Acceleration of Data Center-Hosted Distributed Database-driven Web Applications

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

C&C Research Laboratories - Silicon Valley
NEC USA, Inc.
10080 North Wolfe Road, Suite SW3-350
Cupertino, California 95014
Email:{wen,oliver,whsiung,candan,agrawal}@ccrl.sj.nec.com
Tel:408-863-6008 Fax:408-863-6099

Abstract

Response time is essential to many Web applications. Consequently, many database-driven Web applications rely on data centers that host applications and database contents. Such IT infrastructure enables generation of requested pages at locations much close to the end-users, avoiding network latency. However, it the has additional challenges of database/data center synchronization and data freshness. In this paper, we describe the deployment of NEC's `CachePortal` dynamic content caching technology on a database-driven Web site in a data center-based distribution infrastructure. The new system architecture has been experimentally evaluated and the results show that the deployment of NEC's `CachePortal` accelerates the dynamic content delivery up to 7 times while maintaining the same level or better content freshness.

Keywords: Data center, Web hosting, dynamic content, Web acceleration, application server, edge cache, CDN, edge computing, distributed databases, view migration

1 Introduction

Response time and reliability are two key differentiation points for 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 for accounting for users' quality perception in designing Web servers for e-commerce systems has been highlighted by [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 load 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 deploy cache servers, Web servers, application servers, and database systems at the backend.

As the significance of CDN services [3, 4] becomes widespread, many database and application server vendors are beginning to integrate Web acceleration though data caching in their software. Examples include Oracle 9i [5] which features a suite of application server, Web server, and data cache for deployment at data centers for accelerating the delivery of dynamic content. With all these newly available software for dynamic content caching and delivery acceleration, it is more flexible to architect a "distributed" Web site, which may actually be located in multiple networks and geographical regions. This kind of Web site deployment infrastructure can generate requested pages at locations much closer to the users, avoiding network latency. However, it has additional challenges of of database synchronization and data freshness. Since in most e-commerce Web applications, database changes must be monitored so that those cached pages that are impacted by data updates can be invalidated or refreshed in an acceptable period of time (i.e. 60 seconds).

In this paper, we describe the deployment of NEC's `CachePortal` dynamic content caching technology in a data center-based distribution infrastructure. In particular, we report the results of extensive experiments we conducted to evaluate the performance gained with using `CachePortal`. These experiments were carried on a prototype deployment, which uses BEA WebLogic WAS (Web Application Server), Oracle DBMS, and Apache Web Servers (used as front-end and edge cache servers) as building blocks.

The rest of this paper is organized as follows. In Section 2, we describe the architectural design of typical data center-hosted distributed database-driven Web applications. In Section 3 we give an overview of NEC's `CachePortal` dynamic content caching and invalidation technology. In Section 4 we describe the new system architecture of data center-hosted distributed Web applications that utilizes NEC's `CachePortal` technology to enable dynamic content caching and consequently accelerate content delivery performance. In Section 5 we discuss the experiments to evaluate performance gain and scalability of our proposed system architecture. In Section 6, we discuss how our proposed architecture also supports a higher level of freshness compared with the standard data center-hosted Web applications. In section 7 we summarize related work and in section 8 we give

our concluding remarks.

2 System Architecture of Data Center-Hosted Database-driven Web Sites

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. Optionally, edge caches or frontend caches are deployed to accelerate static content delivery.

Wide-area database replication technology and availability of data centers 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 data closer to the users (similar to caching in which data is moved closer to the users reducing network latency). As shown in Figure 1, 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. The figure shows a configuration in which the WS/AS/DBMS *suite* is installed in remote parts of the network to handle requests which require only accesses to the database replicas. Note that in many commercial products, Web servers and application servers are integrated as Web application servers (WAS). The WS/AS/DBMS suites are hosted in data centers that are strategically located in key network peering points across the Internets. 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.

