Appeared in *Proceedings of VLDB Conference*, August 20-23, 2002

Issues and Evaluations of Caching Solutions for Web Application Acceleration

Wen-Syan Li Wang-Pin Hsiung Dmitri V. Kalashnikov Radu Sion
Oliver Po Divyakant Agrawal K. Selçuk Candan

C&C Research Laboratories - Silicon Valley
NEC USA, Inc.
10080 North Wolfe Road, Suite SW3-350
Cupertino, California 95014, USA
Email:{wen,whsiung,dvk,sion,oliver,agrawal,candan}@ccrl.sj.nec.com

Abstract

Response time is a key differentiation among electronic commerce (e-commerce) applications. For many e-commerce applications, Web pages are created dynamically based on the current state of a business stored in database systems. Recently, the topic of Web acceleration for database-driven Web applications has drawn a lot of attention in both the research community and commercial arena. In this paper, we analyze the factors that have impacts on the performance and scalability of Web applications. We discuss system architecture issues and describe approaches to deploying caching solutions for accelerating Web applications. We give the performance matrix measurement for network latency and various system architectures. The paper is summarized with a road map for creating high performance Web applications.

1 Introduction

For most Web sites, their contents are fairly static and can be generated in advance to serve a large number of requests. However, due to the dynamic nature of electronic commerce (e-commerce), e-commerce Web sites are usually database-driven and their contents change more frequently to reflect the current business state in the database systems. Architectural designs for a high performance e-commerce Web site is challenging because it involves integration of content delivery networks, caching appliances, Web servers, application servers, and databases.

Response time and reliability are two key points of differentiation among e-commerce Web sites. In business terms, the brand name of an e-commerce site is correlated to the type of experience users receive. The need to account for users' quality perception in designing Web servers for e-commerce systems has been highlighted in [1]. Snafus and slow-downs at major Web sites during special events or peak times demonstrate the difficulty of scaling up e-commerce sites. Slow response times and down times can be devastating for e-commerce sites as reported in a study by Zona Research[2] on the relationship between Web page download time and user abandonment rate. 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 jumps to 30% if the download time is around 8 seconds. The abandonment rate reaches 70% as download times exceed 12 seconds. This study clearly establishes the importance of fast response times to an e-commerce Web site to retain its customers.

In technical terms, ensuring the timely delivery of fresh dynamic content to end-users and engineering highly scalable e-commerce Web sites for special peak access times put heavy demand on IT staff. This is compounded by the ever-changing 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 to deploy cache servers, Web servers, application servers, and database systems at the backend.

Applying caching solutions for Web applications and content distribution has received a lot of attentions in the Web and database communities. In this paper, we analyze the issues of caching dynamic con-

Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002

tents in a database-driven Web site. We give an overview of several approaches to deploying caching solutions for accelerating Web applications as well as presenting NEC's *CachePortal* technology as a case study for multi-tiered caching solutions. We have conducted evaluation on caching solutions for Web application acceleration. Based on the performance matrix measured, we provide our views of a road map to a high performance Web site.

2 Approaches to Architectural Designs for Database-driven Web Sites

Several different approaches can be used to accelerate content delivery for 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 user requests can be served from multiple databases. In the following, we describe various alternatives and discuss their characteristics.

2.1 Basic System Architecture

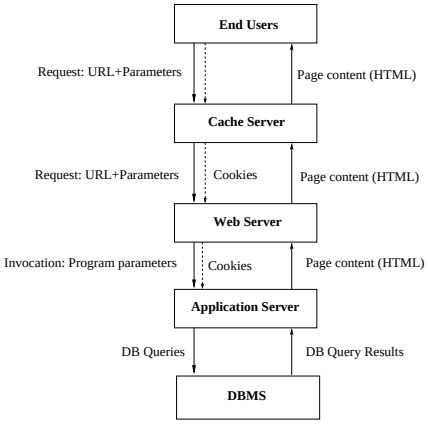

Figure 1: Typical Architecture of a Database-driven E-commerce Web Site

A basic system architecture of database-driven Web sites consists of the following components:

1. A Web server (WS) which receives user requests and delivers the dynamically generated Web pages.
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 database management system (DBMS) to store, maintain, and retrieve all the necessary data and information to model a business.

The architecture of such database-driven Web sites is illustrated in Figure 1. 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, and the application server communicates with the database using queries. When the application server receives such a request from the Web server, it processes it and it may access the underlying databases to extract the relevant information needed to dynamically generate the requested page. Note that the application server may issue several database queries to process a single URL request. There are two key obstacles for enabling dynamic content caching: (1) how to *automatically* derive the relationships between cached pages and database contents (i.e. URL and database query mapping); and (2) how to intelligently monitor database changes and how to efficiently identify impacted pages in the caches that need to be invalidated.

2.2 System Architecture with Dynamic Content Caching Enabled

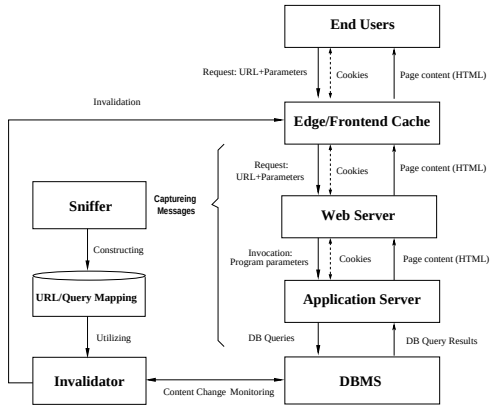

Figure 2: A Database-Driven E-Commerce Site with Dynamic Content Caching Enabled

In Figure 2, we show the architecture and data flow 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* to enable dynamic content caching and to ensure the freshness of dynamic content in cache servers. Our proposed technology *enables* dynamic content caching by deriving the *URL/DB Query Mapping*, the relationships between cached pages and the database contents that are used to generate these pages. The

mapping is constructed by the *sniffer*. The *sniffer* also monitors database changes by scanning database update log. The database change information is then utilized by the *invalidator* to identify the cached pages which are impacted by the database change. The data flow is illustrated in Figure 2.

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 [3, 4, 5, 6], which assume that such mappings are provided by the system designers. The detailed descriptions on automated construction of such URL and database query mapping are given in [7] and Section 3.

2.3 Redundant Web/Application Server Approach

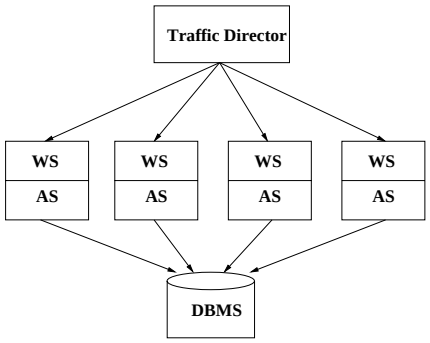

Figure 3: System Architecture Featuring Multiple Web/Application Servers

An alternative to caching is shown in Figure 3 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's 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.

