```
//=============================================
//ARequest.cpp
//=============================================
#include "ARequest.h"

CARequest::CARequest()
{
}
CARequest::CARequest(int infoSize,int contSize)
{
   aPageObject=new CPageObject(infoSize,contSize);
}

CARequest::~CARequest()
{
}

void CARequest::assignThemeID(int ID)
{
   themeID=ID;
}

void CARequest::assignUserID(int ID)
{
   userID=ID;
}

int CARequest::getThemeID()
{
  return themeID;
}

int CARequest::getUserID()
{
```

```
 return userID;
}

void CARequest::assignWebBlkID(int i,int num)
{

  aPageObject->assignWebBlkID(i,num);
}

void CARequest::assignPageID(int pageID)
{
 aPageObject->assignPageID(pageID);
}

void CARequest::print(ostream &outStream)
{
  outStream<<" ThemeID="<< themeID;
  outStream<<" UserID="<< userID;
  aPageObject->print(outStream);
}

int CARequest::getWebBlkID(int index)
{
   return aPageObject->getWebBlkID(index);
}

int CARequest::getPageSize()
{
   return  aPageObject->getPageSize();
}

int CARequest::getPageID()
{
   return  aPageObject->getPageID();
}

int CARequest::getContBlkSize()
{
   return  aPageObject->getContBlkSize();
}


//=============================================
// WorkLoad.cpp
//=============================================
```

```
#include "WorkLoad.h"
#include <time.h>
#include <stdlib.h>

CWorkLoad::CWorkLoad()
{

}
CWorkLoad::CWorkLoad(int infoSize,int contSize,int nTheme,int nUser,int
nWebContentBlk,int nPage)
{
                for(int j=1;j<=nUser;j++)
                {
                        WLTable[j]=new CARequest(infoSize,contSize);
                }
                numTheme=nTheme;
                numUser=nUser;
                numWebBlk=nWebContentBlk;
                numPage=nPage;
}

CWorkLoad::~CWorkLoad()
{

}
void CWorkLoad::WLInitiation()
{
        int themeID,pageID,webBlkID;
        int i,j,tempIndex;


  for(j=1;j<=numUser;j++)
  {

   WLTable[j]->assignUserID(j);
   srand((unsigned)time( NULL )*j);
   themeID=(int)( numTheme*rand() / (RAND_MAX + 1.0)); // 0~M-1
   themeID++;
      WLTable[j]->assignThemeID(themeID);

   srand((unsigned)time( NULL )*j*16789);
   pageID=(int)( numPage*rand() / (RAND_MAX + 1.0)); // 0~M-1
   pageID++;
   tempIndex=NewPageID(pageID,j);
        if(tempIndex)
        {
```

```
        WLTable[j]->assignPageID(WLTable[tempIndex]->getPageID());
        for(i=1;i<=4;i++)
        {
         WLTable[j]->assignWebBlkID(i,WLTable[tempIndex]-
      >getWebBlkID(i));
        }
       }
       else
       {
              WLTable[j]->assignPageID(pageID);
              for(i=1;i<=4;i++)
              {
                srand((unsigned)time( NULL )*j*i);
         webBlkID=(int)( numWebBlk*rand() / (RAND_MAX + 1.0));  1
             webBlkID++;
         WLTable[j]->assignWebBlkID(i,webBlkID);
              }
       }
  }
}

int CWorkLoad::NewPageID(int pageID,int index)
{
   for(int i=1;i<=numUser;i++)
        {
    if(WLTable[i]->getPageID()==pageID) return i;
    if(WLTable[i]->getPageID()==-1) return 0;
        }
        return 0;
}


void CWorkLoad::print(ostream& outStream)
{
  int i;
  outStream<<" numTheme="<< numTheme;
  outStream<<" numUser="<< numUser;
  outStream<<" numWebBlk="<< numWebBlk;
  outStream<<" numPage="<< numPage;
  outStream<<endl;
  outStream<<endl;
  for(i=1;i<=10;i++)
  {
   WLTable[i]->print(outStream);
   outStream<<endl;
```

```
    outStream<<endl;
   }

}

int CWorkLoad::getARequest(int seed)
{
   int requestID;
   srand((unsigned)time( NULL )*seed);
   requestID=(int)( numUser*rand() / (RAND_MAX + 1.0)); // 0~M-1
   requestID++;  //1~M
   return requestID;
}


int CWorkLoad::getWebBlkID(int requestID,int index)
{
 return WLTable[requestID]->getWebBlkID(index);
}

int CWorkLoad::getUserID(int requestID)
