Evaluations of Architectural Designs and Implementation for Database-Driven Web Sites

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

C&C Research Laboratories, NEC USA, Inc.
110 Rio Robles, M/S SJ100, San Jose, CA 95134, USA
Email: {wen,whsiung,oliver,candan,agrawal}@ccrl.sj.nec.com
Tel: 408-943-3008  Fax: 408-943-3099

Abstract

Response time is a key differentiation point among electronic commerce (e-commerce) applications. For many e-commerce applications, Web pages are created dynamically based on the current business state, stored in database systems. The architecture of database-driven e-commerce Web sites are more complex than that of typical Web sites. It requires integration of Web servers, application servers, and back-end database systems as well as synchronization of multiple databases if caches are used for acceleration of content delivery. In this paper, we analyze the factors that impact the performance and scalability of a database-driven Web site. We experimentally test (1) the performance metrics of database update, query, and synchronization; (2) trigger overhead; and (3) application server and database connection overhead and constraints. We describe several architectural design approaches for database-driven Web sites and present experimental results on their performance under various conditions, including varying request rates, update-to-request ratio, cache hit ratio, number of regional application server/data cache suites deployed, and database size. We also discuss how to handle Web page requests that involve SSL, cookies, and fragment pages (i.e. by frame or JSP).

Keywords: Database cache, electronic commerce, dynamic content caching, Web acceleration, application server, i-cache, cookie, SSL, fragment page

1 Introduction

Forrester Research Inc. [1] expects that by 2002 over 47 million people will purchase goods and services online in the United States alone and the U.S. Internet commerce will grow to $327 billion by that same year. The fast growing demand for e-commerce brings a unique set of business and technical challenges in building high performance e-commerce Web sites.

In business terms, the brand name of an e-commerce site is highly correlated with the experience users receive. Response time is a key point of differentiation among e-commerce Web sites. Snafu and slow-downs at major Web sites during special events or peak times demonstrate the difficulty of scaling up e-commerce sites. The need for accounting for users’ quality perception in designing Web servers for e-commerce systems has

†This work was performed when the author visited NEC USA, CCRL.
been highlighted by [2]. Such slow response times and down times can be devastating for e-commerce sites as indicated in a recent study by Zona Research[3]. The study shows that only 2% of users will leave a Web site (i.e. abandonment rate) if the download time is less than 7 seconds. However, the abandonment rate quickly jumps to 30% if the download time is around 8 seconds. The abandonment rate goes up to 70% when the download time is around 12 seconds. This study clearly establishes the importance of fast response times in retaining customers of an e-commerce Web site.

In technical terms, ensuring fast delivery of fresh dynamic content and engineering e-commerce Web sites that can scale during special events and peak times put heavy pressures on IT staff. These tasks are further complicated by complexity of e-commerce applications. 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 deploy cache servers, Web servers, application servers, and database systems at the backend. The roles played by these servers are as follows:

1. A database management system (DBMS) to store, maintain, and retrieve all necessary data and information to model a business.

2. An application server (AS) that incorporates all the necessary rules and business logic to interpret the data and information stored in the database. AS receives user requests for HTML pages and depending upon the nature of a request may need to access the DBMS to generate the dynamic components of the HTML page.

3. A Web server (WS) which receives user requests and delivers the dynamically generated Web pages.

4. Cache servers (edge caches or frontend caches) to accelerate content delivery.

One possible way to scale up database-driven e-commerce sites 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 two advantages: serving users via a nearby cache closer to the users and reducing the traffic to the Web sites. Many content delivery network (CDN) vendors [4, 5] provide Web acceleration services. The study in [6] shows that CDN indeed has significant performance impact. However, for many e-commerce applications, HTML pages are created dynamically based on the current state of a business, such as product prices and inventory, rather than static information. As a result, the time to live (TTL) for these dynamic pages can not be estimated in advance. As a result, content delivery by most CDNs are limited to handling fairly static pages and streaming media rather than the full spectrum of dynamic content discussed in this paper.

Evaluating the traffic patterns and needs, selecting proper architectures, and integrating various servers to
build a large scale high performance e-commerce Web site are complex. Furthermore, it is very challenging to coordinate synchronization between

- pages in the caches distributed in a large content delivery network and databases;
- content in the master database and multiple database replications;
- data caches used by application servers at remote data centers and the master database; and
- data caches at remote data center and the master database.

Such issues related to caching of dynamic data have received significant attention recently [7, 8]. Dynamai[9] from Persistence Software is one of the first dynamic caching solutions 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 in an existing e-commerce framework. Challenger et al. [10, 11, 12] at IBM Research have developed a scalable and highly available system for serving dynamic data over the Web. In fact, the IBM system was used at Olympics 2000 to post sport event results on the Web in timely manner. This system utilizes database triggers for generating update events as well as intimately relies on the semantics of the application to map database update events to appropriate Web pages.

As the popularity of content delivery network (CDN) services [4, 5] increased in recent years, many database and application server vendors start to view Web acceleration through caching as a key functionality of their software. Examples include Oracle 9i [13] in an Oracle’s major announcements in September 2000. Oracle 9i features a suite of application server, Web server, and data cache for deployment at data centers for acceleration of dynamic content delivery. With all of these newly available software for dynamic content caching and delivery acceleration, it is more flexible to architect a “distributed” Web site, which may actually locate in multiple networks and geographical regions.