2.4 Redundant Database Approach

In order to eliminate the database bottleneck, Figure 4 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 addition to the Web servers, there are four database management systems, each serving queries from a single Web and application server suite. 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.

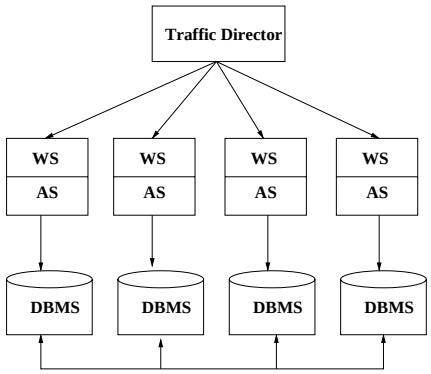

Figure 4: Architecture Featuring Multiple Servers

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 accesses to stale data, between the time at which the slave databases receive the update information until they receive the final commit acknowledgment, the updated data item (tuple, page, or table, depending on database implementation) must be locked. Therefore, queries that rely on this particular data item cannot be processed until all databases incorporate the 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 at the expense of weaker consistency.

2.5 Redundant Application Server/Data Cache Approach

Wide-area database replication allows database copies to be distributed across the network. The goal of this approach is to offset the high cost of replica synchronization by moving the data closer to the users (similar to caching in which data is moved closer to the users). 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 along with the database replicas. Figure 5 shows a configuration in which the WS/AS/DBMS *suite* is installed in remote parts of the network to handle most of the requests which require only accesses to the

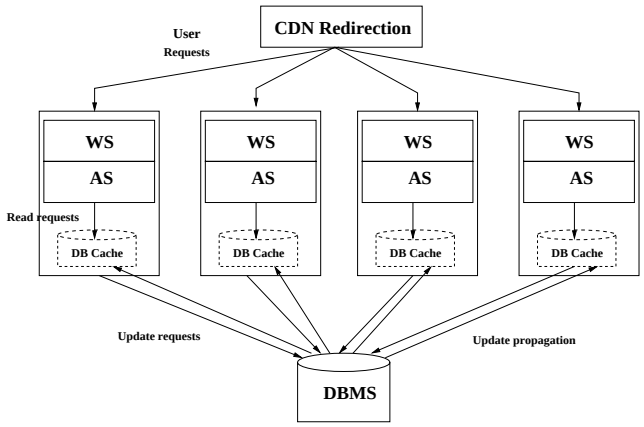

Figure 5: Caching Database Content for E-Commerce Site Acceleration

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 lightweight DBMS without full management functionality since no update operation will be executed at these suites at the remote locations; typically data centers. In [8] Qiong et al. present an extension to the existing federated features in IBM DB2 UDB, which enables a DB2 instance to become a DBCache without any application modification.

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. 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).

3 `CachePortal`: A Case Study of Multi-tiered Caching Solution

In this section we describe an open 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.

3.1 System Architecture

A generic system architecture deploying NEC's CachePortal technology is illustrated in Figure 6. The functionality of Web servers, application servers, and DBMSs are the same as described in Figure 1 in Section 1. In addition to the system described in [7], this new system architecture features a newly developed cache manager, an XJDBC cache server for JDBC database access layer caching, and a modified sniffer module. We describe their functions and data/control flows among these components next:

3.1.1 Cache servers

There are three types of cache servers: edge, frontend, and XJDBC cache servers.

Edge cache servers: Edge cache servers are usually provided by CDN service providers, such as Akamai[9], that have arrangements with ISPs to have their cache server located within ISPs.

Frontend cache servers: Frontend cache servers are usually provided by content providers and Web sites instead of CDN providers. Frontend cache servers can be either software-based or hardware-based.

XJDBC cache servers: The XJDBC cache server is deployed between the AS and the DBMS to provide JDBC layer data caching functionality. It provides not only the necessary connectivity between the application server and DBMS, but also provides caching, invalidation, pre-fetching, refresh functions for database accesses by the application server. Unlike the frontend/edge cache servers which store Web pages, the storage unit at the XJDBC cache servers are the query results. In our current implementation, all three kinds of cache servers are software-based. Edge cache servers and front-end cache servers are implemented on top of Apache Web Servers.

3.1.2 CachePortal software modules

Sniffer: *Sniffer* is a software module that behaves as a wrapper to "sniff" the messages incoming and outgoing to and from the application servers. When a request arrives, *Sniffer* extracts the URL string, cookie information, and IP address of the requesting cache server. After this information is extracted, *Sniffer* assigns (i.e. tags) each request a *unique* identifier and forwards the request back to the appropriate application server servlet. When a unique-identifier-tagged servlet issues database queries through JDBC, *Sniffer* extracts the identifier of the requesting servlet and the query statements. *Sniffer* then uses the unique identifier to associate each page and its corresponding query statements issued to generate the page. The association is then stored in the $URL/DBQueryMapping$.

Invalidator: *Invalidator* periodically polls the database log to monitor database content changes. The invalidator receives the log in the form of new value and old value for those tuples which have been inserted, deleted, and updated. Based on the tuple values and $URL/DBQueryMapping$, *Invalidator* is responsible for identifying the query results that