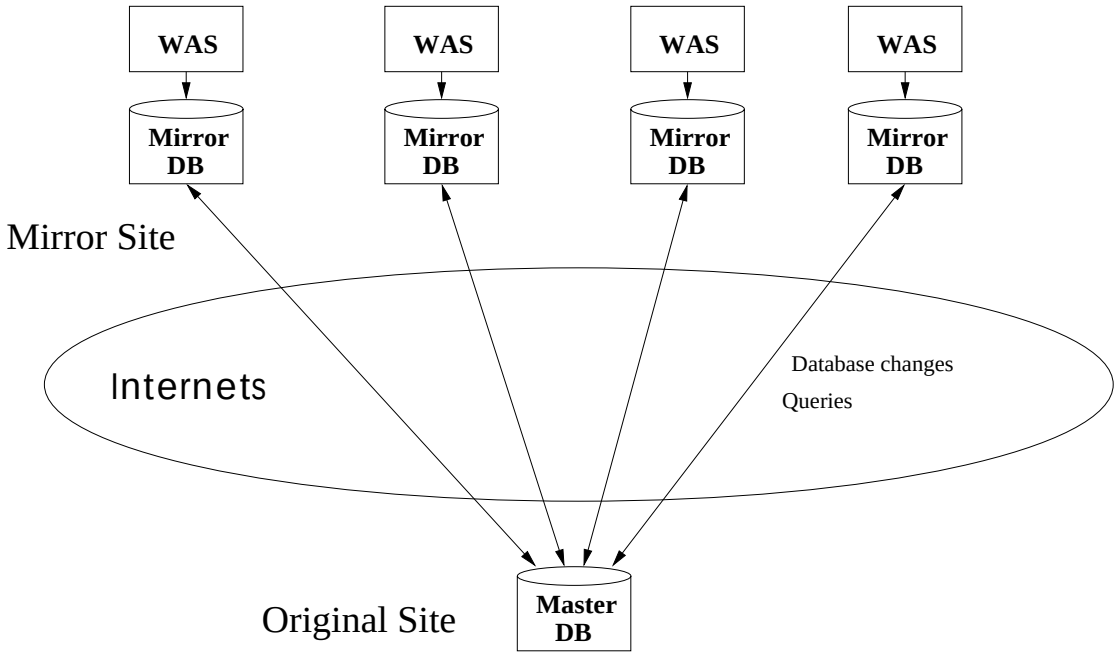

Figure 1: Typical Data Center-hosted Database-driven Web Site Architecture

DBCache can be a lightweight 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 most of e-commerce Web applications, the freshness of pages needs to be assured (within an acceptable threshold, such as 60 seconds) and asynchronous update propagation is used.

The interactions between the components in the typical data center-based are summarized as follows:

1. The requests are directed to a data center that is close to them based on network distance.
2. If the requests are read only and the data cache has all the required data, the requests are processed at the mirror site and the pages are generated dynamically and returned to the users.
3. Otherwise, the database queries are forwarded to the master database at the original site. After the WAS

receives the query results, it generates the pages dynamically and returns to the users.

4. The changes of database contents are periodically migrated from the master database to the mirror database.

3 Overview of the CachePortal Technology

In this section we give an overview of the CachePortal technology[6, 7] for enabling caching of dynamic contents generated by database-driven Web applications. We especially discuss the implementation options that impact the scalability, in terms of acceleration and content freshness.

3.1 System Architecture

CachePortal is a plug and play software that can be installed to accelerate Web site performance without any change on the existing system architecture and operations. For a Web site that utilizes CachePortal, the functionality of Web servers, application servers, and DBMSs are the same as described in Section 2. The two *CachePortal* software modules that are added into the infrastructure are as follows:

Sniffer: *Sniffer* is a software module that behaves as a wrapper to "sniff" the messages that pass through 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 a JDBC call, *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 a $URL/DBQueryMapping$.

Invalidator: *Invalidator* periodically polls the database log to monitor database content changes. The invalidator receives the log in the form of new and old value pairs for those tuples which have been inserted, deleted, and updated. Based on the tuple values and the $URL/DBQueryMapping$, *Invalidator* identifies the query results that 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 is used by a *Cache Manager* (if available) to determine the caching priority of each page.