It is required that coherence of Web pages delivered from caches and database content that these pages represent must be assured. That is, if the database content changes, then all pages in cache servers that are generated using this data need to be invalidated or refreshed within a reasonable time period. Note that the assurance of the freshness of cached content can only be based on a best-effort approach since the computers are connected through the Internet.

In this paper, we present approaches to accelerating dynamic content delivery in the context of general e-commerce setting. We investigate various e-commerce system architectures based on databases, data caches, edge caches, Web servers, and application servers. We also report on experiments that have been conducted to evaluate relative performance of different alternatives.
The rest of this paper is organized as follows. In Section 2, we give an overview of the system architecture of a typical database-driven e-commerce Web site and identify major factors that impact Web site performance. In Section 3, we describe several approaches to e-commerce Web site acceleration approaches based either database replication or data cache. In Section 4, we present a loosely-coupled approach, CachePortal, that we have developed at NEC for e-commerce Web site acceleration as well as comparisons with other approaches. We also discuss issues related to caching, including secured Web sites using SSL, personalized using cookie, and how to construct a "cache friendly" Web site. In Section 5, we summarize experimental results and compare all mentioned approaches. In Section 6, we review existing work. Finally we give our concluding remarks and future work in Section 7.

2 Factors That Impact Web Site Performance

CDNs have gained a significant prominence in the context of Internet technologies. They aim to utilize Web-based network elements (or servers) to achieve efficient delivery of Web content. CDNs are based on either few large data centers[5] or a massive number of servers[4]. Although these service providers use varying technologies, the study in [6] shows that static content caching and multi-server content distribution indeed have significant performance impact. [14] provides concrete evidence that shows that a distributed architecture of coordinated caches perform consistently better in terms of hit ratio, response time, freshness, and load balancing.

CDNs do improve the network delay; however, In systems which deliver dynamically generated content, the server delay caused by the application server, the database, or connection between them could be more substantial than the network delay. In this section, we identify bottlenecks that limit the scalability of Web sites which are based on caches, Web servers, applications, and databases.

2.1 Experimental Evaluation of Current E-Commerce Web Site Architecture

In order to identify the bottlenecks of e-commerce Web site architectures, we considered the following four different configurations:

**Configuration 1:** Users in San Jose, CA send requests to Princeton, NJ to retrieve static content page of size 9K bytes. Pages are pre-generated and served from a Web server. No application server or database is involved in this configuration.

**Configuration 2:** Users in San Jose, CA send requests to Princeton, NJ to retrieve dynamic content pages of size 9K bytes. Pages are generated dynamically by CGI calls to the Web server and data used to generate these pages are located in local file system or memory. After these pages are generated, they are served
from the Web server to the users. In this configuration, no database is involved in this configuration.

**Configuration 3:** Users in San Jose, CA send requests to Princeton, NJ to retrieve dynamic content pages of size 9K bytes. Pages are generated dynamically by an application server (servlets) and data used to generate these pages are located in local file system or memory. After these pages are generated, they are served from a Web server to the users. In this configuration, the Web server and the application server are located on the same machine. No database is involved in this configuration.

**Configuration 4:** Users in Princeton, NJ send requests to San Jose, CA to retrieve dynamic content pages of size 9K bytes. Pages are generated dynamically by an application server and data used to generate these pages are accessed from an underlying database. After these pages are generated, they are served from a Web server to the users. In this configuration, the Web server, the application server, and the database are located on the same machine.

**Configuration 5:** The same as the configuration 4 except that the Web server and the application server are on the same machine while the database is located on a separate machine.

No cache is used in these experiments. All of the machines where the Web servers and applications located are Pentium III 700Mhz single CPU PCs running Redhat Linux 6.2 with 1G memory. User requests are generated using a cluster of high powered machines. Oracle 8i is used as the DBMS and BEA WebLogic 5.1 is used as the Web/Application Server.

Figure 1 shows the experimental results of configurations 1, 2, and 3. The traffic generated from San Jose to Princeton continues for 3 seconds and the experiments were repeated twice during different times of days. The response time is measured and the averages are used for plotting. As the figure shows, the Web server serving static content can scale up very well and the response time is excellent even with more than 2,000 requests per second. For the Web site serving dynamic content using the Web server and the application server (without access to a database), the performance is less scalable but it can still provide sub-second response time for traffic up to 2,000 requests per second for response time less than 2 seconds.

Now let us consider the results (shown in Figure 2) from the two database-driven e-commerce sites (configurations 4 and 5). It is clear that both configurations are much less scalable compared with other configurations. With traffic around 200 requests per second, both architectures already yield very poor response times. Based on Figure 1 we know that Web servers and application servers are not the bottleneck in the e-commerce site performance. Thus, the experimental results suggest that the bottleneck of a database-driven E-commerce Web site is the DBMS or setting-up a connection to the DBMS. Thus, based on the experimental results presented here, we believe that the appropriate approach to Web acceleration for database-driven e-commerce applications
Figure 1: Performance Evaluation on Web Servers and Application Servers (No DBMS is Involved)

Figure 2: Performance Evaluation on Web Servers and Application Servers (DBMS is Involved)
should be

- to *reduce the load on the database*; or
- to *reduce the frequency of DBMS accesses*.