{
  return WLTable[requestID]->getUserID();
}

int CWorkLoad::getPageSize(int requestID)
{
 return  WLTable[requestID]->getPageSize();
}

int CWorkLoad::getPageID(int requestID)
{
  return WLTable[requestID]->getPageID();
}

int CWorkLoad::getThemeID(int requestID)
{
  return WLTable[requestID]->getThemeID();
}

int CWorkLoad::getContBlkSize(int requestID)
{
  return WLTable[requestID]->getContBlkSize();
}

int CWorkLoad::getInfoBlkSize(int requestID)
```

```
{
 return WLTable[requestID]->getContBlkSize();
}

void CWorkLoad::updataWLTableEntry(int requestID)
{
   int pageID,webBlkID;
        int i,tempIndex;

        srand((unsigned)time( NULL )*requestID*447156);
     pageID=(int)( numPage*rand() / (RAND_MAX + 1.0)); // 0~M-1
        pageID++;
     tempIndex=NewPageID(pageID,requestID);

        if(tempIndex)
        {

         WLTable[requestID]->assignPageID(WLTable[tempIndex]-
      >getPageID());

         for(i=1;i<=4;i++)
         {
          WLTable[requestID]->assignWebBlkID(i,WLTable[tempIndex]-
       >getWebBlkID(i));
         }
        }
        else
        {
                WLTable[requestID]->assignPageID(pageID);
                for(i=1;i<=4;i++)
                {
                  srand((unsigned)time( NULL )*requestID*i);
          webBlkID=(int)( numWebBlk*rand() / (RAND_MAX + 1.0));
         // 0~M-1
              webBlkID++;
          WLTable[requestID]->assignWebBlkID(i,webBlkID);
                }
        }

}

//=============================================
// WebServer.h
//=============================================
```

```
#if !defined(AFX_WEBSERVER_H__F2C8A9E6_60EF_4D61_A032_75EB0269257B_
_INCLUDED_)
#define
AFX_WEBSERVER_H__F2C8A9E6_60EF_4D61_A032_75EB0269257B__INCLUDE
D_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CWebServer
{
private:
        double dispatchTime;
        double sendTime;  //clock #

public:
        double getSendTime();
        double getDispatchTime();
        CWebServer();
        CWebServer(double dTime,double sTime);
        virtual ~CWebServer();

};

#endif
// !defined(AFX_WEBSERVER_H__F2C8A9E6_60EF_4D61_A032_75EB0269257B__
INCLUDED_)


//=============================================
// AppServer.h
//=============================================

#if !defined(AFX_APPSERVER_H__8D33B6CC_2DA2_4EA8_A3B6_27F3C545DE2
C__INCLUDED_)
#define
AFX_APPSERVER_H__8D33B6CC_2DA2_4EA8_A3B6_27F3C545DE2C__INCLUD
ED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CAppServer
{
```

```
private:
        double webInfoBlockTime;
        double webContentBlockTime;
        double diskAccessTime;
public:
        double getDiskAccessTime();
        double getContentBlockTime();
        double getInfoBlockTime();

        CAppServer();
        CAppServer(double infoTime,double contTime,double diskTime);
        virtual ~CAppServer();

};

#endif
// !defined(AFX_APPSERVER_H__8D33B6CC_2DA2_4EA8_A3B6_27F3C545DE2C_
_INCLUDED_)


//=========================================
// DBServer.h
//=========================================

#if !defined(AFX_DBSERVER_H__14034D14_5C86_4A33_8AB6_409B5C8729A7__I
NCLUDED_)
#define
AFX_DBSERVER_H__14034D14_5C86_4A33_8AB6_409B5C8729A7__INCLUDED
_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CDBServer
{
private:
        double DBTime;
public:
        double getDBTime();
        CDBServer();
        CDBServer(double time);
        virtual ~CDBServer();

};
```

```
#endif
// !defined(AFX_DBSERVER_H__14034D14_5C86_4A33_8AB6_409B5C8729A7__I
NCLUDED_)


//=====================================
// PageCache.h
//=====================================

#if !defined(AFX_PAGECACHE_H__FB2A8EC8_91A6_4234_B3C5_119F1A5A7203
__INCLUDED_)
#define
AFX_PAGECACHE_H__FB2A8EC8_91A6_4234_B3C5_119F1A5A7203__INCLUDE