3.2 Invalidation Process

In this section, we describe the invalidation process in `CachePortal`. We start with an example database and introduce relevant terminology. We then use a set of running examples to illustrate the proposed consolidated invalidation approach.

3.2.1 Concept of Polling Queries for Invalidation Checking

Let us assume that the database has the following two tables.

```
Car(maker, model, price)
Mileage(model, EPA)
```

Let us also assume that the following query, $Query1$, has been issued to produce a Web page, $URL1$:

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

Say, now, we observe based on the database log that a new tuple $(Toyota, Avalon, 25,000)$ is inserted into the table Car. since $Query1$ only uses table Car, we can check if the new tuple satisfies the condition stated in $Query1$. In this case, since the new tuple satisfies the query, $Query1$ has been impacted and consequently needs to be invalidated (or refreshed).

Next, let us assume the following query, $Query2$, has been issued to produce a Web page, $URL2$:

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

This query involves more than one table and has a join operation. Now we observe that a new tuple $(Toyota, Avalon, 25,000)$ is inserted into the table Car. Since $Query2$ uses two tables, we first check if the tuple satisfies the condition associated with Car. If it does not satisfy, we do not need to test the other condition and we know the new tuple does not impact $Query2$ and consequently $URL2$ does not need to be invalidated or refreshed.

If the tuple satisfies the condition associated with Car, we need to check the rest of the condition associated with the table $Mileage$ to determine whether or not $Query2$ has been impacted. To check whether or not the condition $Car.model = Mileage.model$ can be satisfied, on the other hand, we need to issue the following query, $Query3$, to the database:

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

If the result of $Query3$ is non-empty, the query result for $Query2$ needs to be invalidated. The queries, such as $Query3$, that are issued to determine if certain query results need to be invalidated are refereed as *polling queries*.

3.2.2 Query Types

We now introduce the terminology that is relevant to the proposed consolidated invalidation checking:

- *Query type:* A query type is the definition of a query. It is a valid SQL statement which may or may not contain variables. We denote a query type as $Q(V_1, \ldots, V_n)$, where each V_i is a variable that has to be instantiated by the application server before the query is passed to the DBMS.
- *Bound-query type:* A bound query type is a valid SQL statement which does not contain variables. We denote a bound query type as $Q(a_1, \ldots, a_n)$, where each a_i is a value instantiated for variable V_i. Queries that are passed by the application server to the DBMS are bound queries.
- *Query instances:* A query instance, is a bound query type with an associated request time stamp. We denote a bound query type as $Q^t(a_1, \ldots, a_n)$, where t is the time at which application server passed the request to the DBMS.

Therefore, multiple query instances can have the same bound query type; and, multiple bound query types may have the same query type. For example, if the invalidator observes the following three query instances, $Query4$, $Query5$, and $Query6$, in the log to generate user requested pages:

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

select maker, model, price
from Car
where maker = "Honda";

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

we can derive a common query type, $Query\_Type1$ as:

```
select maker, model, price
from Car
where maker = $var;
```

In fact, we can create a temporary table $Query\_Type1\_ID$ to represent the above three query instances, $Query4$, $Query5$, and $Query6$:

```
 QUERY_ID    QUERY_INSTANCE
---------- ------------------
  Query4        Toyota
  Query5        Honda
  Query6        Ford
---------- ------------------
```

There usually are a limited number of query types designed for a Web-based application. For example, the above query type could be associated with a query interface that allows users to specify car maker names to retrieve model and price information of the cars by the specified car maker.

3.2.3 Consolidated Invalidation

Now let us assume that there are three queries, $Query4$, $Query5$, and $Query6$, to be checked. Let us also assume that the following four tuples are inserted to the Car table:

```
(Acura, TL, 30,000)
(Toyota, Avalon, 25,000)
(Honda, Accord,  20,000)
(Lexus, LS430, 54,000)
```

We can create a temporary table $Delta_{car}$ for the content changes in the table Car. A single polling query, $Query10$, can return the names of the cached queries to be invalidated:

```
select Query_Type1.QUERY_ID
from Query_Type1, Delta
where Delta.Maker = Query_Type1.QUERY_INSTANCE;
```

In this example, $Query4$ and $Query5$ need to be invalidated.

