Challenges and Practices in Deploying Web Acceleration Solutions for Distributed Enterprise Systems

Wen-Syan Li  Wang-Pin Hsiung  Oliver Po  Koji Hino  K. Selçuk Candan  Divyakant Agrawal

NEC Laboratories America, Inc.
10080 North Wolfe Road, Suite SW3-350, Cupertino, California 95014, USA

ABSTRACT
For most Web-based applications, contents are created dynamically based on the current state of a business, such as product prices and inventory, stored in database systems. These applications demand personalized content and track user behavior while maintaining application integrity. Many of such practices are not compatible with Web acceleration solutions. Consequently, although many web acceleration solutions have shown promising performance improvement and scalability, architecting and engineering distributed enterprise Web applications to utilize available content delivery networks remain a challenge. In this paper, we examine the challenge to accelerate J2EE-based enterprise web applications. We list obstacles and recommend some practices to transform typical database-driven J2EE applications to cache friendly Web applications where Web acceleration solutions can be applied. Furthermore, such transformation should be done without modification to the underlying application business logic and without sacrificing functions that are essential to e-commerce. We take the J2EE reference software, the Java PetStore, as a case study. By using the proposed guideline, we are able to cache more than 90% of the content in the PetStore and scale up the Web site more than 20 times.

Categories and Subject Descriptors
H.4 [Information Systems]: Information Systems Applications; D.2 [Software]: Software Engineering; D.2.8 [Software Engineering]: Metrics—complexity measures, performance measures

General Terms
Performance, Reliability, Experimentation

Keywords
J2EE, dynamic content, application server, edge server, fragment, web acceleration, reliability, scalability

1. INTRODUCTION
For many e-commerce applications, Web pages are created dynamically based on the current state of a business, such as product prices and inventory, stored in database systems. This characteristic requires e-commerce Web sites to deploy Web servers, application servers, and database management system (DBMS) to generate and serve user requested content dynamically. When the Web server receives a request for dynamic content, it forwards the request to the application server along with its request parameters (typically included in the URL string). The Web server communicates with the application server using URL strings and cookie information, which is used for customization. When the application server receives such a request from the Web server, it may query the underlying databases to extract the relevant information needed to dynamically generate the requested page.

To improve the response time, one option is to build a high performance Web site by improving network and server capacity by deploying a state of the art IT infrastructure. However, without the deployment of dynamic content caching solutions and content delivery networks (CDN), dynamic contents are generated on demand. In this case, all delivered Web pages are generated based on the current business state in the source database.

To improve scalability and performance, one solution is to deploy network-wide caches so that a large fraction of requests can be served remotely rather than all of them being served from the origin Web site. This solution has the advantage of serving users via caches closer to them and reducing the traffic to the Web sites, reducing network latency, and providing faster response times. Many CDN services [1] provide Web acceleration services. A study in [2] shows that CDN indeed has significant performance impact. However, for many e-commerce applications, content is created dynamically based on the current state of a business, such as product prices and inventory, rather than static information. Therefore, content delivery by most CDNs is limited to handling static portions of the pages and media objects, rather than the full spectrum of dynamic content that constitutes the bulk of the e-commerce Web sites.

Wide-area database replication technologies and the availability of data centers allow database copies to be distributed across the network. This requires a complete e-commerce web site suite (i.e., Web servers, application servers, and DBMS) to be distributed along with the database replicas. A major advantage of this approach is, like the caches, the possibility of serving dynamic content from a location close to the users, reducing network latency.

Many web acceleration solutions, such as [3, 4, 5, 6, 7, 8], have shown promising performance improvement and scalability. Li et al. in [9] provide evaluations of architectural designs and various implementation practices for database-driven Web sites. In [10], Li et al. further analyze the factors that have impacts on the performance and scalability of Web applications and outline a road map for Web acceleration for dynamic content. However, with the necessary requirements for enterprise Web applications, such as personalization and user behavior tracking, architecting and engineering distributed enterprise Web applications based on available Web acceleration solutions remains a challenge.

Copyright is held by the author/owner(s).
ACM 1-58113-844-X/04/0005.
After a browser receives the page index.html, it parses the page and then requests three additional pages. Note that although this composite page is displayed to the user as a single and personalized Web page, all fragment pages are cacheable. The category page and search page have fairly long TTL and the personalized fragment page is also cacheable for the individual user and it can be invalidated when a new purchase occurs.

For this frame-based implementation, an edge cache server deployment will be most suitable for fast delivery through caching.