D_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <string.h>
#include <list>
using namespace std;

struct PCacheElement
{
  int objectID;
  int objectSize;
  int numLRU;
  int userID;
};

typedef struct PCacheElement *PCEPointer;
typedef struct PCacheElement PCacheElement;


class CPageCache
{
private:
        int hitCount;
        int cacheSize;
        int cacheSizeUsed;
   long cacheTime;
        long overAllCacheTime;
        list<PCEPointer> cacheList;
        list<PCEPointer>::iterator  cacheIter;
```

```
public:
        long getOverAllCacheTime();

         int cacheTimeCalculation(int flag,int pageID,int userID,int size);
        int getHitCount();
        long getCacheTime();
        void print(ostream& outStream);

        CPageCache();
        CPageCache(int cSize);
        ~CPageCache();
};

#endif
// !defined(AFX_PAGECACHE_H__FB2A8EC8_91A6_4234_B3C5_119F1A5A7203__
INCLUDED_)


//=============================================
//HTMLSegmentCache.h
//=============================================

#if !defined(AFX_HTMLSEGMENTCACHE_H__AA7D19B4_2F07_420C_9FFE_9C6
5D6F14780__INCLUDED_)
#define
AFX_HTMLSEGMENTCACHE_H__AA7D19B4_2F07_420C_9FFE_9C65D6F14780_
_INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#include <string.h>
#include <list>
using namespace std;

struct SCacheElement
{
  int objectID;
  int objectSize;
  int numLRU;
  int themeID;
};

typedef struct SCacheElement *SCEPointer;
typedef struct SCacheElement SCacheElement;
```

```
class CHTMLSegmentCache
{
private:
        int hitCount;
        int cacheSize;
        int cacheSizeUsed;
   long cacheTime;
        long overAllCacheTime;
        list<SCEPointer> cacheList;
        list<SCEPointer>::iterator  cacheIter;

public:
        long getOverAllCacheTime();

        int cacheTimeCalculation(int flag,int segmentID,int themeID,int size);
        long getCacheTime();
        int getHitCount();
        void print(ostream& outStream);

        CHTMLSegmentCache();
        CHTMLSegmentCache(int cSize);
        ~CHTMLSegmentCache();



};

#endif
// !defined(AFX_HTMLSEGMENTCACHE_H__AA7D19B4_2F07_420C_9FFE_9C65
D6F14780__INCLUDED_)


//=============================================
//InfoCache.h
//=============================================

#if !defined(AFX_INFOCACHE_H__36F7C4F4_B2E0_41FA_AB55_FB3409068876__
INCLUDED_)
#define
AFX_INFOCACHE_H__36F7C4F4_B2E0_41FA_AB55_FB3409068876__INCLUDE
D_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#include <string.h>
```

```
#include <list>
using namespace std;

struct ICacheElement
{
  int objectID;
  int objectSize;
  int numLRU;
};

typedef struct ICacheElement *ICEPointer;
typedef struct ICacheElement ICacheElement;
class CInfoCache
{
private:
        int hitCount;
        int cacheSize;
        int cacheSizeUsed;
    long cacheTime;
        long overAllCacheTime;
        list<ICEPointer> cacheList;
        list<ICEPointer>::iterator  cacheIter;
public:

        long getCacheTime();
        int cacheTimeCalculation(int flag, int infoID, int size);
        int getHitCount();
        void print(ostream& outStream);

        CInfoCache();
        CInfoCache(int cSize);
        virtual ~CInfoCache();



};

#endif
// !defined(AFX_INFOCACHE_H__36F7C4F4_B2E0_41FA_AB55_FB3409068876__I
NCLUDED_)


//=============================================
// DualCache.h
//=============================================
```

```
#if !defined(AFX_DUALCACHE_H__6D436931_8AAF_4953_984A_64F9136589A0_
_INCLUDED_)
#define
AFX_DUALCACHE_H__6D436931_8AAF_4953_984A_64F9136589A0__INCLUDE
D_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#include "HTMLSegmentCache.h"
#include "InfoCache.h"

class CDualCache
{

private:
        CHTMLSegmentCache* HTMLSegmentCache;