### 2.2 Cost of Updates Versus Queries in Databases

We also conducted experiments to evaluate the cost of updates on a database. Updates are generally more expensive than queries. Furthermore, updates may require data synchronization between a master database and replicas (or multiple copies of databases). In Figure 3(a), we show that the cost of updates is in general twice expensive than the cost of queries in databases in various sizes. Of course, the cost of updates varies by many data and DBMS related factors, including the underlying indexing schemes, database sizes, transaction management techniques. It is generally very costly to keep multiple databases synchronized and up-to-date in the presence of frequent update activity at the databases [15].

### 2.3 Scalability and Limitations of Invalidation Using Triggers

In order to enable caching of dynamic content, we must also develop an invalidation mechanism so that the cached content can be removed when the associated data undergoes changes at the DBMS. An obvious solution to cache invalidation is to embed into the database update sensitive triggers, which generate invalidation messages when certain changes to the underlying data occurs. However, this approach could put heavy trigger management burden on the database, especially when a large number of triggers needed to be defined, checked, and dropped according to the dynamically generated Web pages in cache servers. We have tested the scalability of invalidation using triggers. The experiment setup is as follows: we construct six tables of 100,000 rows each and the tables have 0, 200, 400, 600, 800, 1,000 triggers respectively. We then test the number of updates can be executed on these tables in 60 seconds. Each update would fire only one trigger. As we can see in the figure, almost 800 updates can be executed on the table which has no trigger but only 65 updates can be executed on the table with 1,000 triggers. Such a big difference is due to the overhead of trigger condition checking and currently there is no intelligent indexing scheme built in most commercial DBMSs for efficient trigger condition checking. Note that all triggers on a table need to be checked for a single tuple being inserted, updated, or deleted. Trigger management is a very expensive operation.

Figure 3(c) shows experimental results on evaluating the overhead of using triggers for database response time for the requests from application servers. In this experiment, we constructed six tables, where each table has 1,000,000 rows of tuples and 50, 100, 500, 1,000, 1,500, and 2,000 triggers respectively. We then issued various
Figure 3: (a) Cost of Update Operation versus Query Operation in Databases of Different Sizes; (b) Cost of Invalidation Using Triggers in Term of Update Operations; (c) Overhead of Using Triggers
numbers of queries to the database. Among these queries, 10% of them are update operations and each update changes 5 rows of data. In the figures, the X-axis represents numbers of queries submitted (including update operations). We observe that the average response time for all queries increases as the number of triggers defined and the number of queries issued increase. Note that one trigger needs to be defined for each page cached in the cache server. As the figure indicates, when there are 2000 triggers defined and 300 requests (including 30 update operations), the average response time is close to 2000 seconds rather than 0.003 second on average for a table without any trigger defined. Please note that if there are 30 updates per second, each update changes 5 rows of data, and there are 2,000 triggers defined for the table, there will be 0.3M (i.e. $30 \times 5 \times 2,000$) trigger condition checking per second. The experiment and analysis clearly show that invalidation based on triggers can not scale.

Another significant limitation of using triggers for invalidation is that, in most of commercially available DBMSs including Oracle and DB2, triggers can be applied to queries involving only a single table (although many prototype systems, such as TriGS Active Object-Oriented Database System[16], have been developed for research purposes). For queries that involve more than one table, the content change monitoring function needs to be done through view definition and view maintenance; which would be even more expensive than triggers. This further restricts or, at least, complicates the usage of using triggers for the invalidation purpose since most database access requests issued by the application server are queries over multiple tables. Many research prototypes and caching solution vendors, such as Xcache [17] and SpiderCache [18], utilize triggers for invalidation of pages in the caches or Web servers. Although a lot of research have been done on view maintenance[19, 20] and constructions of composite events, such as [16, 21], we believe that it is an option limited to building and customizing a a Web site which has a limited number of distinct pages, such as the IBM Web site for Olympic Games[10, 11, 12]. In our system[22, 23], we intend to support automated constructions of the mapping between Web pages and queries that retrieve the database contents to generate such pages.

3 Approaches to Acceleration for E-Commerce Web Sites

Several different approaches can be used to accelerate content delivery from database-driven e-commerce sites. A common thread among all these approaches is to employ some form of data redundancy so that content can be delivered via multiple paths through the network thus eliminating bottlenecks. One way data redundancy can be realized is by caching dynamically generated HTML pages in the network cache components (e.g., front-end cache, proxy cache, and edge cache). The other approach is to use database replication in conjunction with load balancing so that the user requests can be served from multiple databases. In the following, we describe various alternatives as well as the outcome of our experimental evaluation to explore the feasibility of each of these approaches.
3.1 Redundant Server/Database Approach

An alternative to caching is shown in Figure 4(a) where there are multiple Web/application server pairs to handle user requests. The WS/AS pairs are load balanced using a traffic balancer, such as Cisco LocalDirector, which directs the user requests appropriately to a WS/AS pair so as to keep the load on each pair balanced. This configuration enables a Web site to partition its load between multiple Web servers and application servers, and hence it can achieve higher scalability without the use of caches. Note, however, that since pages delivered by e-commerce sites are database dependent, replicating only the Web servers and application servers is not necessarily enough for scaling-up the entire architecture. We need to make sure that the underlying database does not become a bottleneck.