4 Data Centers using `CachePortal` Technology

In this section, we describe the system architecture of data centers that benefit from the `CachePortal` technology. A representative system architecture is shown in Figure 2. Note that Figure 2 is similar to Figure 1 except the following changes:

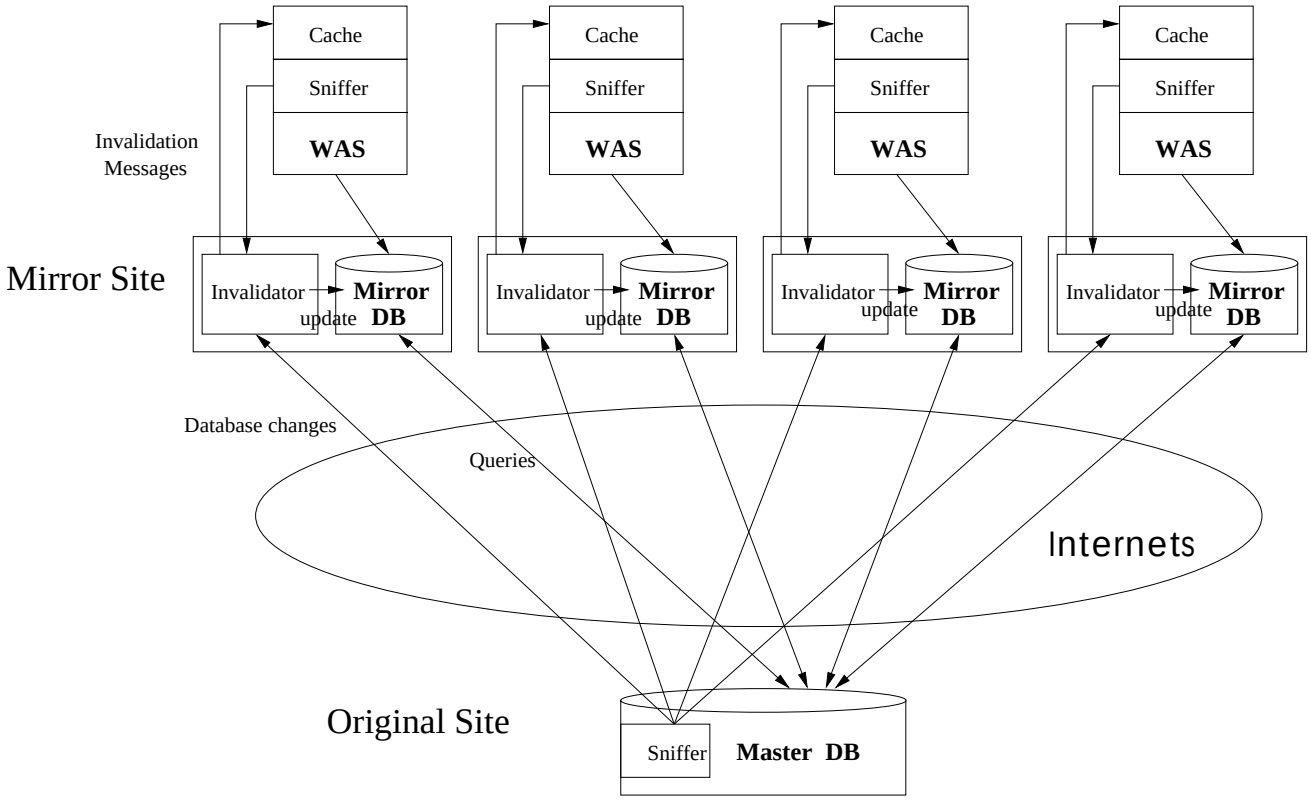

Figure 2: Data Center-hosted Database-driven Web Site Architecture with Deployment of `CachePortal`

- *sniffer:* sniffer components are installed at the WAS and the master database. The sniffer at the WAS is responsible for creating the mappings between the URLs (identifications of pages requested) and the query statements issued for the pages requested. The sniffer at the master database is responsible for tracking the database content changes.
- *Invalidator:* invalidator is responsible for the following two tasks:
  - retrieving the database content change log from the sniffer at the master database and propagating the changes to the mirror database.
  - performing invalidation checking based on the database content change log, URL and database query mapping, and the content in the mirror database. The detailed of the invalidation procedure is described in Section 3.