        CInfoCache* InfoCache;
        long cacheTime;
        long overAllCacheTime;
        int hitCount;

public:
        long getOverAllCacheTime();
        int cacheTimeCalculation(int workLoadType,int flag,int
    webBlkID,int themeID,int size);
        long getCacheTime();
        int getHitCount();

        CDualCache();
        CDualCache(int contCacheSize,int infoCacheSize);
   ~CDualCache();

};

#endif
// !defined(AFX_DUALCACHE_H__6D436931_8AAF_4953_984A_64F9136589A0__I
NCLUDED_)


//=============================================
// PageObject.h
//=============================================

#if !defined(AFX_PAGEOBJECT_H__F1F6C710_ABB8_4B8F_97FF_575787A3EFB4
__INCLUDED_)
```

```
#define
AFX_PAGEOBJECT_H__F1F6C710_ABB8_4B8F_97FF_575787A3EFB4__INCLUD
ED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <iostream>
using namespace std;
class CPageObject
{
private:
  int webBlkID[5];
  int WebInfoBlkSize;
  int WebConBlkSize;
  int pageSize;
  int pageID;

public:

    void assignPageID(int pageID);
        void assignWebBlkID(int index,int num);
        void print(ostream& outStream);

        int getPageID();
        int getInfoBlkSize();
        int getContBlkSize();
        int getPageSize();
        int getWebBlkID(int index);

        CPageObject();
        CPageObject(int infoSize,int contSize);
        ~CPageObject();

};

#endif
// !defined(AFX_PAGEOBJECT_H__F1F6C710_ABB8_4B8F_97FF_575787A3EFB4_
__INCLUDED_)


//=========================================
// ARequest.h
//=========================================
```

```
#if !defined(AFX_AREQUEST_H__F82BC27B_9F38_4B20_9582_4C38FB403A26__I
NCLUDED_)
#define
AFX_AREQUEST_H__F82BC27B_9F38_4B20_9582_4C38FB403A26__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#include <iostream>
using namespace std;
#include "PageObject.h"

class CARequest
{
private:
        CPageObject* aPageObject;
        int themeID;
        int userID;


public:

        void assignPageID(int pageID);
        void assignWebBlkID(int index,int ID);
        void assignUserID(int ID);
        void assignThemeID(int ID);
    void print(ostream& outStream);

        int getContBlkSize();
        int getPageID();
        int getPageSize();
        int getWebBlkID(int index);
        int getUserID();
        int getThemeID();


        CARequest();
        CARequest(int infoSize,int contSize);
        ~CARequest();

};

#endif
// !defined(AFX_AREQUEST_H__F82BC27B_9F38_4B20_9582_4C38FB403A26__IN
CLUDED_)
```

```
//=============================================
// WorkLoad.h
//=============================================

#if !defined(AFX_WORKLOAD_H__E771D69B_2907_4F14_8EF2_9162B6CC46CA_
_INCLUDED_)
#define
AFX_WORKLOAD_H__E771D69B_2907_4F14_8EF2_9162B6CC46CA__INCLUDE
D_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <iostream>
using namespace std;
#include "ARequest.h"


class CWorkLoad
{
 private:
  CARequest* WLTable[1001];
        int numTheme;
        int numUser;
        int numWebBlk;
        int numPage;

 public:

        CWorkLoad();
        CWorkLoad(int infoSize,int contSize,int nTheme,int nUser,int
    nWebContentBlk,int nPage);
   ~CWorkLoad();

        void WLInitiation();
        int NewPageID(int pageID,int index);
        void updataWLTableEntry(int requestID);
        void print(ostream& outStream);
```

```
        int getContBlkSize(int requestID);
        int getThemeID(int requestID);
        int getPageID(int requestID);
        int getPageSize(int requestID);
        int getUserID(int requestID);
        int getWebBlkID(int requestID,int index);
        int getARequest(int seed);
    int getInfoBlkSize(int requestID);

};

#endif
// !defined(AFX_WORKLOAD_H__E771D69B_2907_4F14_8EF2_9162B6CC46CA__I
NCLUDED_)
```