In order to eliminate the database bottleneck, Figure 4(b) depicts a configuration in which database servers are also replicated along with the Web and Application server pairs. In this setup Cisco’s LocalDirector is also used to direct user traffic for balancing the load among multiple components. In our experimental set-up, the Web server farm behind the load distributor consists of four PCs running Linux and Apache web servers. In addition to the Web servers, there are four database management systems, each serving queries from a single Web server. Note that this architecture has the advantage of being very simple. However, it has two major shortcomings: (1) since it does not allow caching of dynamically generated content, it still requires redundant computation when users make duplicate requests; and (2) it requires costly database synchronization.

In a later section, we will show that our experimental results (and intuitions) suggest that the architecture in Figure 4(a) is suitable for a Web site with light weight database transaction requirements, since the number of connections that can be handled by the application servers could be a bottleneck for this type of traffic pattern.\footnote{Note that many application servers have a pre-set number of connections to databases for performance purposes.} On the other hand, the architecture in Figure 4(b) is more suitable for a Web site whose traffic pattern is a large
number requests and each request involves expensive database queries since the number of concurrent accesses to databases could be a bottleneck. However, this architecture will yield slow response times if most of database accesses are expensive update transactions since the update propagation will be costly.

In order to estimate the cost of synchronization, consider the following. Assume that all of the updates are first received on a master database and then broadcasted to all other (slave) databases. Since the e-commerce site needs to maintain consistency across all copies of databases, an update will commit only when a transaction is committed across all databases. To prevent access to stale data, between the time at which the slave databases receive the update information until they receive the final commit acknowledgment, assume that the updated data item (tuple, page, or table, depending on database implementation) may be locked. Therefore, queries that rely on this particular data item cannot be processed while the synchronization is not complete; i.e., until all databases finish their updates and inform the master and master informs the slaves to remove locks on the data item. Alternatively, if synchronization is not essential, we can use asynchronous replication. Candan et al. have conducted a complete study of various architectures described above. The experimental results can be found in [22].

### 3.2 Redundant Application Server/Data Cache Approach

Wide-area database replication allows database copies to be distributed across the network. Basically, the goal of this approach is to offset the high cost of replica synchronization by moving data closer to the users (similar to caching in which data is moved closer to the users reducing network latency). However, as shown in Figure 5, this requires a complete e-commerce web site suite (i.e. Web server, application server, and DBMS) to be distributed

---

2 Also note that similarly many DBMSs have a pre-set number of connections to databases for the performance purpose.
along with the database replicas. The updates to the database are still handled using a master/slave database configuration and therefore all updates are handled via the master DBMS at the origin site. The scheme for directing user requests to the closest server is the same as what typical CDNs are using.

In order to distinguish between the asymmetric functionality of master and slave DBMSs, we refer the remote DBMS copies as data cache or DBCache since they are basically read-only copies and cannot be updated directly. DBCache can be a light weight DBMS since no transaction management system needs to be deployed since no update operation will be executed at these suites at the remote locations; typically data centers. Note that the DBCache may cache only a subset of the tables in the master databases. Either a pull or a push method can be used to keep DBCache synchronized with the master DBMS. For example, Oracle 9i i-cache uses a pull method in which the DBCache periodically synchronizes with the master DBMS. Note that Oracle 9i supports two synchronization methods: incremental refresh or complete refresh. Typical production environment will employ a hybrid approach in which a complete refresh is done at a coarser granularity (e.g., once in a day) and incremental refresh is done at a finer granularity (e.g., once every hour).

In this paper, we assume that the freshness of cached pages needs to be assured and asynchronous update propagation is used. Assume that the traffic to a Web site based on the architecture in Figure 4(a) is 200 requests per second and the update request ratio is 20%. Let us also assume that the load is equally distributed among all regional application server/DBCache suites. Thus the load on each Web server/application server is 50 requests per second. The master database receives 40 update requests per second from 4 different application servers and the master database needs to propagate 40 updates to four DBCaches. For each DBCache, it needs to handle 50 non-update requests from the application server and 40 updates from the master database. As we can see based on this analysis, the user response time at this Web site depends on the number of requests per second, the update request ratio, and the number of suites deployed.

In Figure 6(a), we show experimental results on a Web site which deploy only one suite. BEA WebLogic [24] is used for both the Web server and the application server and Oracle 9i [13] the DBMS. The Web/Application server and the database are located on two separate machines in San Jose, CA, USA. In the configurations of 1 suit, 2 suits, 4 suits, and 10 suits, 3 machines, 5 machines, 9 machines, and 21 machines are deployed respectively. The database contains 7 tables and each table contains 40,000 tuples. The queries involve two tables and one join operation. The updates involve only one table and each update modifies the values in five rows. The sizes of result page for all requests are 4.7K. The user requests are initiated from Tokyo, Japan.