The interactions between the components in the data center-hosted Web site with edge caches are summarized as follows:

1. The requests are directed to the edge caches close to the users based on the network distance. If there is a cache hit, the requested page is returned to the user. Otherwise, request is forwarded to the WAS in the closest data center based on network distance.

2. If the requests are read only and the data cache has all the required data, the requests are processed at the mirror site and the pages are generated dynamically and returned to the users.

3. Otherwise, the database queries are forwarded to the master database at the original site. After the WAS receives the query results, it generates the pages dynamically and returns to the users.

4. The changes of database contents are periodically migrated from the master database to the mirror database.

Note that when compared with the system architecture in Section 2, the new architecture has the following advantages: (1) serving the content from edge caches is much faster than generating pages dynamically; and (2) because of majority of load is distributed to the edge caches, the WAS and the mirror database can generate requested pages faster. On the other hand, it has the overhead of invalidation. Note that the overhead of sniffing and forwarding requests from edges caches to the WAS is negligible.

5 Experiments

In this section, we evaluate the proposed system architecture and the performance gained through the deployment of `CachePortal`. We start with a description of the experimental environment followed by an analysis of the experimental results.

5.1 Experiment Setup

We used the following experimental setup in our evaluations:

- We used the two heterogeneous networks that are available in NEC's Cupertino facilities: one is used by the C&C Research Laboratories (referred to as CCRL) and the other one is used by *cacheportal.com* (referred to as CP). The users, Web servers, application servers, and mirror databases are located in CCRL while the master database is located in CP. The average number of hubs between CCRL and CP are 15. The average throughput measured for CCRL-CP and CCRL-CCRL connections are 84.7 and 482.4 Kilobytes/second respectively. The average round trip time on CCRL-CP and CCRL-CCRL connections are 321 and 0.2 ms respectively. To summarize, connectivity within the same network is substantially better than that across the Internet and there is network latency

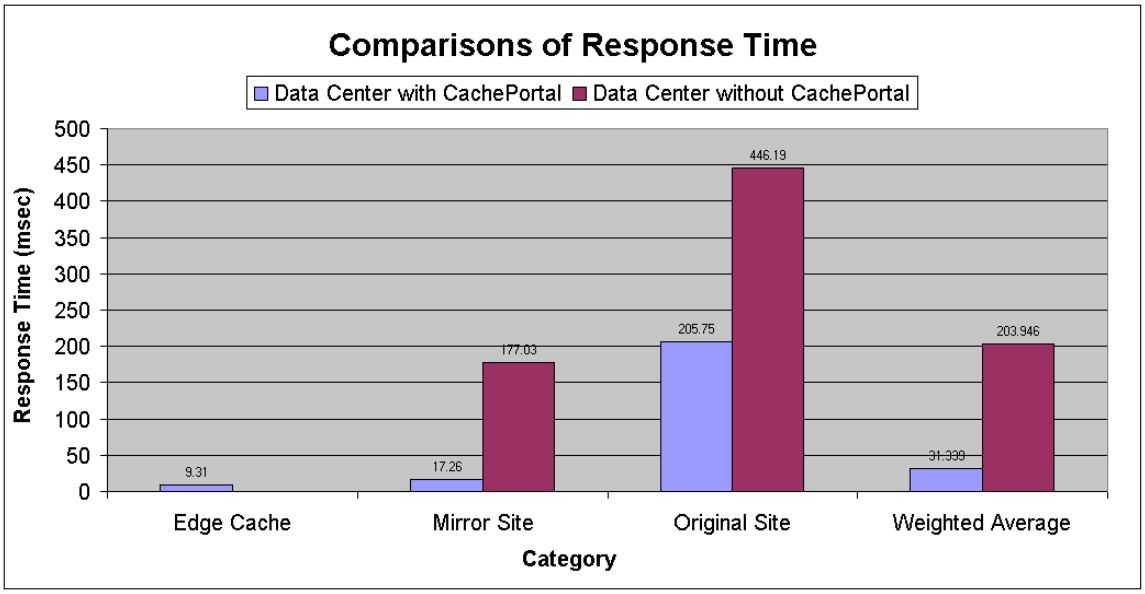

Figure 3: Comparisons of Response Time by Data Center with `CachePortal` and by Data Center without `CachePortal`