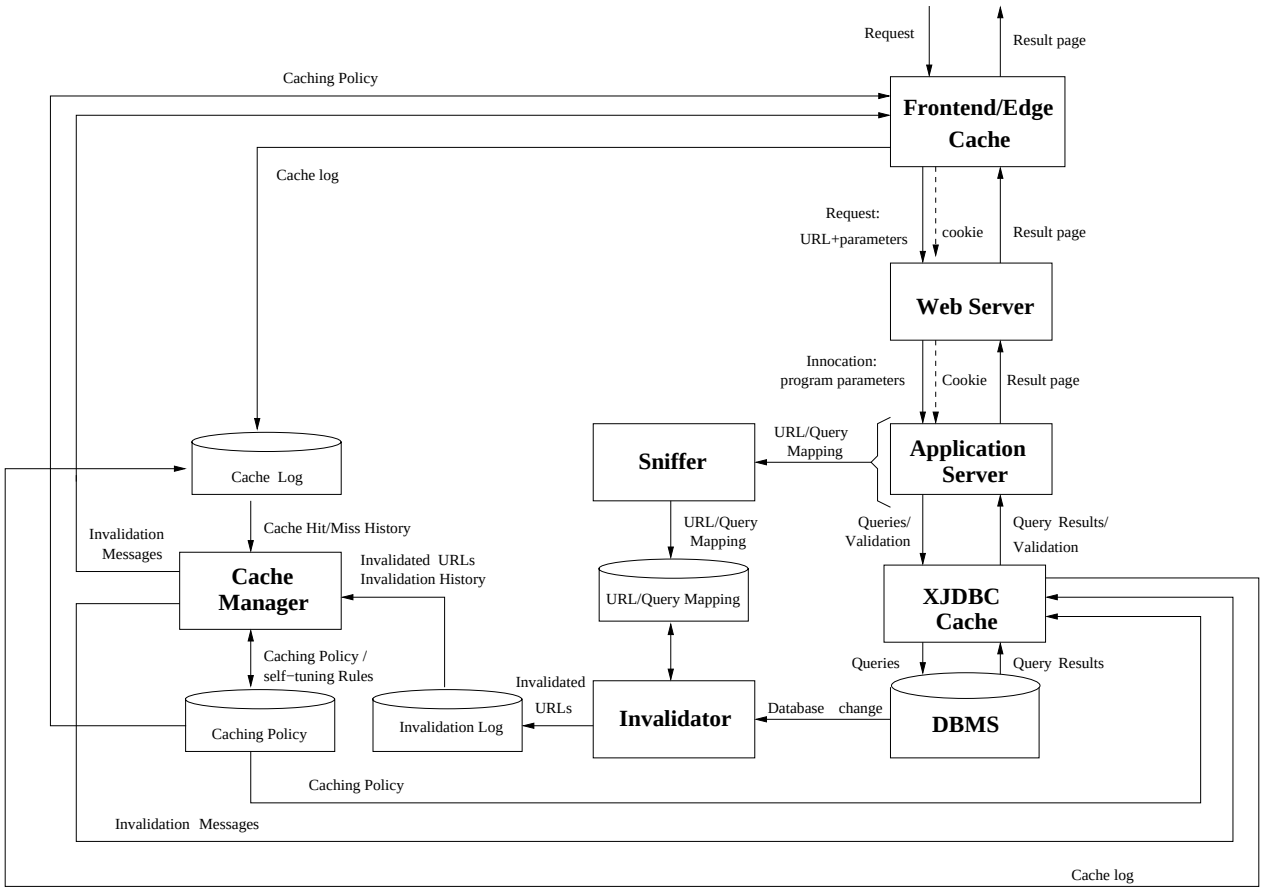

Figure 6: System architecture of a database-driven web site with *CachePortal* technology

should be invalidated. Once the query results are identified, the query statements and associated cookie information are sent to *Cache Manager* to determine the corresponding URLs and the IP addresses where the pages are cached. *Cache Manager* then sends out invalidation messages to the appropriate cache servers. *Invalidator* is also responsible for maintaining the invalidation log, which will be utilized by *Cache Manager* to determine the caching priority of each page. The detailed information of invalidation schemes is described in [10].

Cache Manager: *Cache Manager* is responsible for various tasks: (a) it maintains the IP addresses of the cache servers where the pages are cached; (b) it tunes replacement strategies and enforces the caching policies, which specify the rules regulating which URLs must be cached and which others must not be cached; (c) it pulls the cache logs from the frontend, edge, and XJDBC cache servers and (d) sends out invalidation messages. In the current implementation, the cache manager maintains persistent connections with all cache servers and the XJDBC cache.

3.1.3 Operational files

URL/DB Query Mapping: The URL/DB query mapping is constructed by the sniffer. The URLs here are extended to include the IP address of the cache servers and cookie information. The URL/DB Query Mapping is maintained as follows: (1) When a query result is invalidated (i.e. identified by the Invalidator), the Invalidator will delete the URL/DB Query entries that are associated with the query being invalidated; (2) If a page is specified as non-cacheable based on the caching policy, the URL/DB Query entries that are associated with the page are deleted from the mapping so that the Invalidator does not need to check such pages; and (3) If a query result set in the XJDBC cache is deleted due to replacement operations initiated by the tuning tasks of the Cache Manager or the XJDBC cache itself, the URL/DB Query entries that are associated with deleted pages or query result sets are removed from the mapping so that the Invalidator does not need to check.

Cache logs: The cache hit and miss history files are gathered by the cache manager from front-end cache servers, edge cache servers, and the XJDBC cache servers. The cache hit and miss history files are used by the Cache Manager to tune the cache policy and the caching priority of objects (i.e. pages and query result sets).

Cache policy: The cache policy consists of rules on what to cache and what not to cache using regular expressions. It also specifies the caching priority of cer-

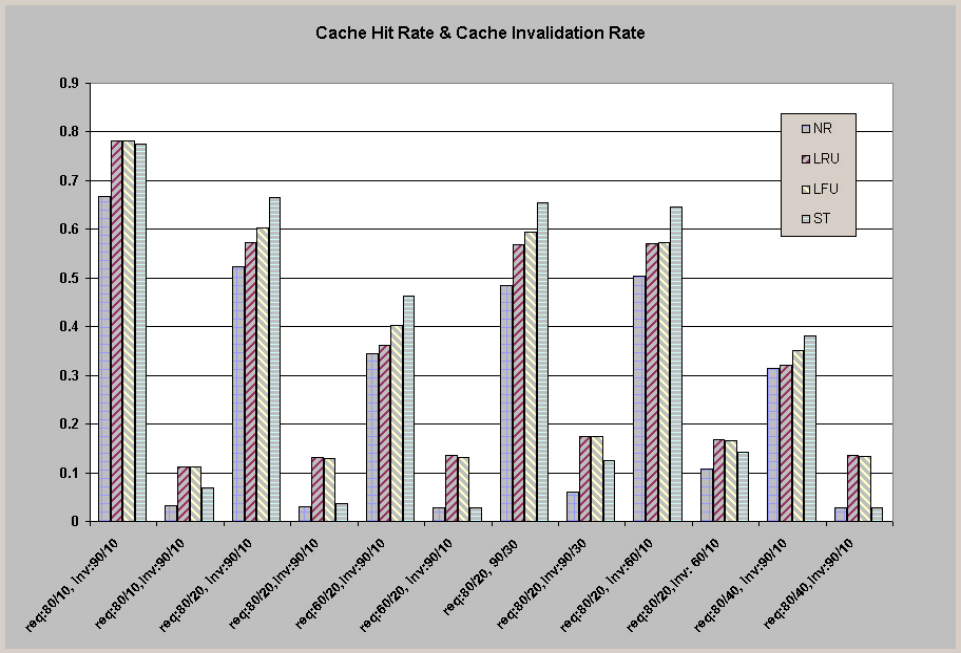

Figure 7: Comparisons of hit rates and invalidation rates

tain objects that are currently in the caches or objects with high hit rates. The cache policy also includes self-tuning algorithm specifications that will be used by the Cache Manager to tune the cache replacement strategy.