In the first experiment, we vary the request rate and update request ratio. We see that the response time increases as the request rate increase. We also see that the response time increases as the update request rate increases for the same request rate. Since this Web site only deploys one suite, when the update request ratio
increases, the load on the DBCache increases (although the total number of database accesses are the same). This is because update operations is in general more expensive than queries (as shown by our experimental results in Figure 3(a)). In the second experiment, we vary the number of suites deployed for this Web site. Figure 6(b) illustrates the experimental results. We can see that the user response time decreases as the number of suites deployed increases. We can also see that the Web site which deploys more suites scale better than that deploys fewer suites. The Web site that deploys more suits does have the advantage of supporting more connections at the application servers and fast delivery time. Note that the maximum number of connections from the application to the database is set to 20 as defaulted in the Web/Application Server we use in this experiment. In addition, the configuration with more suits has more computation resource to process queries. In the next section, we present the approach that is based on dynamic content caching and invalidation. And we will compare these mentioned approaches later in Section 5.

4 CachePortal Technology for Accelerating Content Delivery

In this section we describe an architecture for accelerating delivery of e-commerce content over wide-area networks. The architecture and underlying technology, referred to as CachePortal, enables caching of dynamic data over wide-area networks. CachePortal is an open architecture in the sense that it can apply to any combination of the following components:

- cache servers that feature invalidation APIs or comply with Web Cache Invalidation Protocol (WCIP)[25];
- database systems that support JDBC; and
- J2EE complied Web and Application Servers.
4.1 System Architecture

In Figure 7, we show the architecture and data flows of a database-driven e-commerce Web site, which employs the CachePortal technology. Note that this architecture is very similar to that of a typical e-commerce web site, except for the two components introduced by CachePortal: Sniffer and Invalidator enable dynamic content caching and ensure the freshness of dynamic content in cache servers.

The data flow, as shown in Figure 7, between the caches, the Web server, the application server, and the database is as follows:

- The Web server communicates with the application server using URL strings and cookie information, which is usually used to identify the user identity so that a personalized Web page can be generated.
- The application server communicates with the database using queries.
- The database changes can be monitored by the tuples and query statements recorded in the database log. However, the database itself, which knows how and what data changes, does not know how the data is used to generate dynamic Web pages. In addition, the database does not know which dynamic content, identified by URLs, is impacted by these database changes.

Our proposed technology enables dynamic content caching by deriving the relationships between cached pages and database access via a sniffer; and intelligently monitoring database changes to ”eject” related pages from caches via an invalidator. Note that knowledge about dynamic content is distributed across three different servers:
the Web server, the application server, and the database management server. Consequently, it is not straightforward to create a mapping between the data and the corresponding Web pages automatically in contrast to the other approaches [9, 10, 11, 12], which assume such mappings are provided by system designers. The detailed descriptions on automated construction of such a URL/Query map are given in [22].

4.2 Handling Secured Communication Protocol and Cookies

Many e-commerce transactions and even query requests are transmitted through some secured communication network protocols, such as SSL (Secure Sockets Layer)\(^3\). This protocol is applied to the communication between users and a secured Web server. When the HTTP request messages sent to the application server, the HTTP request messages are not encrypted. Since the sniffer listens to the messages between the Web server and the application server and the messages between the application server and the database system, whether or not the Web server is a secured server has not impact to the construction of URL/query mapping.

As to handling cookie information, the sniffer treats cookie information as a part of the URL. Therefore, for two Web page requests of the same URL but different cookie values, CachePortal views such two pages differently. Note that cache servers must also be cookie enabled.

4.3 Handling Fragmented Pages

Frames are frequently used in most commercial Web sites for the flexibility it provides in page formatting and layout, as well as for convenient navigation for users. Below, we show an example of an index page, index.html, which consists of six fragment pages grouped by a frame.

```html
<HTML>
<TITLE>
Personal Shopper Home Page
</TITLE>
<FRAMESET ROWS="10%,10%,80%">
  <FRAME NAME="top" SRC="/PS/personal_greeting.html" SCROLLING="no">
  <FRAME NAME="top" SRC="/PS/target_ads.html" SCROLLING="no">
  <FRAMESET COLS="30%,30%,30%,10%">
    <FRAME NAME="left" SRC="/PS/left.html" SCROLLING="auto">
    <FRAME NAME="left" SRC="/PS/middle.html" SCROLLING="auto">
    <FRAME NAME="left" SRC="/PS/right.html" SCROLLING="auto">
    <FRAME NAME="right" SRC="/PS/topsellers.html" SCROLLING="auto">
  </FRAMESET>
</FRAMESET>
</HTML>
```

After a browser receives this page, it parses it and then requests six additional pages. Note that although this

\(^3\)Available at http://home.netscape.com/eng/ssl3/draft302.txt.
composite page is displayed to the user as a single and personalized Web page (which may not be cachable), six out of seven pages that the browser needs to fetch are cachable. In this example, topsellers.html can be updated hourly and left.html, middle.html, right.html, and index.html are static content unless the application programs change. target_ads.html is customized through cookie and could be handled and delivered by a typical cookie enable cache server. Only the page personal_greeting.html needs to be generated on demand and it is also cachable although the reusability could be low. In this example, we show that the popular practice of using frame for page formatting and layout is still covered by CachePortal sniffer. The sniffer basically treats each requests independently. As matter of fact, composing pages using frame makes a Web site more cache friendly since it allows more caching opportunities by producing pages in a finer granularity.