3.10.2 Dynamically-assembly Implementation

If the requested page is dynamically assembled at user request time, all three fragment pages are cacheable but an edge application server is needed to assemble the three fragment pages into one page to serve the user. In this system architecture, the invalidation message for the personalization page needs to be sent to the edge application server instead to the cache server. The network latency and processing latency for fragment pages are reduced.

Note that in the case that neither the edge application server nor the CachePortal is deployed, the application server at the origin site needs to dynamically generate the personalization page for each user and then assemble the three fragment pages into one page to serve the user. The combination of network latency and processing latency may result in slow response time.

3.11 Cluster Architecture

A typical approach to supporting high-availability and scalability is to use a cluster architecture for Web/Application Server and DBMS. If middleware is deployed to provide transparent access to WAS, such as [14], and DBMS, such as [15], the cluster architecture is not really an obstacle to construct the URL and query mapping. If the cluster architecture does not provide transparent access or shared memory, it requires additional effort and it is challenging to monitor all activities across multiple servers.

4. CASE STUDY: JAVA PETSTORE

The Java PetStore is a reference application provided by the Java 2 Platform, Enterprise Edition BluePrints (J2EE BluePrints) program at Java Software, Sun Microsystems. This reference application demonstrates how to use the capabilities of the J2EE platform to develop flexible, scalable, cross-platform e-business applications. It comes with full source code and documentation for users to experiment with J2EE technology and learn how to use it to build enterprise solutions. The J2EE BluePrints program defines the application programming model for the J2EE platform. It provides practice guidelines and architectural recommendations for real-world application scenarios to enable developers to build portable, scalable, and robust applications using J2EE technology. In this paper, we use it to demonstrate how we can enable dynamic content caching and accelerate an e-commerce J2EE applications.

4.1 Overview of the Java PetStore

We installed the Java PetStore and populated the database as instructed in the manual. We then deployed CachePortal on top of the
Java PetStore. The PetStore represents a typical e-commerce Web site, where users are provided with options to navigate through the catalog, items, and place orders.

Figure 6 shows the main page of the Java PetStore, where the users can see all five categories of pets as well as performing search on the whole inventory of pets. When the users click on one of the category page links, a category page similar to Figure 7 will appear. Then, the users can click on one of the two links in the category page to go to sub-categories until an item page is reached. An example of the item page is shown in Figure 8. At the item page, the users can add the pet to the shopping cart or click on the link to go to the information page. At the information page (as shown in Figure 9), the users are presented with detailed information about the pet and have option to click on add to cart link. When the add to cart link is clicked, the users can modify the quantity or remove the items from the shopping cart completely. The users have an option to check out and log off from the PetStore. At any page, the users can click on the categories, sign in, account, and change language links.

Note that users can navigate the categories and add items to shopping carts without signing in. However, after the users login to the system, the system will provide a personalized page, My List, on the right side of the window as shown in Figure 10. The links in My List is based on what the users purchased recently and the database content is accessed every time when personalization is enabled.

4.2 Enabling Dynamic Content Caching

The purpose of our case study is to see if we are able to enable dynamic content caching without modification to its application program. The tools available to us via CachePortal include:

- CachePortal sniffer and invalidator as described in Section 2;
- Page Differentiator as described in Section 3.7;
- Squid cache server with enhancement of POST message handling and configurability through Page Differentiator;
- Light weight edge application server that is able to assemble fragment pages if necessary. The edge application server is integrated with the Squid cache server.

We do not modify the Java PetStore to build an API for the URL encoding/decoding purpose and passing cacheability and hashing keys to the edge cache/application server. There are five steps to enable dynamic content caching for PetStore.

There are total 84 possible screens in the Java PetStore. Among them, 79 screens, catalog and item information pages, are cacheable.

The remaining 5 non-cacheable screens are sign in, account, check cart, add to cart, and checkout.

In the Java PetStore, cookies and session objects are created when the users login to the main screen of the system. However, the cookie information is not really used in the PetStore while the session objects are used to store shopping cart information.

We navigated through every page in the Java PetStore and identified that the URL strings can be classified into the following ten patterns:

1. petstore/category.screen?category_id=FISH
2. petstore/product.screen.screen?product_id=AV-CB-01
3. petstore/item.screen?item_id=EST-18
4. petstore/changelocale.do?locale=en_US
5. petstore/changelocale.do?locale=ja_JP

6. petstore/signon_welcome.screen
7. petstore/cart.do
8. petstore/customer.do
9. petstore/cart.do?action=purchase&itemId=EST-19
10. petstore/enter_order_information.screen

On the list, the first three URLs are for browsing categories, subcategories, items, and item information. The fourth and the fifth URLs are for language switching between English and Japanese. All of these pages are cacheable using Page Differentiator to configure the edge cache/application server to ignore the session objects and cookies. The remaining URLs (from the sixth to the tenth) are for the sign in screen, check cart screen, account screen, add to cart screen, and checkout screen. They are not cacheable.