- Oracle 8i is used as the DBMS and is on a dedicated machine. The database contains 7 tables with 100,000 rows each. The database changes are propagated through the `DBMS_SNAPSHOT.REFRESH` DBMS built in procedure. The refresh option is set as `FAST` and `ON DEMAND`. The refresh cycle is set as 60 seconds.

- BEA WebLogic 6.1 is used for the WAS, and Apache is used as for both the edge cache servers and is located in the same machine as WAS. All of the machines are installed an Pentium III 700Mhz one CPU PCs running Redhat Linux 7.2 with 1GB of memory.

5.2 User Response Time Improvement

The first experiment we conducted is to measure the performance gain by deployment of `CachePortal`. Additional experiment-specific settings are as follows:

- Invalidation and view migration cycles: 60 seconds.

- Experiment duration: 300 seconds.

- Traffic patterns for the configuration in Figure 1: 90 percent of queries access the mirror database and 10 percents of traffic (database queries only).

- Traffic patterns for the configuration in Figure 2: 60 percent of traffic is served from the edge cache, 30

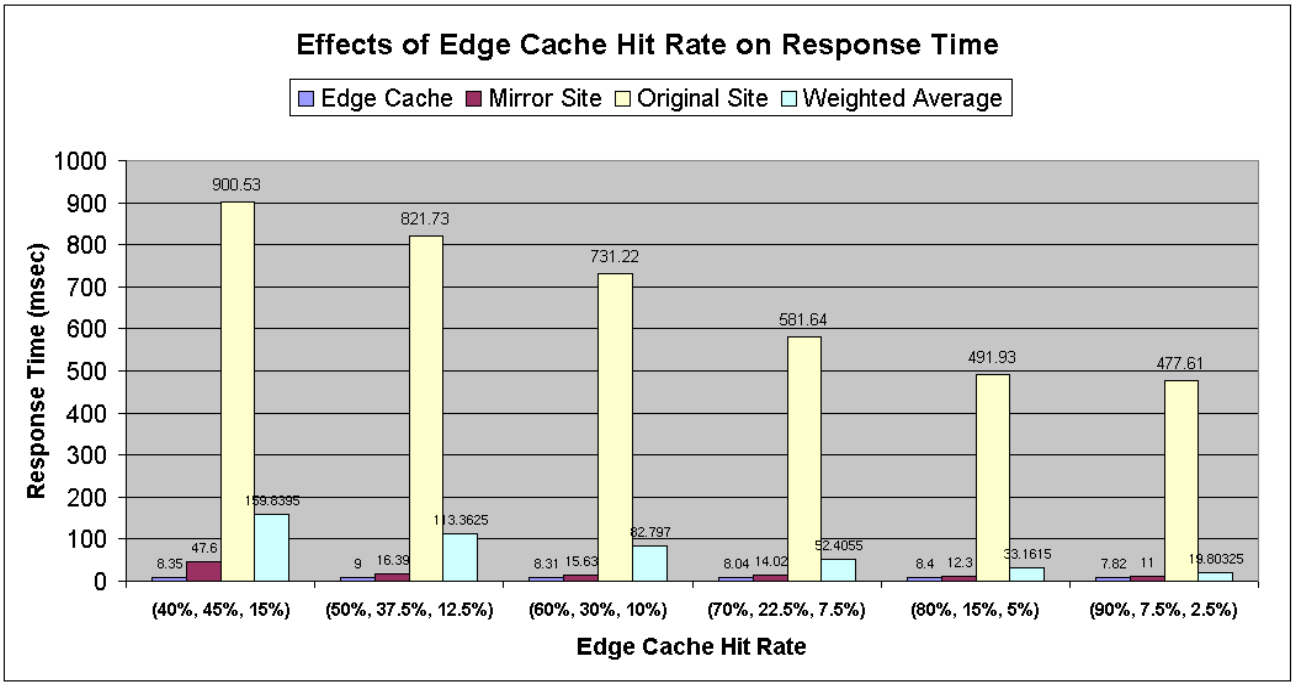

Figure 4: Effects of Query Rate on Response Time

percent of traffic is served from the mirror database, and 10 percent of traffic is served from the master database.

- The query rate to the whole system is 600 per minute and the update rate is 500 rows per minute.
- `CachePortal` module maintains 1,000,000 pages in the edge cache: 100 query types and 10,000 query instances per query type.

Figure 3 shows the comparisons of response times for two system configurations shown in Figures 1 and 2. Since the user requests are served at three locations: edge caches, edge suites, and master databases. We measure the response time for requests severed from these three locations separately and calculate their weighted averages. The average response time for data center-hosted Web sites with `CachePortal` is almost 7 times faster. Most of the performance gains are from

- Delivery from the edge caches (or any caches) is much faster than generating requested pages dynamically on demand.
- For the system configuration with `CachePortal`, the mirror database has lighter load than the system configuration without `CachePortal` since a large portion of traffic is handled by the edge caches.