4.4 Invalidator

The invalidator listens to the updates in the database and using the URL/Query map, identifies pages to be invalidated and notifies the relevant caches about the staleness of the cached page. For this purpose, it interprets the query instances in the URL/Query map. The invalidator consists of three subcomponents:

- The first component periodically examines the database log to extract the updates since the previous extraction.
- The second component analyzes the updates (which are in terms of inserts, deletes, and modifications of tuples to specific tables) and identifies the queries that are impacted. These queries are registered in the URL/Query map.
- The third component identifies the URLs that are invalidated due to impacted queries and generates cache invalidation messages which are destined to the network-wide caches via the API provided by the CDNs.

We now show the invalidation process using two examples.

Example 4.1 Let us assume that we have an e-commerce application which uses a database that contains two tables, Car(maker, model, price) and Mileage(model, EPA). Let us also assume that the following query, Query1, has been issued to produce a Web page, say URL1:

```
select maker, model, price from Car where maker = "TOYOTA";
```

If we observe that a new tuple (Toyota, AVALON, 25,000) is inserted into (or deleted from) the table Car in the database, we would know that the results of Query1 is impacted, and consequently URL1 needs to be invalidated.
Example 4.2 In the above example, since the query contains only one table, it is possible to evaluate the impact without any additional information. On the other hand, if we assume the following query Query2 which has been issued to produce URL2,

```sql
select Car.maker, Car.model, Car.price, Mileage.EPA
from Car, Mileage
where Car.maker = "TOYOTA" and
  Car.model = Mileage.model;
```

Since Query2 needs to access two tables, we can check whether a new tuple inserted into Car does not satisfy the condition in Query2 without any additional information. But if the new tuple satisfies the condition, then we cannot check whether or not the result is impacted until we check the rest of the condition, which includes the table Mileage. That is, we need to check whether or not that the condition `Car.model = Mileage.model` can be satisfied. To check this condition, we can issue the following polling query, PollQuery, to the database:

```sql
select Mileage.model, Mileage.EPA from Mileage where "AVALON" = Mileage.model;
```

If there is a result for PollQuery, we know that the new insert operation has an impact on the query result of Query2 and consequently URL2 needs to be invalidated.

Note that the process of analyzing the impact of updates to previous queries requires an intelligent query type classification and identification. If we observe that the tuple (Toyota, AVALON, 25,000) is updated and its new value is (Toyota, AVALON, 27,000), one naive way to find the impacted pages is to check all queries that access to the table Car, the same way as how triggers are checked in most DBMSs. However, this could be very expensive. A more intelligent way, as implemented in the CachePortal invalidator, is to indexing queries based on query type classification.

Let us consider the following five queries, Query3, Query4, Query5, Query6, and Query7, whose results are used to produce URL3, URL4, URL5, URL6, and URL7 respectively.

Query 3:
```
select maker, model, price
from Car
where maker = "TOYOTA";
```

Query 4:
```
select maker, model
from Car
where maker = "TOYOTA";
```

Query 5:
```
select maker, model
from Car
where price > 30,000;
```
We classify the queries by their query types based on the tables and the attributes the queries access and operate on. In this example, the queries that access Car.price are Query3 and Query5 and the queries that access Car.model are Query3, Query4, Query5, and Query6.

Now, if we observe that the tuple (Toyota, AVALON, 25,000) is updated to (Toyota, AVALON, 27,000), the invalidator will only check if the query results of Query3 and Query5 need to be checked for possible invalidation, rather than checking all queries. We can further classify queries in a finer granularity, say, both Car.price and the operation on Car.price, such as > 30,000. If such a finer classification exists, the invalidator will only check if the query result of Query3 needs to be invalidated.

If we observe that the tuple (Toyota, AVALON, 27,000) is updated again and the new value is now (Toyota, AVALON XLS, 27,000), the invalidator will only check if the query results of Query3, Query4, Query5, and Query6 need to be invalidated.

With the query type classification, the impact checking by the invalidator in CachePortal is much more efficient than implemented using database triggers.

4.5 Experiments on Invalidation Response Time

In this section, we discuss the response time of an invalidation cycle. One invalidation cycle includes three phases: scanning the database log to find database content changes; preparing and issuing necessary polling queries to retrieve additional necessary information; and performance invalidation check within the invalidator. We have conducted experiments to measure the response time of our invalidation scheme. We are concerning the tasks from checking the database log to issuing deletion messages to caches to invalidate cached pages which are impacted.

Note that we do not discuss issues related to invalidation message protocols or schemes between the invalidator and caches.

In this experiment, we define the invalidation cycle as 5 seconds. That is, CachePortal accesses database log every 5 seconds. And, during this period, a number of requests may arrive. The X-axis shows that there are 200,
500, 1000, and 2000 queries captured by the sniffer in a cycle. Among these queries, we assume that 0.5% of accesses result in updates and each update will cause invalidation on 4 cached pages. Thus, in the case in which there are 2,000 queries captured, the sniffer needs to register these queries. At the same time, the invalidator scans the database log, perform polling queries, and check if any page should be invalidated. In the most demanding case, in which 2,000 unique queries are issued (i.e. 400 requests per second), the CachePortal can complete sending out invalidation messages within 3 seconds. We believe it is satisfactory compared with the invalidation response of dynamic content caching and invalidation systems based on triggers.