Note that in this list, we can only show the URL strings for user requests.

Since fragment pages are used in the Java PetStore, additional included and forwarded requests for the fragment pages are sniffed at the application server filter. The URL and query map stores the mapping between

- URL of origin user requests and queries,
- URL of origin user requests and queries and URLs of included and forwarded requests for the fragment pages, and
- URL of requests for the fragment pages and queries.

For example, for the request to generate the page shown in Figure 7, a URL string, petstore/category.screen?category_id=FISH, is captured by the sniffer as A request results in a page with two fragment pages, category on the left and product_id=FISH on the right in Figure 7. The internal request for the fragment page category results in the following query, Query1, issued to the database:

```
select a.catid, name, descn
from category a, category_details b
where a.catid = b.catid and locale = "en_US"
order by name
```

The sniffer also captures the query, Query2, issued as a result of the fragment page category_id=FISH. Query2 is as follows:

```
select a.productid, name, descn
from product a, product_details b
where a.productid = b.productid
    and locale = "en_US"
    and a.catid = "FISH"
order by name
```
For the user request for a category page for pets related to dogs, the sniffer captures Query1 and the following query, Query3, for the fragment page, category_id=DOGS:

```
select a.productid, name, descn
from product a, product_details b
where a.productid=b.productid
and locale = "en_US"
and a.catid = "DOGS"
order by name
```

For the same URL request with a user login, a customized page shown in Figure 11 is generated. Additional query, Query4, is issued to retrieve the purchase history of the user and to generate the fragment page, my_list, on the right of Figure 11. When a request for the page shown in Figure 8 is issued, the sniffer captures the URL /petstore/product.screen?product_id=AV−CB−01 as well as two queries to the database, Query1 for the category fragment page and the following query, Query5, to generate the fragment page, product_id=AV−CB−01, containing detailed information about product items:

```
select catid, name, a.itemid, b.image, b.descn,
attr1, attr2, attr3, attr4, attr5, listprice, unitcost
from item a, item_details b, product d
where a.itemid=b.itemid
and a.productid=c.productid
and d.productid=c.productid
and b.locale = "en_US"
and a.productid = "AV−CB−01"
```

### 4.3 Monitoring and Invalidation

In Figure 12, we summarize the relationship between HTML pages cached at edge cache servers and fragment pages cached in either at the edge application servers or at the application servers (at the origin site). The relationship is captured by the application sniffer automatically. The figure also illustrates the map between fragment pages and queries, which is captured by the JDBC sniffer. Note that if a HTML page is generated without using any fragment page, a direct map between the HTML page and queries is constructed based on multiple fragment pages with or without user login. Also note that a request for a page may result in multiple queries.

From these captured query statements, we can identify a list of tables, including category, product, product_details, item, and item_details, that store information necessary to generate requested pages. Thus, these tables need to be monitored for invalidating cached HTML pages or fragments when database content changes occur. For example, if a database change is detected and the invalidator determines that Query3 needs to be invalidated, the fragment page category_id=DOGS is invalidated from the edge application servers and consequently the HTML page category.screen?category_id=DOGS needs to be invalidated. For detailed description of the invalidation scheme, please see [10].

CachePortal is designed to accelerate very large scale data center hosted database-driven Web applications. We have evaluated the system using an e-commerce application and it is capable of tracking 50 million dynamic content pages in 10 cache servers and it assures content freshness of these 50 million pages by invalidating impacted pages within 12 seconds once database content is changed. Our approach also provides better scalability and significantly reduced response times up to 70% in the experiments. Some experimental evaluation results are described in [16, 12].