- The overhead of migration/invalidation cost for the system configuration with `CachePortal` is much lighter than the system configuration without `CachePortal` (this will be described in detail in the next section).

The performance gains measured in the experiments are shown in Figure 3. Note that the system configuration without `CachePortal` does not have any edge cache to delivery the content. Thus, the request rate for its mirror database is three time higher than that of the system configuration with `CachePortal`. However, the actual load for the mirror database is higher than three times (based on its response time) due to the fact that it has higher overhead of database change migration.

Figure 4 shows the impacts of the query rates on response time for the system configuration with `CachePortal`. Note that the average response time increases slowly while the response time for the queries to the master database increases is fairly significant. This is because the queries to the master across networks is much more sensitive to both the network conditions and load.

5.3 Comparisons of Invalidation/Migration Overhead

Figure 5 show the comparisons of overhead associated with invalidation and update propagation. We set the traffic to the mirror databases of the two configurations as 300 requests per minutes and the update rate to the master database is 500 rows per minute. We measure the response times of two mirror databases when the invalidation and migration are enabled and disabled. Our observations are as follows:

- When the invalidation and migration are disabled, the response time for the system configuration with `CachePortal` is slight higher. This is due to the cost of checking if there is a hit at the edge cache and the forwarding if there is a miss.
- When the invalidation and migration are enabled, the response time for the system configuration with `CachePortal` is much lower. This is due to the cost of migrating database changes across networks is very expensive. On the hand, the database content synchronization of `CachePortal` is much lower. This is because that `CachePortal` moves the database update logs to the mirror site and performs the propagation locally instead of across the networks.