Please note that 400 requests to the database server per second is extremely high traffic to a typical Web site. If the hit ratio of front-end caches or edge caches is 80%, 400 requests to the database server per second means 2,000 requests to the Web site.

4.6 Evaluating Effectiveness of Web Acceleration by CachePortal

We now report the results of our experimental evaluation of the CachePortal technology under different settings for Web acceleration. We test the response time improvement on an e-commerce Web site in the area of housing. The Web site is implemented using Apache (used as a front-end cache), BEA WebLogic Application Server, and Oracle DBMS and CachePortal is deployed for Web acceleration. Two machines are used: one for the front-end cache and the other machine for the Web server, application server, and the database. The database used in this experiment contains 7 tables and more than 1M records. The update rate is 1 per second and each dynamic content page request results in a query with one join operation to the database. We considered the following two architectural configurations:

Case 1: the Web server and the application server are located on the same machine. and database is
located on the separate machine. No cache is used.

**Case 2**: the Web server and the application server are located on the same machine. The database is located on a separate machine. *CachePortal* technology is applied and a front-end cache, a cache close to servers, is used. The freshness of cached content is guaranteed.

The experimental results in Figure 9 indicate that the response time in case 1 increases sharply when the number of HTTP requests increases. On the other hand, the response time in case 2 remains relatively flat for the settings where the hit ratios are 60%, 80%, and 100% respectively. By analyzing response times and server latencies in each case we found that the user response time delay is mainly caused by the server latency rather than the network delay. This means that the requests do arrive at the server without much network delay, but they are queued at the application server waiting for query results from the database.

Another meaningful measure is the percentage of the requests that are served under 7 seconds, where the user abandonment rate is less than 2 percent as shown in the study by Zona Research[3]. In Figure 10, we see that the system architectures (upper three plots) with *CachePortal* technology provide good response time most of the time whereas the system architecture without the *CachePortal* technology (the lower plot) has poor performance.

In the previous experiment, the updates are issued to the database directly. We have also conducted experiments by varying the update request ratio. Note that, update request results are not cachable. Thus, if the cache hit rate is 60% and the update request rate is 50%, only 30% (i.e. (1-0.5)*0.6) of the total requests will be served from the caches. Therefore, as the update request ratio increases, the response time increases accordingly. When the update request rate reaches 100%, no page is delivered from the cache. Thus, the three plots of response time merge at the point where the update request rate is 100%. In Figure 11, we see the benefit of Web acceleration by caching decreases as the update request ratio increases. When the update request ratio is below 10%, the user response time is below 2 seconds.

An update request ratio of 10% for overall HTTP requests is extremely high. For a reasonably successful e-commerce site which has a global presence can expect that every 20 Web site visitors will result in actual sales (and therefore will involve updates to the underlying DBMS). Based on our informal survey of existing e-commerce Web sites, it requires approximately 10 page views before completing a purchase transaction even if the end-user knows exactly the product he/she wishes to purchase and each page is composed of 5 fragment pages on average. In this case, the update request ratio is $1/(20*10*5) = 0.1\%$. This analysis suggests that in most times and most Web sites, caching is useful to Web acceleration.
Figure 9: Evaluation on Average User Response Time

Figure 10: Evaluation on Percentage of User Response Time Under 7 Seconds
5 Summary and Discussions of the Results

5.1 Caching is Effective and Cost Efficient

In this subsection, we compare the performance of a Web site that deploys data caches (described in Section 3.2) and the performance of a Web site that deploys data caches described in this section. For the purpose of easy discussion, we overlapped Figures 6(b) and 11 as Figure 12. Figure 16 shows detailed plots for response time where update request rates are under 10%. We see that in most cases (i.e. reasonable update request ratio and cache hit) a Web site using 3 machines (i.e. cache server, Web server/Application server, and database server) with CachePortal (shown in three dotted lines) can outperform a Web site using 21 machines by deploying 10 suites. On the other hand, the architecture that deploy multiple suites seem to scale up in a wider range of setting, especially when most of requests are unique and update operations. This is understandable since caching is less effective when most requests are non-cachable or non-repeating.

5.2 Construction of Cache-Friendly Web Sites

In-order to benefit from caching, careful and intelligent design of cache friendly Web sites is essential. In Figure 14(a), we show a non-cache friendly Web site design. Using this interface, users can issue unlimited number of query types to the application server and the hit ratio will be very low as well since the probability that two users issue the same query would be extremely low. On the other hand, the Web site design shown in Figure 14(b) is a
Figure 12: Performance Comparisons between Data Cache Approaches with and without Deployment of CachePortal

Figure 13: Detailed Performance Comparisons between Data Cache Approaches with and without Deployment of CachePortal
Figure 14: (a) A "Non-Cache Friendly" Web Site (b) A "Non-Cache Friendly" Web Site

A cache friendly Web site interface since there will be a limited number of query types that will be generated:

\[ 3 \times 3 \times \text{number of distinct value for location} \times 9. \]

The Web site can generate all possible pages in advance and store them in the cache server while deploying CachePortal technology to invalidate pages in the cache that are impacted by database content changes. The most desirable cache friendly Web site design would be using the interface in Figure 14(b) as the default query interface for most users while still supporting the interface in Figure 14(a) for advanced query and search.