Invalidation log: The invalidation log file stores the invalidation history and frequency of objects cached. This information is used by the Cache Manager to tune the cache replacement strategy and calculate object caching priority.

3.2 Cache Management in Consideration of Invalidation

The problem of cache replacement has been extensively studied. Many algorithms have been proposed for general purpose caching, such as LRU and LFU. Some variations of these are designed specifically for cache replacement of Web pages. However, in the scope of dynamic caching for a Web site, cache invalidation rates is an important factor since a high invalidation rate will lead to a potentially high cache miss rate in the future. As a result of the high miss rate, the WAS and DBMS have to handle a higher load to generate Web pages. Also, as a result of high invalidation rate, the *Invalidator* component requires to handle more invalidation checking, issue more polling queries to the DBMS, and send out more invalidation messages.

We have developed a self tuning cache replacement algorithm (ST) that takes into consideration (1) user access patterns, (2) page invalidation pattern, and (3) temporal locality.

The caching priority of each page is re-calculated periodically. In the current implementation, the priority is re-calculated every minute. Note that the frequency of re-calculation does have an impact on the cache hit rate. Potentially, the more often the caching priorities are re-calculated, the higher are the cache hit rates. The frequency of re-calculation should be dynamically adjusted by considering the trade-off between the benefit of higher hit rates and the additional cost incurred due to frequent re-calculations.

The access rate and the invalidation rate is the access count and invalidation count within a time period. The caching priority of a page during a time period t, $Caching\_priority(t)$, is calculated as

$$(1-\alpha) \times \frac{access\_rate}{invalidation\_rate + 1} + \alpha \times Caching\_priority(t-1)$$

where α is the temporal decay factor whose value is between 0 and 1. A value of 1 for α makes the system treat all access patterns equally, while a value of 0 makes the system consider only the access patterns during the current time period. Currently the value of α is set to 0.8.

The intuition behind this formula is that it estimates the average number of accesses for the page between any two successive invalidations. The higher this number the larger the benefit to keep this page in the cache. Note that in our current implementation, when a page is initially accessed by users, the cache manager does not try to keep it in the cache unless it is specified by the cache rule as "must be cached". The cache manager will monitor the caching priority of a page and will cache the page when its priority is higher than a certain threshold value.

We have conducted experiments to evaluate our algorithm (ST) and compare its performance with LRU and LFU algorithms, two of the most frequently used

algorithms. We also measured the performance of cache hits when no replacement strategy is deployed (NR). Under the NR strategy, the cache server will continue to fill in the pages until it is full. When the cache is full, a page will be cached only after some pages are removed due to invalidation. In the latter part of this section, we will provide our analysis on the performance of the ST algorithm. In the analysis, the hit rate of NR algorithm will be viewed as a lower bound of the hit rate. The analysis will also show the potential upper bound for the hit rates under various experimental settings.

The general experimental settings are as follows: The total number of the pages in the system is 5000, whereas the cache size is 1000 pages. All pages are of the same size. In the experiments, the cache starts empty and each experiment is run for 200 minutes, where 5 requests are generated per second. The page access pattern shows an uneven distribution ($\frac{W}{X}$, where W percentage of the user requests access to X percentage out of all 5000 pages). The invalidation rate is also 5 requests among all 5000 pages per second. The invalidation pattern also shows an uneven distribution ($\frac{Y}{Z}$, where Y percentage of invalidations affect Z percent of the pages). When the access patterns change, these parameters stay the same, whereas the set of popular pages change. Note that the invalidation pattern is assumed to be independent of the user access patterns. The cache hit rate is calculated every 200 seconds.

In Figure 7, we summarize the average hit rates (long bars) and the average cache invalidation rates (short bars) for various experimental settings. The hit rates and cache invalidation rates are calculated after both rates become stable. As we can see in this figure, the hit rates achieved by the ST replacement strategy are better than other algorithms in all settings. In general it performs 5 to 10 percent better than the most frequently used LRU and LFU algorithms. This improvement is good, but it should not be viewed as significant by itself. More importantly, the cache invalidation rates tuned by the ST replacement strategy are 75% lower than the LRU and LFU algorithms. This is because LRU and LFU only track access patterns and do not account for update activities and invalidation patterns. On the other hand, ST tracks both. The combination of higher hit rates and lower invalidation rates give us the advantages of faster delivery from the caches and faster generation of the missed pages due to lower system load.

3.3 Handling Fragmented Pages and JSP-based Dynamically Assembled Pages

`Frames` are frequently used in most commercial Web sites for the flexibility they provide in page formatting and layout as well as to simplify users' navigation. The *Sniffer* treats each request independently. Consequently, using frames for page formatting and layout is easily handled by the *sniffer*. As a matter of fact, composing pages using frame makes a Web site more cache friendly since it allows more caching opportunities by producing pages at a finer granularity.

The concept of independently caching the fragments of a web page and assembling them dynamically has advantages of reducing load at the WAS. JSP (Java Servlet Pages) pages are similar to HTML pages in terms of syntax. In these pages, Java scripts are embedded into the HTML code to enable web servers to generate dynamic content. The rest of the page appears as a static HTML page. The dynamic content is limited only to those fragment pages where the Java script is embedded. ESI (Edge Side Includes) is a markup language that developers can specify different caching properties for different fragments within the same page. These fragments can be cached as independent entities at the edge side caches. These fragments are assembled into a single HTML page at the edge caches. With JSP and ESI, the caching is at the granularity of a fragment rather than at the page level. Web site designers who use ESI can benefit from the sniffer and invalidation functionality provided by NEC's *CachePortal* technology at the fragment level. Similarly, Datta et al.[11] also address the issue of caching fragmented pages.

4 Evaluations of Approaches to Accelerating Web Applications

In this section, we evaluate the factors that impact Web site performance, including network latency, resource utilization, cache hit rates, cache server locations, and invalidation methods. We start with a description of the experimental environment followed by an analysis of the experimental results.

4.1 Experiment Setting

We have deployed a number of servers at NEC locations world wide acting as a small-scale content delivery network. All of the machines where the Web servers and applications are located are installed on Pentium III 700Mhz one CPU PCs running Redhat Linux 7.1 with 1GB of memory.

For the experiments reported in this section, we utilize those servers located in NEC facilities in San Jose California (SJ), Princeton New Jersey (NJ), and Tokyo Japan (JP). Each of these three facilities is located close to a network backbone. Two networks are available in San Jose facilities: one is used by the C&C Research Laboratories (referred to as CCRL) and the other one is used by *cacheporal.com* (referred to as SJ). All Web servers, application servers, and databases are located in SJ.