APPENDIX E

Table theme: Every theme is identified uniquely by a themeID which defines characteristics of a kind of Web page styles.

Field	Type	Attributes	Null	Default	Extra
ID	int(10)		No		auto_increment
themeID	int(10)		No	0	
naviTable	varchar(85)		No		
naviTd	varchar(80)		No		
itemsTable	*varchar(110)*		No		
itemsTd	varchar(50)		No		
mainFont	varchar(50)		No		
secondFont	varchar(50)		No		
thirdFont	varchar(50)		No		
endFontToken	varchar(20)		No		

Table users: Every user is identified uniquely by a userID. Every user has a themeID which represents his preference of the Web page Style.

Field	Type	Attributes	Null	Default	Extra
ID	int(100)		No		auto_increment
userID	int(100)		No	0	
userName	varchar(50)		No		
themeID	int(20)		No	0	

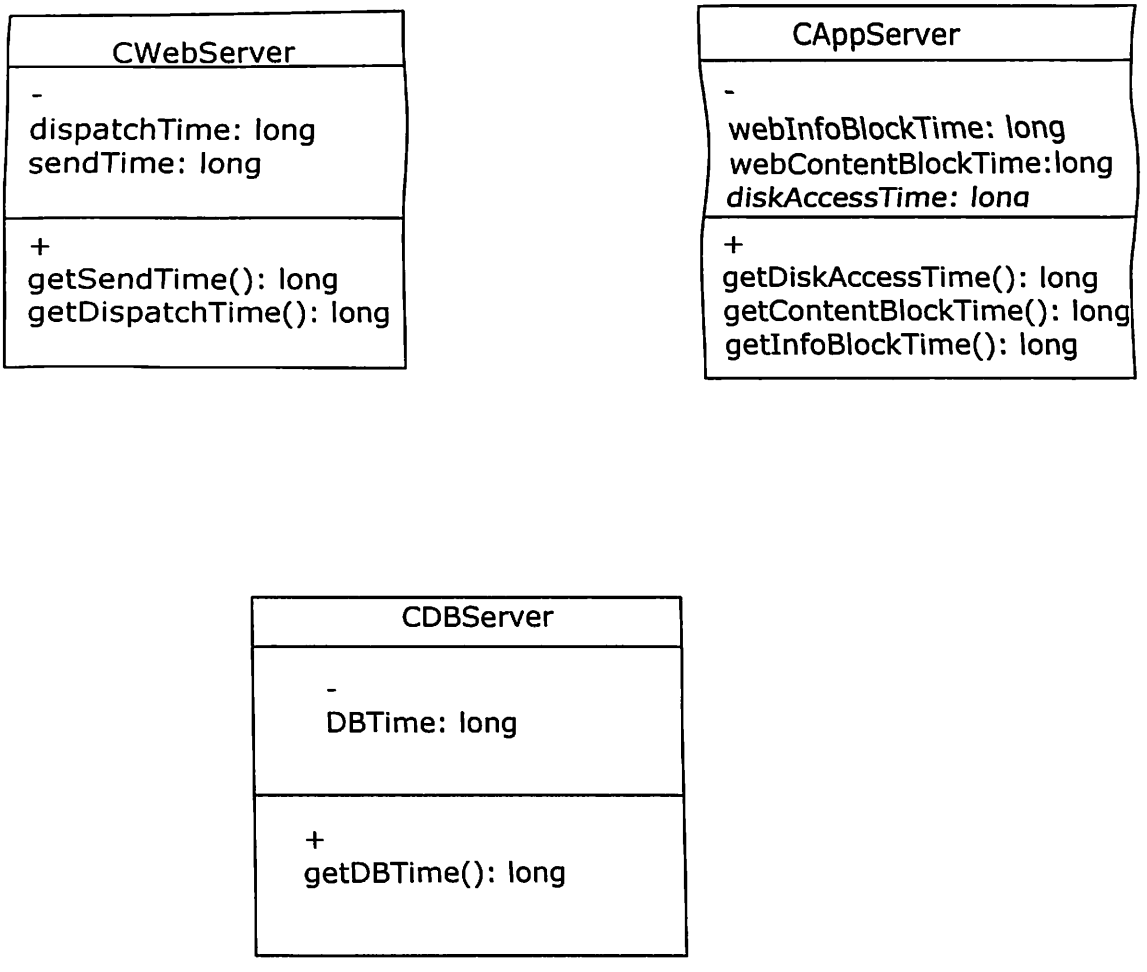
Table items: Every item has a name, an image, a price and a detailed description as well as the catalog this item belongs to.

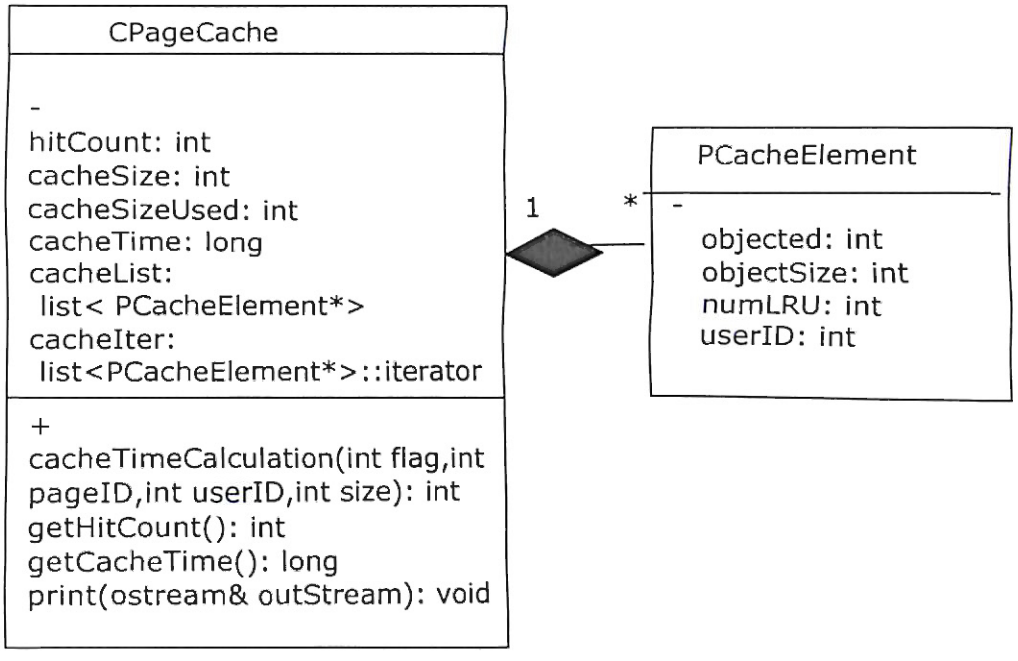
Field	Type	Attributes	Null	Default	Extra
ID	int(11)		No		auto_increment
itemName	varchar(30)		No		
itemImage	varchar(50)		No		
price	varchar(20)		No		
details	varchar(200)		No		
catalog	varchar(50)		No		