5.3 Integration of Data Cache Scheme and CachePortal

Another possible alternative configuration is to integrate data cache configuration with CachePortal as shown in Figure 15. In this configuration (four suites are deployed), CachePortal monitors the requests to four application servers and the data changes on one database system (i.e. the master database). In this configuration, CachePortal simply reduces the overall traffic to the Web site in all four data cache configurations. We can see that the response time for the configuration deploying 10 suits is the fastest one since it has more computation resource and connection points to handle requests that are not served by the cache. Deployment of cache servers and CachePortal technology for invalidation is both effective and cost efficient for most traffic conditions and e-
commerce applications.

6 Related Work

In addition to the work mentioned in Section 1, related works include [26, 27], where authors propose a diffusion-based caching protocol that achieves load-balancing, [28] which uses meta-information in the cache-hierarchy to improve the hit ratio of the caches, [29] which evaluates the performance of traditional cache hierarchies and provides design principles for scalable cache systems, and [30] which highlights the fact that static client-to-server assignment may not perform well compared to dynamic server assignment or selection.

SPREAD[31], 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 [31] focuses on static content and describes techniques to synchronize static content, which gets updated periodically, across Web servers. Therefore, in a sense, the invalidation and validation messages travels horizontally across Web servers. Other works which study the effects of invalidation on caching performance are [32, 33]. Consequently, there has been various cache consistency protocol proposals which rely heavily on invalidation [31, 34, 25]. In our work, however, we concentrate on the updates of data in databases, which are by design not visible to the Web servers. Therefore, we introduce a vertical invalidation concept, where invalidation messages travel from database servers and Web servers to the caches.
Figure 16: Performance Comparisons between Eight Configurations of Web Sites Deploying Data Cache and/or CachePortal
7 Concluding Remarks

In this paper, we focus on the issue of caching dynamic content in the context of database-driven e-commerce sites. We examined a range of solutions that already exist as well as potential approaches which can be used to develop a solution to the problem of dynamic content caching. We evaluate the architectural designs of many approaches by varying the request rate, update request ratio, and cache hit ratio.

We see that in general caching, along with appropriate invalidation technologies, is a good solution to Web acceleration and scale up the capacity of a database-driven Web site. We feel that it is important to evaluate the traffic patterns and needs first and then select proper architectural designs for integration of available software on the market to build a scalable high performance e-commerce Web site. To utilize caching and many of the technologies described in this paper, careful and intelligent design of cache friendly Web sites is also essential to the scalability.

References


Author Bibliography

Wen-Syan Li is a Senior Research Staff Member at Computers & Communications Research Laboratories (CCRL), NEC USA Inc. He received his Ph.D. in Computer Science from Northwestern University in December 1995. He also holds an MBA degree. His main research interests include content delivery network, multimedia/hypermedia/document databases, WWW, E-Commerce, and information retrieval. He is leading CachePortal project at NEC USA Venture Development Center and Content Awareness Network project at NEC CCRL in San Jose. Wen-Syan is the recipient of the first NEC USA Achievement Award for his contributions in technology innovation.

Wang-Pin Hsiung is a Research Associate at Computers & Communications Research Laboratories(CCRL), NEC USA Inc. He received his masters degree in Computer Science from Stanford University in 1999 and BSE from Arizona State University. His research interests are in the areas of computer network architectures and protocols; multimedia systems; performance evaluations; and databases. He works on CachePortal project developing static/dynamic web content caching and also involve in the development of Content Awareness Network.

Oliver Po is currently a Research Engineer at Computer & Communications Research Laboratories (CCRL), NEC USA. From 1985 to 1993 he worked for IBM Taiwan as a system engineer. From 1994 to 1999 he joint
IBM Toronto Lab and worked as a database developer focusing on database functions and tools design and enhancement. He received his BS degree at 1984. His current work involves mainly the CCRL CachePortal project which studies and provides dynamic content caching solutions.

Kasım Selçuk Candan is a tenure track assistant professor at the Department of Computer Science and Engineering at the Arizona State University. He joined the department in August 1997, after receiving his Ph.D. from the Computer Science Department at the University of Maryland at College Park. His dissertation research concentrated on multimedia document authoring, presentation, and retrieval in distributed collaborative environments. He received the 1997 ACM DC Chapter award of Samuel N. Alexander Fellowship for his Ph.D. work. His research interests include development of formal models, indexing schemes, and retrieval algorithms for multimedia and Web information and development of novel query optimization and processing algorithms. He has published various articles in respected journals and conferences in related areas. He received his B.S. degree, first ranked in the department, in computer science from Bilkent University in Turkey in 1993.

Divyakant Agrawal is a Visiting Senior Research Scientist at CCRL, NEC USA, Inc. in San Jose. He received his Ph.D. from SUNY, Stony Brook. His current research interests are in the area of distributed hypermedia databases, multimedia information storage and retrieval, and Web based technologies.