All user requests are from the CCRL and JP locations. The average numbers of hubs between CCRL and SJ, and JP and SJ are 15 and 17 respectively. The average throughput measured for JP and SJ, CCRL and SJ, and SJ and SJ are 85.5 Kilobytes/second, 89.7

Kilobytes/second, and 589.5 Kilobytes/second respectively. The average round trip time between JP and SJ, CCRL and SJ, and SJ and SJ are 413 ms, 321 ms, and 0.2 ms respectively. Note that the round trip time is an integrated measurement that really has an impact on the network latency. To summarize, the connectivity between CCRL and SJ is better than that between JP and SJ. And, the connectivity within the same network is substantially better than that across the Internet.

Each group of front-end caches, Web servers, application servers, and databases are located in the same network. Similarly, edge caches are placed in the same network as the users they serve.

Oracle 8i is used as the DBMS, BEA WebLogic 6.0 is used for the Web and Application Server, and Apache is used as for both the edge cache servers and the front-end cache servers. The database contains 7 tables with 1 million records each.

Configuration 1: the Web server, the application server, and the database are located on the same machine and *CachePortal* technology is not deployed; thus no cache is used.

Configuration 2: the Web server and the application server are located on the same machine and database is located on a separate machine. *CachePortal* technology is not deployed; thus no cache is used.

Configuration 3: the Web server, the application server, and the database are located on the same machine. *CachePortal* technology is deployed and a front-end cache, a cache close to Web and application servers, is used.

Configuration 4: the Web server and the application server are located on the same machine. and database is located on a separate machine. *CachePortal* technology is deployed and an edge cache, a cache close to the users, is used.

We next present the experimental results on important factors that have impacts to the Web site performance.

4.2 Effects of Network Latency

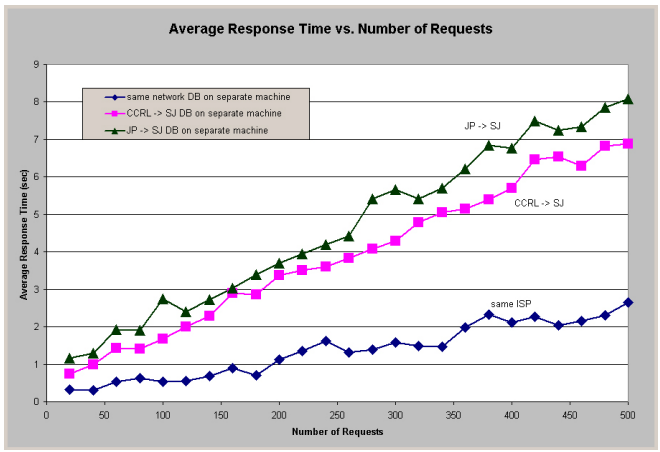

Figure 8: Network Latency Factor

The literature contains many studies, including [12], that evaluate the impact of the network latency on the response time. However, large objects, such as the image files, are the subjects of most of these studies. In general, the average size of a dynamic web pages is small (i.e. 4K bytes per fragment page in our study). Furthermore, the WAS is in general far less scalable compared with Apache servers, potentially rendering back-end delays to be more significant than network delays. Therefore, it is essential to conduct experiments to establish a qualitative correlation between network latency and response time in the context of dynamic web content delivery.

Figure 8 shows the response times observed by the users for requests issued from JP, CCRL, and SJ to SJ. The Web sites in SJ are based on Configuration 2. All requests are served by the WAS and no cache is used. Although the pages are relatively small (4K byte each), the network latency has significant impact on the response times. Response times for requests in the same ISP (i.e. SJ) is the lowest. On the other hand, response times for requests across the Internet are substantially higher due to the network latency. We also find that the response time for the requests from CCRL to SJ is lower than that from JP to SJ. This experimental result is consistent with the round trip times we measured from the networks.

4.3 Effects of CPU Resources

We next evaluated the impact of CPU resources on the response time. Experimental results in Figure 9 show that the response time for Configuration 1 can be reduced by almost 50% when a dedicated machine is deployed for the DBMS. This is because in Configuration 1, the Web server, the application server, and the database compete for system resources with each other.

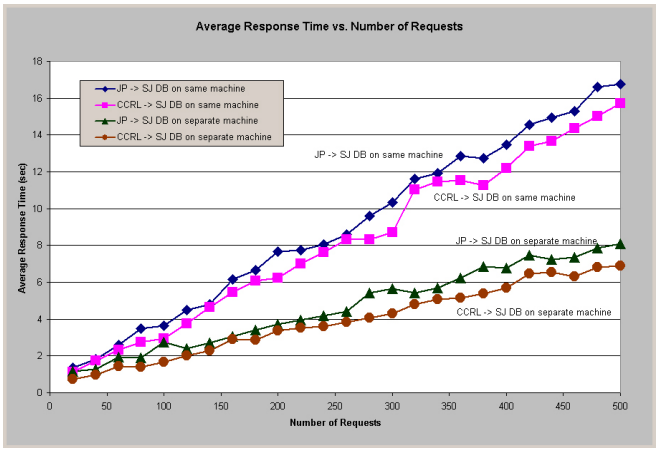

Figure 9: Hardware Resource Factor

4.4 Effects of Cache Hit and Request Rates

The next experiment we conducted is to examine the impact of cache hit and request arrival rates on

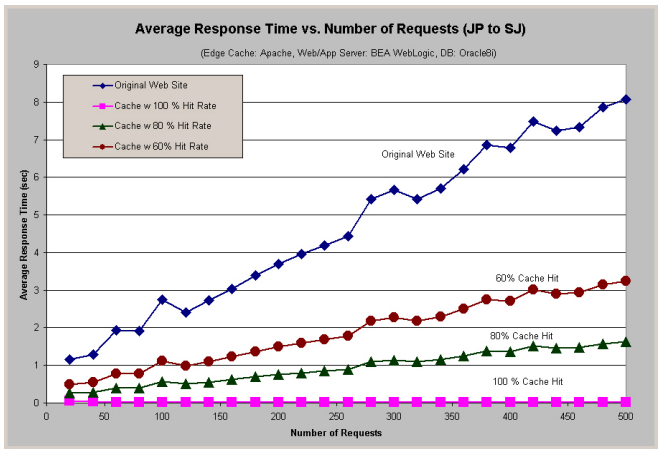

Figure 10: Edge Cache Hit Rate Factor

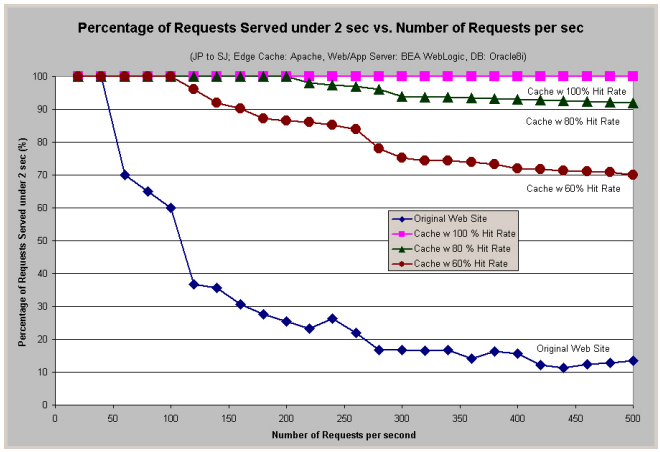

Figure 11: Percentage of Requests Served under 2 Seconds at Various Edge Cache Hit Rates