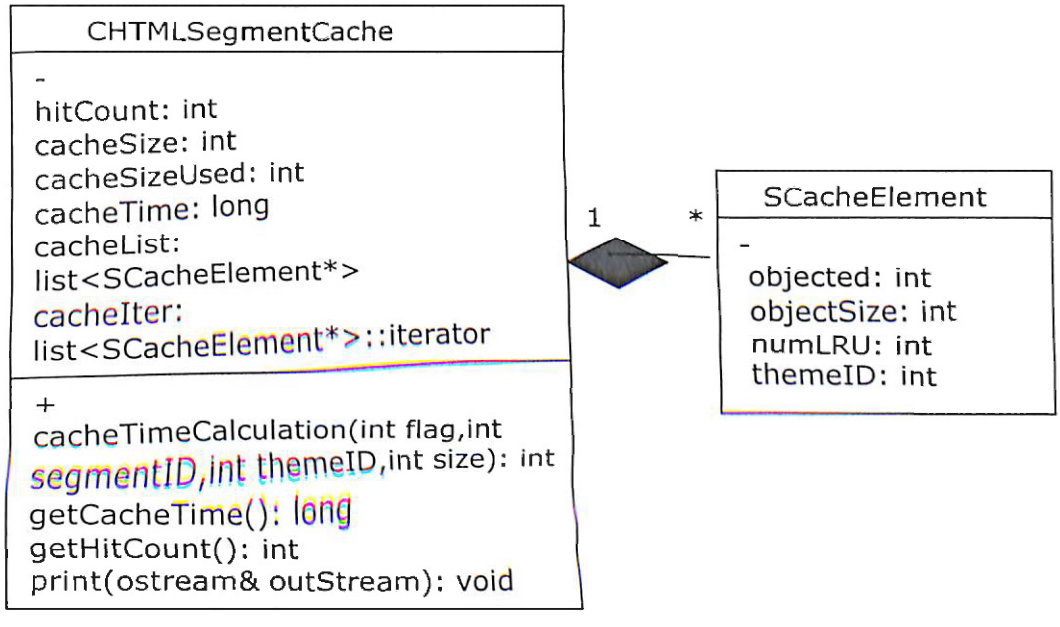
Table Navibar:

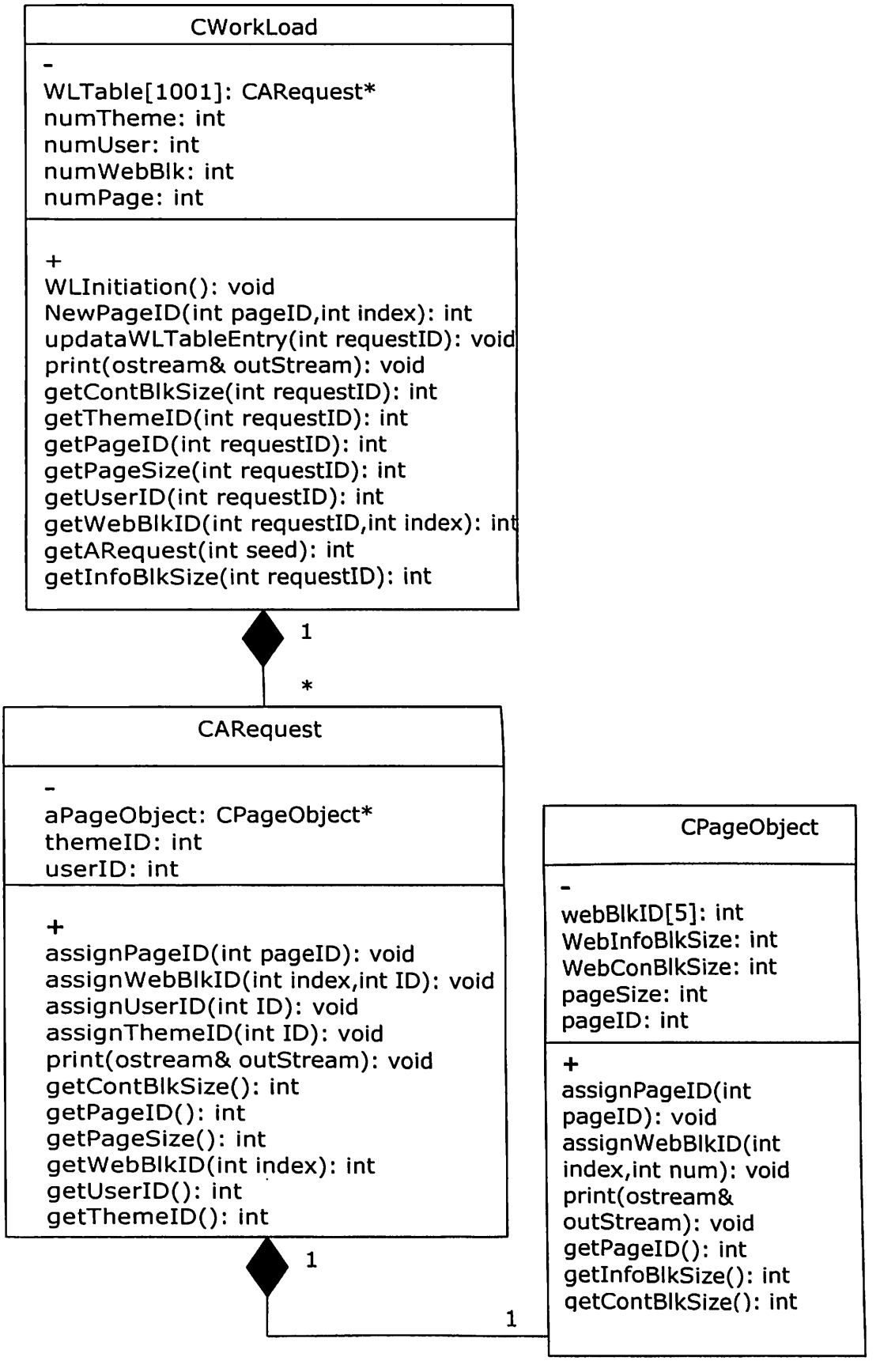
Field	Type	Attributes	Null	Default	Extra
ID	int(10)		No		auto_increment
buttonName	varchar(50)		No		