The Java PetStore is now dynamic content caching enabled and is ready for the evaluation of performance gain through the deployment of CachePortal. Note that since HTML pages in Java PetStore are personalized if users login and they have purchased some items in the past, the cache hits on edge cache servers are only for requests from the users who do not login or have no purchase history. For all other requests other than transactional requests related to orders and shopping carts, edge application servers can dynamically assemble requested pages based on cached fragments at the edge application servers. The invalidations will occur to the following fragments:

- **my_list fragment** (on the right Figure 11) when a user makes new purchases;
- **category fragment** (on the left of Figures 6 to 11) when a new category is added to the PetStore;
- **category_id fragment** (on the right of Figure 7) when a new item added to a category;
- **item fragment** (on the right of Figure 8) when a new item is added to a category or the standard list price for an existing item is changed; and
- **item_details fragment** (on the right of Figure 9) when the standard list price or special price for an existing item is changed.
5. EXPERIMENTS

In this section, we present the experimental results of evaluating the Java PetStore’s performance gain and scalability improvement with the deployment of NEC’s CachePortal Web acceleration solution and the proposed guideline. We first describe the general experiment setup that consists of Web servers, application servers, DBMS, and network infrastructure that are used in the experiments.

5.1 General Experimental Setting

The content delivery configuration is similar to that described in Section 2. We used two heterogeneous networks that are available in the NEC’s facility in Cupertino, California: one is used by the C&C Research Laboratories (referred to as CCRL) and the other one is used by cacheportal.com (referred to as CP). Users and edge cache servers are located in the CP network while Web server, application server, and DB Caches are located in the CCRL network. The average round trip time on the CCRL-CP connections is through a network delay generator using Dummy Net provided by FreeBSD. The round trip time within the same network is negligible. In summary, connectivity within the same network is substantially better than that across the Internet and there is large network latency.

The system and software configuration for each component are as follows:

- Edge cache server: Squid 2.4.7 with enhancements to handle POST requests on a 1500 MHZ Pentium 4 machine running Linux 8.0.
- Web server and application server: Tomcat 4.1.24 Web Server and JBoss 3.2.1 Application Server are used on a 1500 MHZ Pentium 4 machine with 1 G Byte main memory running Linux 8.0.
- DBMS: Oracle 9i is used as the database system and it runs on a 2300 MHZ Xeon machine with 2 G Byte main memory running Linux 8.0.

5.2 Evaluation of Performance Gain

The first experiment we conducted is to measure the performance gain (in terms of the response time observed by the users) achieved through our proposed approach. In this experiment, the network latency is set to 200ms, 400ms, and 800ms and the number of concurrent users are set to in a range between 20 and 100. We randomly assign the users to navigate the Java PetStore applications. Thus, some users may look at the catalog while some of them may add items to shopping carts and are ready to check out. We ran the experiments based on combinations of all these parameter values for the PetStore without CachePortal and PetStore with the deployment of CachePortal with cache hit rates at 60%, 70%, 80%, and 90% (i.e., hits at the edge cache servers or the edge application servers). We then record the response time of each users. The experiments are repeated five times and the average user response time is used to plot in Figure 13.

As we can see in Figure 13, the system without dynamic content caching and CachePortal yields very large response time and the response time increases almost in proportion to the number of concurrent users and network latency. On the other hand, the systems with dynamic content caching and CachePortal consistently provide fast average response time under 4 seconds.

5.3 Evaluation of User Experience

The next experiment we conducted is to measure user experience in terms of

- percentage of the users receiving error message due to system overload or time out; and
- percentage of the users receiving response under 7 seconds.

The experiment setting is the same as the previous experiment except we vary the number of concurrent users between 20 and 200 in order to create system overload.

Note that these experiments illustrate the user’s actual experience. As we can see in Figure 14, at the load of 50 concurrent users, the users of the Java PetStore start to experience unpleasant delay and at the load of more than 100 concurrent users, more than half of the users experience delay more than 7 seconds. In Figure 15, we can see at the load of more than 120 concurrent requests, the users start to receive error messages.

In the figure, we also see that the network latency has an impact to the user response time. Note that the network latency not only increases the transmission time but also cause the requests to hold on to the limited number of connections to the application servers and the DBMS.

5.4 Experiments on System Scalability

The next experiment we conducted is to measure the scalability of five different systems for Java PetStore: one system without dynamic content caching and four systems with dynamic content catching enabled and deployment of CachePortal. These four systems have hit rates of 60%, 70%, 80%, and 90%, respectively. We want to experimentally derive the limitation of each system configurations to serve user requests under 7 seconds on average.

In Table 1, we show the maximum number of concurrent users that each system can support with average user response time under 7 seconds. In Figure 16, the X-axis indicates the number of concurrent user requests and the Y-axis indicates the network latency between the users and the application server. We tested the limitation of each system by increasing the number of concurrent user requests and network latency until the average user response time is above 7 seconds. We then plot the number of concurrent user requests and network latency as the limitation for the system in terms of scalability.