the response time for Configuration 4. The rate at which requests were generated was varied from 20 requests/second to 500 requests/second. Figure 10 depicts that the response time of the original Web site increases linearly. On the other hand, the response time for configurations with *CachePortal* deployed to enable dynamic content caching remains low as the request rate is increased. The advantage of caching is quantified by a notion referred to as the *hit ratio*. For example a cache hit ratio of 80% means that 80% of the requests are served from the cache. Clearly, the hit ratio has an inverse relationship with the response time. The figure quantifies the advantages of increasing the hit ratio to improve response time.

Another meaningful measure is the percentage of requests that are served under 7 seconds, where the user abandonment rate is less than 2 percent as claimed in the study by Zona Research[2]. However, since a Web page usually consists of multiple fragment pages, to ensure that the user requests are served within 7 seconds, each page (either whole page or fragment page) needs to be served in much less than 7 seconds. As a result, we measure the percentage of the requests that are served under 2 seconds instead of 7 seconds. In Figure 11, we see that the system architectures (upper three lines) with *CachePortal* technology provide good response time most of time whereas the system architectures without *CachePortal* technology (lowest line) have extremely poor performance. The experimental results show that even when dealing with a high volume of request at 500 requests per second and a modest cache hit rate, an e-commerce site with *CachePortal* technology can still ensure that 70 percent of the requests are served in 2 seconds or less.

4.5 Edge Cache versus Frontend Cache

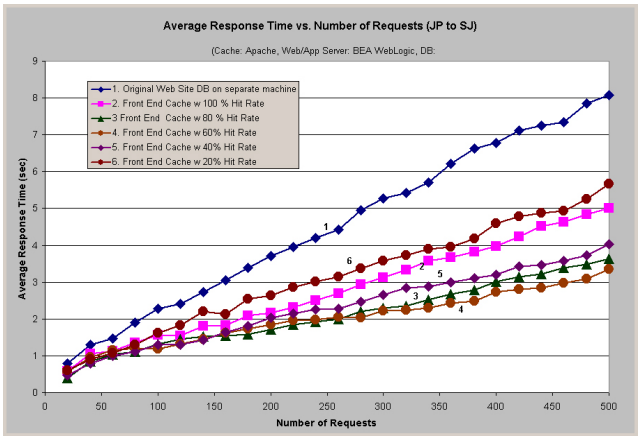

Figure 12: Front-End Cache Hit Rate Factor

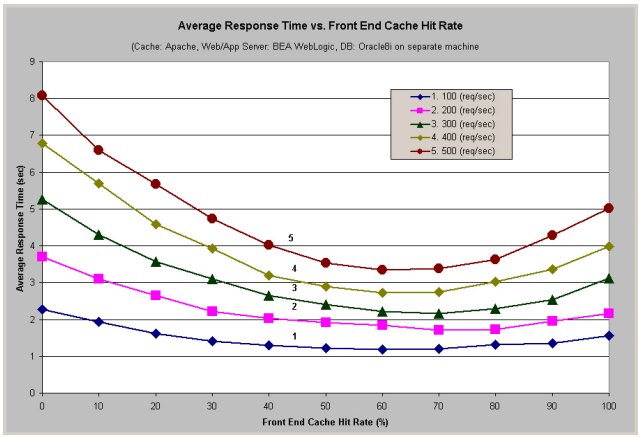

Figure 13: User Response Time versus Front-end Cache Hit Rates

In Figure 10, we see that the edge cache with a high hit rate can improve the user response time significantly. In the next experiment, we evaluated the impact of cache hit rates on response time in configuration 3 where a frontend cache is used. The experimental results are shown in Figure 12. As expected, the response time of the original Web site increases linearly. On the other hand, the response time for configurations with *CachePortal* to enable dynamic content caching remains low as the request rate is increased.

Surprisingly, we discovered that a higher the cache hit ratio did *not* necessarily result in lowering the response time. We then conducted more experiments by

varying the hit rate from 0% to 100% for request rates of 100, 200, 300, 400, and 500 per second (Figure 13). The experimental results indicate that the lowest response time occurs when the front-end cache hit rates are between 60% and 70%. This was rather surprising. After further investigation, we found out that the system resource factor also has a significant impact on the response time in this configuration. This is because the WAS needs to maintain the session connections for much longer time in configuration 3 compared with that in configuration 4. As a result, when the hit rates are around 65%, the load is better distributed between the cache server and the WAS.

From these experimental results, we conclude that the benefit of cache servers are clear but where the cache servers are located is also very important. Edge caches provide better and consistent response time improvement as the hit ratio is increased. This is not the case with the front-end caches. However, the deployment of cache servers at the network edges is in general more expensive and requires a larger investment. Hence there is a trade-off in using edge caches versus front-end caches.

5 Road Map to High Performance Web Sites

We use the empirical results obtained through our experimental set-up to identify the factors that most influence the performance of Web sites. As discussed earlier, there are four important factors that impact the performance of the Web sites: network latency, system resource, cache hit ratio, and cache server locations. We have extensively evaluated caching solutions for a variety of system configurations. In Figure 14, we illustrate the performance matrix measured in this paper (cache hit = 80%) and we summarize the impacts of these factors as follows:

- Network latency: It is always better to bring the applications, data, servers, and caches close to the users.
- System resource: System contention among several processes/applications can result in performance bottleneck. For example, we found that deploying a dedicated machine for DBMS improves the performance by average of 2.3X.
- Cache hit ratio: Cache hit ratios are tuneable and the higher the hit ratio, the better the response time (in general). However, the location of cache servers do have a significant impact to the performance gain. For example, in our experiments we found that under certain configurations the front-end cache can become a bottleneck if the hit ratio exceeds beyond 70%. Thus the rule of thumb is that higher hit ratios are better as long as the server that holds the cache has excess capacity to serve additional user requests.
- Cache server locations: edge cache servers provide higher performance gain. Deploying edge caches yield 50% more response time improvement over deploying front-end caches. However, deploying edge caches is more expensive.