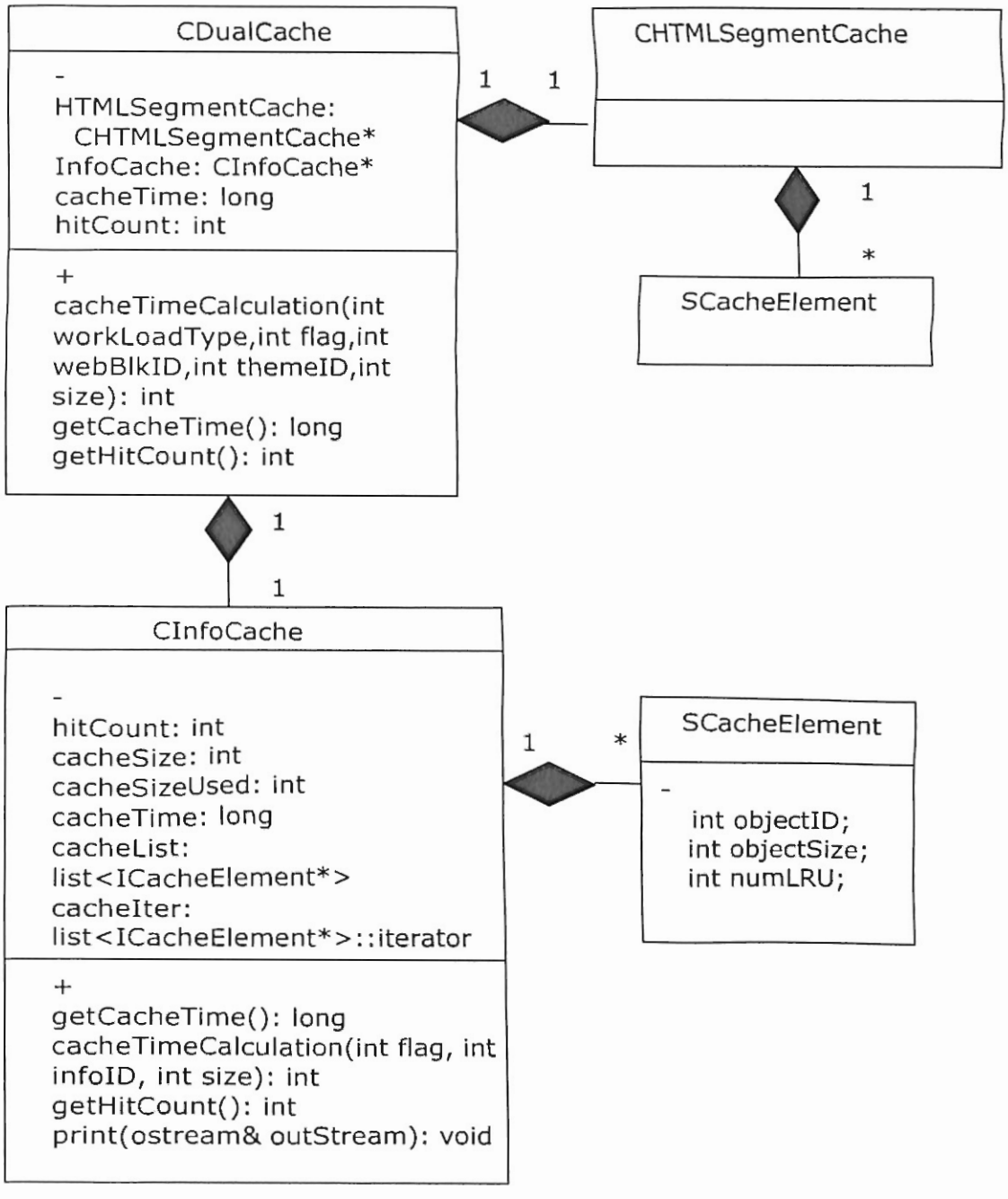
APPENDIX F


CWebServer
-
dispatchTime: long
sendTime: long
+
getSendTime(): long
getDispatchTime(): long
CAppServer
-
webInfoBlockTime: long
webContentBlockTime:long
diskAccessTime: long
+
getDiskAccessTime(): long
getContentBlockTime(): long
getInfoBlockTime(): long
CDBServer
-
DBTime: long
+
getDBTime(): long


CPageCache
-
hitCount: int
cacheSize: int
cacheSizeUsed: int
cacheTime: long
cacheList:
list< PCacheElement*>
cacheIter:
list<PCacheElement*>::iterator
+
cacheTimeCalculation(int flag,int pageID,int userID,int size): int
getHitCount(): int
getCacheTime(): long
print(ostream& outStream): void
1
*
PCacheElement
-
objected: int
objectSize: int
numLRU: int
userID: int


CHTMLSegmentCache
-
hitCount: int
cacheSize: int
cacheSizeUsed: int
cacheTime: long
cacheList:
list<SCacheElement*>
cacheIter:
list<SCacheElement*>::iterator
+
cacheTimeCalculation(int flag,int segmentID,int themeID,int size): int
getCacheTime(): long
getHitCount(): int
print(ostream& outStream): void
1
*
SCacheElement
-
objected: int
objectSize: int
numLRU: int
themeID: int


CWorkLoad
-
WLTable[1001]: CARequest*
numTheme: int
numUser: int
numWebBlk: int
numPage: int
+
WLInitiation(): void
NewPageID(int pageID,int index): int
updataWLTableEntry(int requestID): void
print(ostream& outStream): void
getContBlkSize(int requestID): int
getThemeID(int requestID): int
getPageID(int requestID): int
getPageSize(int requestID): int
getUserID(int requestID): int
getWebBlkID(int requestID,int index): int
getARequest(int seed): int
getInfoBlkSize(int requestID): int
1
*
CARequest
-
aPageObject: CPageObject*
themeID: int
userID: int
+
assignPageID(int pageID): void
assignWebBlkID(int index,int ID): void
assignUserID(int ID): void
assignThemeID(int ID): void
print(ostream& outStream): void
getContBlkSize(): int
getPageID(): int
getPageSize(): int