As shown in Figure 16, when the system has no dynamic content caching (the left most line), its scalability is very limited. On the other hand, when we deploy CachePortal software for dynamic content caching, the system can be scaled up to handle more concurrent users as well as higher network latency. For a common network condition in which the round trip time is around 400 ms, the system with 90% hit rate is scaled up to for more than 20 times as pointed out in Figure 16 and in the second row of Table 1.

6. RELATED WORK

Applying caching solutions for Web applications and content distribution has received a lot of attention in the Web and database communities [6, 17, 18, 19, 20, 21, 22]. These provide various solutions to accelerate content delivery, such as middleware level cache/pre-fetch solutions, which lie between application servers and underlying DBMS or file systems. They do not provide automated URL/Query mapping construction and invalidation functionalities.
Figure 13: Effects of Number of Requests on Average User Response Time for Network Latency of (a) 200 ms, (b) 400 ms, and (c) 800 ms

Figure 14: Effects of Number of Requests on Percentage of User Requests Served under 7 Seconds for Network Latency of (a) 200 ms, (b) 400 ms, and (c) 800 ms

Figure 15: Effects of Number of Requests on Percentage of User Requests Resulting in Error Messages for Network Latency of (a) 200 ms, (b) 400 ms, and (c) 800 ms
WebCQ [23] is one of the earliest prototype systems for detecting and delivering information changes on the Web. However, the change detection is limited to ordinary Web pages. Yagoub et al. [24] have proposed caching strategies for data intensive Web sites. Their approach uses materialization to eliminate dynamic generation of pages but does not address the issue of view invalidation when the underlying data is updated. Labrindis and Roussopoulos [25] present an innovative approach to enable dynamic content caching by maintaining static mappings between database contents and Web pages, and therefore this approach requires a modification to underlying Web applications.

Dynamai [3] from Persistence Software is one of the first dynamic caching solution that is available as a product. However, Dynamai relies on proprietary software for both database and application server components. Thus it cannot be easily incorporated into existing e-commerce framework. Jim Challenger et al. [26, 27] at IBM Research have developed a scalable and highly available system for serving dynamic data over the Web. The IBM system was used at Olympics 2000 to post sport event results on the Web in a timely manner. The system provides tools to define fragment pages and their dependency. It utilizes database triggers to generate update events as well as intimately relies on the semantics of the application to map database update events to appropriate Web pages.

SPREAD [28], a system for automated content distribution, is an architecture which uses a hybrid of client validation, server invalidation, and replication to maintain consistency across servers. Note that the work in [28] focuses on static content and describes techniques to synchronize static content, which gets updated periodically, across Web servers.

Yuan et al. [29] evaluated the benefit of edge caching and offloading for dynamic content delivery. In this work, Pet Shop is used for benchmark. Note that Pet Shop comes from Sun’s J2EE reference software PetStore but implemented using ASP.NET. The evaluations verify the benefit of dynamic content caching but at the same time they point out the enabling process is overly complex and even counter-productive. In their prototype, application programs need to be modified and user-specified TTL is used to enforce the consistency in contrast to that CachePortal provides more plug-and-play deployment and features invalidation schemes to ensure cache content freshness.

All these related research activities focus on Web acceleration solution or maintenance of strong consistency for the caches rather than the practical and engineering issues of how to apply and enable dynamic content solutions. None of their approaches, however, evaluate the impact of these caching strategies in a real e-commerce environment such as the one described in this paper.

7. CONCLUDING REMARKS

The framework of enabling dynamic content caching to accelerate Web applications and its applicability truly rely on automated construction of the URL and query mapping. CachePortal develops sniffer and Invalidator to automate and scale up this task. In this paper, we identify the challenges in deploying CachePortal to real world e-commerce J2EE-based Web applications. We recommend practices to transform a typical database-driven J2EE applications to a cache friendly Web application where Web acceleration solutions can be applied. Furthermore, such transformation can be done without modification to the underlying application business logic and without sacrificing functions that are essential to e-commerce. We take the J2EE reference software, the Sun’s Java PetStore, as a case study. After applying the guideline, we are able to cache more than 90% of the content in the PetStore and scale up the Web site more than 20 times.

Future work includes extending our solutions to outside the J2EE setting and PHP-based applications as well as more in-depth study of deploying acceleration solutions for Web applications based on cluster architectures.

Acknowledgments

The authors would like to thank Satish Murthy for useful discussions on this work and acknowledge his contributions in engineering the CachePortal SDK used for experiments in this paper.
8. REFERENCES