In addition to these factors, there are two key obstacles in enabling dynamic content caching: (1) how to *automatically* derive the relationships between cached pages and database contents (i.e. *URL/database query mapping*); and (2) how to intelligently monitor database changes and how to efficiently identify impacted pages in the caches that need to be invalidated. Aiming at these two key obstacles, NEC's *CachePortal*[7, 10] does provide a suite of solutions as described in this paper.

In Figure 15, we show a road map to improve the response time of a Web site. In the figure, each box represents a system configuration plotted in Figure 14. On the left of Figure 15, we show a Web site without any caching solution and its performance is poor (with response time higher than 17 seconds). However, the Web site has multiple options to improve its response time. At the bottom, the figure shows the average response time for such a system configuration in responding to a traffic of 500 requests per second. From the base configuration (configuration 1), we have three options: deploying dedicated machine for DBMS; deploying *CachePortal* and front-end cache; or deploying *CachePortal* and edge caches.

Based our experimental evaluation, we found that option 1 results in 53% reduction in response time; option 2 results in 72% reduction; and option 3 results in 81% reduction. However, the best case arises when we use both dedicated machine for the DBMS and deploy *CachePortal* with edge-caches. In this case the overall response time is reduced by 89%. As we can see and learn from the experiments, the best system configuration is to deploy dedicated machines for DBMS and WAS. Furthermore, the Web site should deploy *CachePortal* (or another invalidation system) to enable dynamic content caching and CDN services that can deliver database driven content through edge caches.

6 Related Work

Applying caching solutions for Web applications and content distribution has received a lot of attentions in the Web and database communities[7, 13, 14, 15, 16]. 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 in existing e-commerce framework. Challenger et al. [4, 5, 6] 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

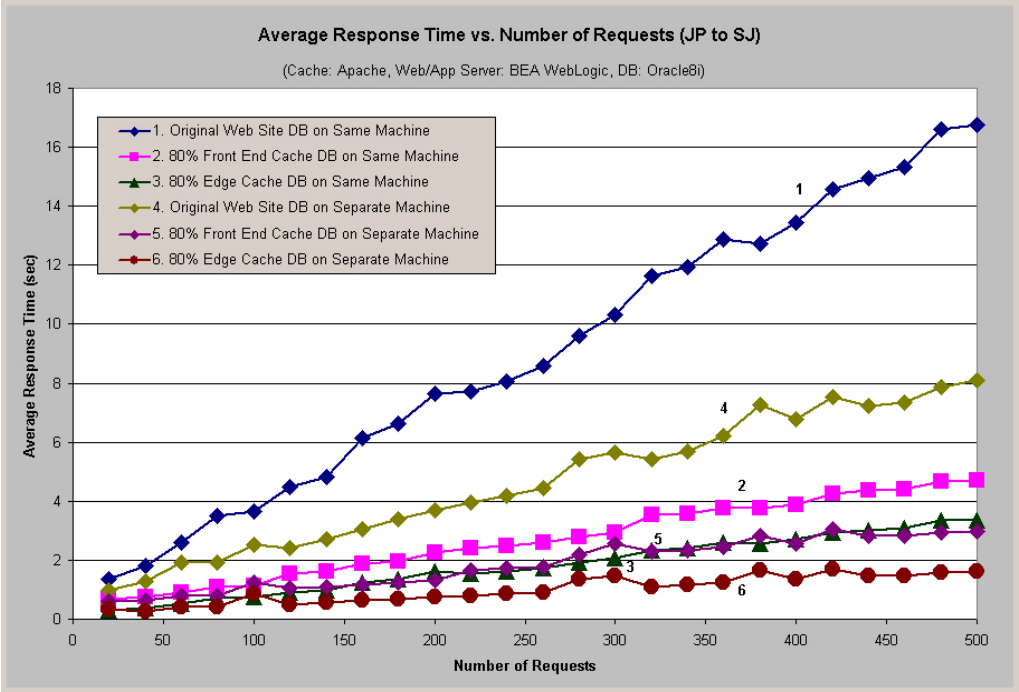

Figure 14: Performance matrix measurement for various caching solution deployments (cache hit = 80%)