getWebBlkID(int index): int
getUserID(): int
getThemeID(): int
1
1
CPageObject
-
webBlkID[5]: int
WebInfoBlkSize: int
WebConBlkSize: int
pageSize: int
pageID: int
+
assignPageID(int
pageID): void
assignWebBlkID(int
index,int num): void
print(ostream&
outStream): void
getPageID(): int
getInfoBlkSize(): int
getContBlkSize(): int


CDualCache
-
HTMLSegmentCache:
CHTMLSegmentCache*
InfoCache: CInfoCache*
cacheTime: long
hitCount: int
+
cacheTimeCalculation(int
workLoadType,int flag,int
webBlkID,int themeID,int
size): int
getCacheTime(): long
getHitCount(): int
1
1
CHTMLSegmentCache
1
*
SCacheElement
1
1
CInfoCache
-
hitCount: int
cacheSize: int
cacheSizeUsed: int
cacheTime: long
cacheList:
list<ICacheElement*>
cacheIter:
list<ICacheElement*>::iterator
+
getCacheTime(): long
cacheTimeCalculation(int flag, int
infoID, int size): int
getHitCount(): int
print(ostream& outStream): void
1
*
SCacheElement
-
int objectID;
int objectSize;
int numLRU;

VITA (1)

Yingcui Qu

Candidate for the Degree of

Master of Science

Thesis: WEBSITE ARCHITECTURE AND WEB PROGRAMMING CO-DESIGN FOR ACCELERATING HIGHLY PERSONALIZED DYNAMIC WEB PAGES GENERATION

Major Field: Computer Science

Biographical:

Personal Data: Born in Tacheng, Xinjiang, P. R. China, Nov. 10, 1976, the daughter of Mr. Guishan Qu and Mrs. Peiyun Wang

Education: Graduated from No. 1st High school of Kelamayi, Xinjiang, P. R. China, in July 1995; received bachelor degree in Chemical Engineering and Minor degree in Computer Science from University of Petroleum, Shandong, China, in July 1999; completed the requirements for Master of Science at Oklahoma State University in May, 2004.

Professional Experience: Software developer in Chemical Engineering Department in University of Petroleum using Visual C++ in 1999; employed by Web office of Art & Science in Oklahoma State University as a Web programmer since Nov. 2001.