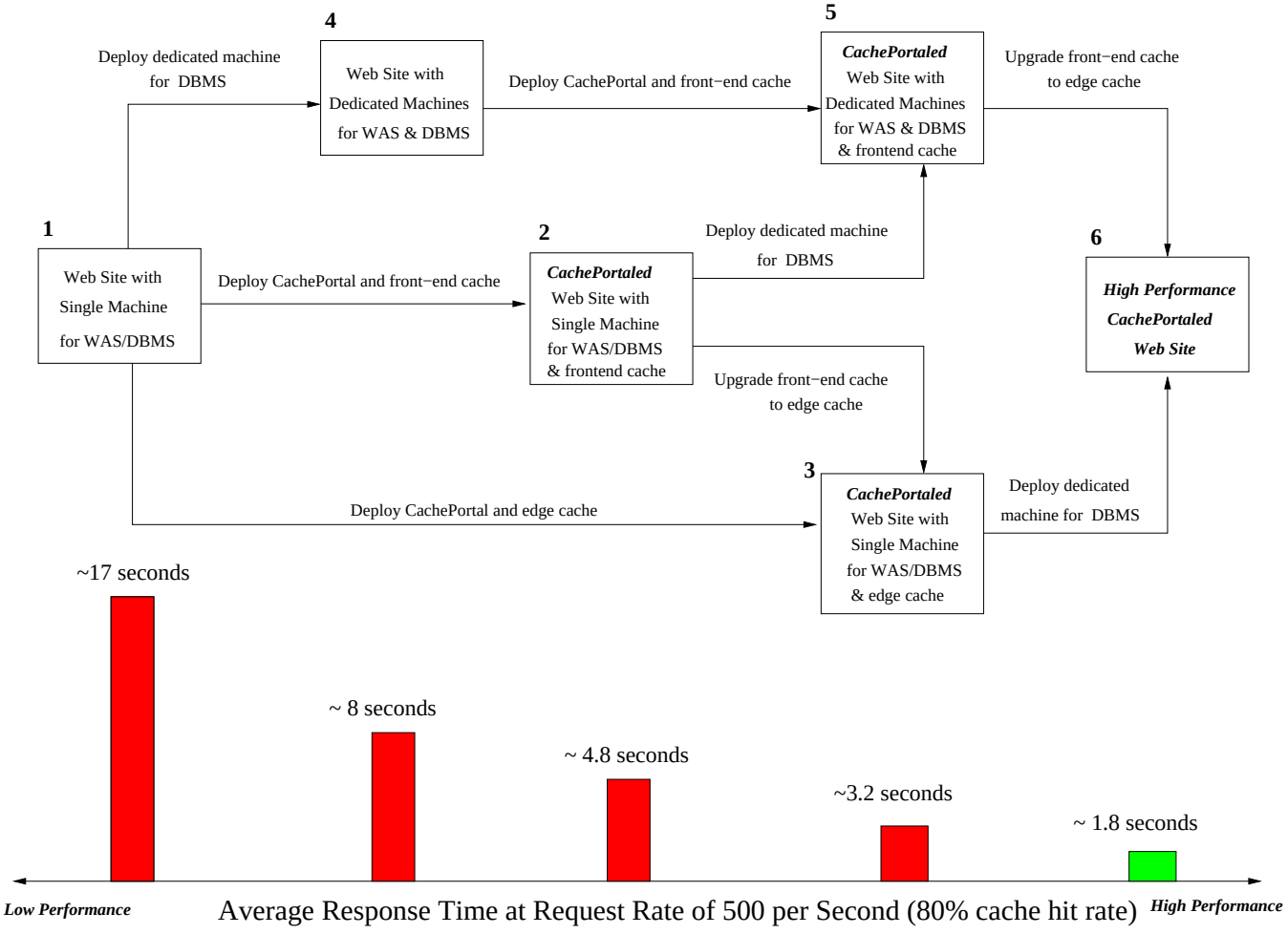

Figure 15: Road map to high performance web sites and response time improvement

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. In [17] Deolasee et al. proposed an approach of adaptive push-pull to ensuring Web content freshness. In [18] Ninan et al. introduce the notion of cooperative consistency along with a mechanism, called clustered leases, to achieve both scalability and flexibility of consistency maintenance. By using a single lease for multiple proxies, clustered leases allows the notion of leases to be applied in a flexible and scalable manner to CDNs.

7 Conclusion

In this paper, we begin by addressing system architecture issues associated with caching and acceleration for Web applications. We analyze the factors that have impacts on the performance and scalability of a database-driven Web site in the scope of NEC's *CachePortal* technology. We have built an e-commerce Web site using some of the most popular commercially available software as building blocks. We identify the important factors that impact the performance of the Web sites, including network latency, system resource, cache hit ratio, and cache server locations. Through extensive experimental evaluations, we observe that in general caching is a good solution to Web application acceleration. We believe these experimental results provide valuable guideline and road map to engineer a high performance Web site.

References

[1] N. Bhatti, A. Bouch, and A. Kuchinsky. Integrating user-perceived quality into web server design. In *Proceedings of the 9th World-Wide Web Conference*, pages 1–16, Amsterdam, The Netherlands, June 2000.

[2] Zona Research. http://www.zonaresearch.com/.

[3] Persistent Software Systems Inc. http://www.dynamai.com/.

[4] Jim Challenger, Paul Dantzig, and Arun Iyengar. A Scalable and Highly Available System for Serving Dynamic Data at Frequently Accessed Web Sites. In *Proceedings of ACM/IEEE Supercomputing'98*, Orlando, Florida, November 1998.

[5] Jim Challenger, Arun Iyengar, and Paul Dantzig. Scalable System for Consistently Caching Dynamic Web Data. In *Proceedings of the IEEE INFOCOM'99*, New York, New York, March 1999. IEEE.

[6] Eric Levy, Arun Iyengar, Junehwa Song, and Daniel Dias. Design and Performance of a Web Server Accelerator. In *Proceedings of the IEEE INFOCOM'99*, New York, New York, March 1999. IEEE.

[7] K. Seluk Candan, Wen-Syan Li, Qiong Luo, Wang-Pin Hsiung, and Divyakant Agrawal. Enabling Dynamic Content Caching for Database-Driven Web Sites. In *Proceedings of the 2001 ACM SIGMOD Conference*, Santa Barbara, CA, USA, May 2001. ACM.

[8] Qiong Luo, Sailesh Krishnamurthy, C. Mohan, Hamid Pirahesh, Honguk Woo, Bruce G. Lindsay, and Jeffrey F. Naughton. Middle-tier Database Caching for e-Business. In *Proceedings of 2002 ACM SIGMOD Conference*, Madison, Wisconsin, USA, June 2002.

[9] Akamai Technology. *Information available at* `http://www.akamai.com/html/sv/code.html`.

[10] K. Selcuk Candan, Divyakant Agrawal, Wen-Syan Li, Oliver Po, and Wang-Pin Hsiung. View Invalidation for Dynamic Content Caching in Multitiered Architectures . In *Proceedings of the 28th Very Large Data Bases Conference*, Hongkong, China, August 2002.

[11] Anindya Datta, Kaushik Dutta, Helen M. Thomas, Debra E. VanderMeer, Suresha, and Krithi Ramamritham. Proxy-Based Acceleration of Dynamically Generated Content on the World Wide Web: An Approach and Implementation. In *Proceedings of 2002 ACM SIGMOD Conference*, Madison, Wisconsin, USA, June 2002.

[12] R.L. Carter and M.E. Crovella. On the network impact of dynamic server selection. *Computer Networks*, 31(23-24):2529–2558, 1999.

[13] C. Mohan. Application Servers: Born-Again TP Monitors for the Web? (Panel Abstract). In *Proceedings of the 2001 ACM SIGMOD Conference*, Santa Barbara, CA, USA, August 2001.

[14] C. Mohan. Caching Technologies for Web Applications. In *Proceedings of the 2001 VLDB Conference*, Roma, Italy, September 2001.

[15] Mitch Cherniack, Michael J. Franklin, and Stanley B. Zdonik. Data Management for Pervasive Computing. In *Proceedings of the 2001 VLDB Conference*, Roma, Italy, September 2001.

[16] Qiong Luo and Jeffrey F. Naughton. Form-Based Proxy Caching for Database-Backed Web Sites. In *Proceedings of the 2001 VLDB Conference*, Roma, Italy, September 2001.

[17] P. Deolasee and A. Katkar and A. Panchbudhe and K. Ramamritham and P. Shenoy. Adaptive Push-Pull: Dissemination of Dynamic Web Data. In *the Proceedings of the 10th WWW Confernece*, Hong Kong, China, May 2001.

[18] Anoop Ninan and Purushottam Kulkarni and Prashant Shenoy and Krithi Ramamritham and Renu Tewari. Cooperative Leases: Scalable Consistency Maintenance in Content Distribution Networks. In *the Proceedings of the 11th WWW Confernece*, Honolulu, Hawaii, USA, May 2003.