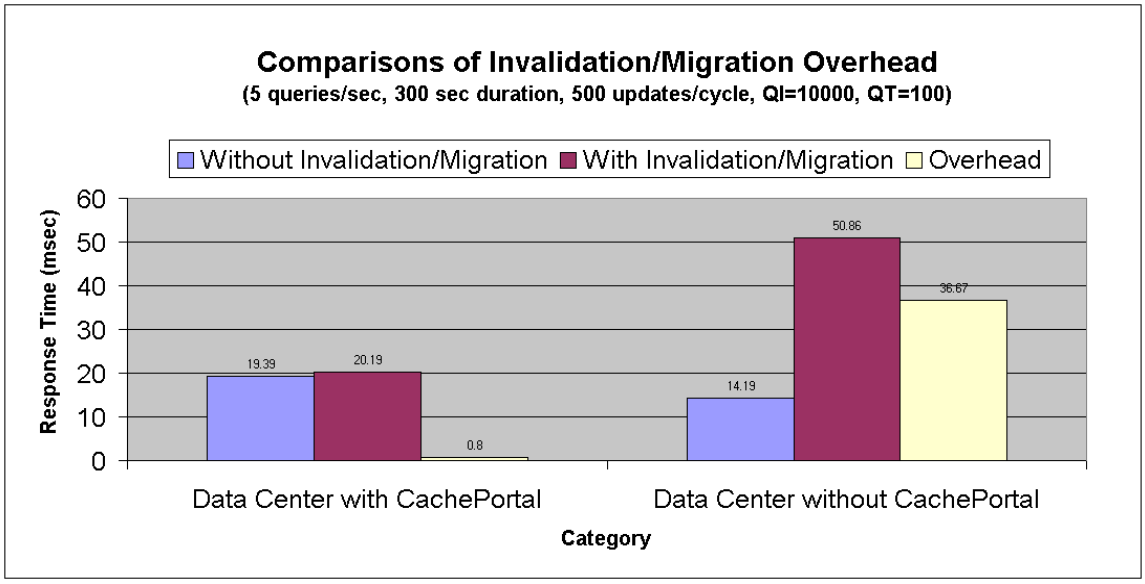

Figure 5: Comparisons of Overhead for Invalidation and Migration

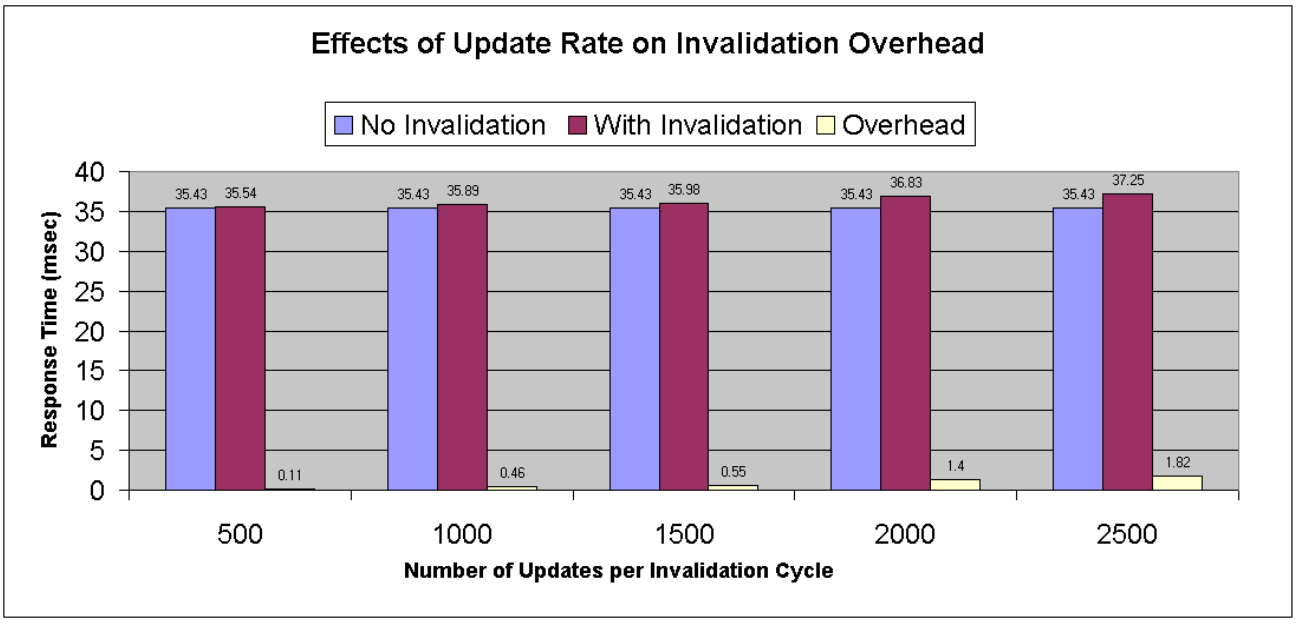

Figure 6: Effects of Update Rate on Invalidation Overhead

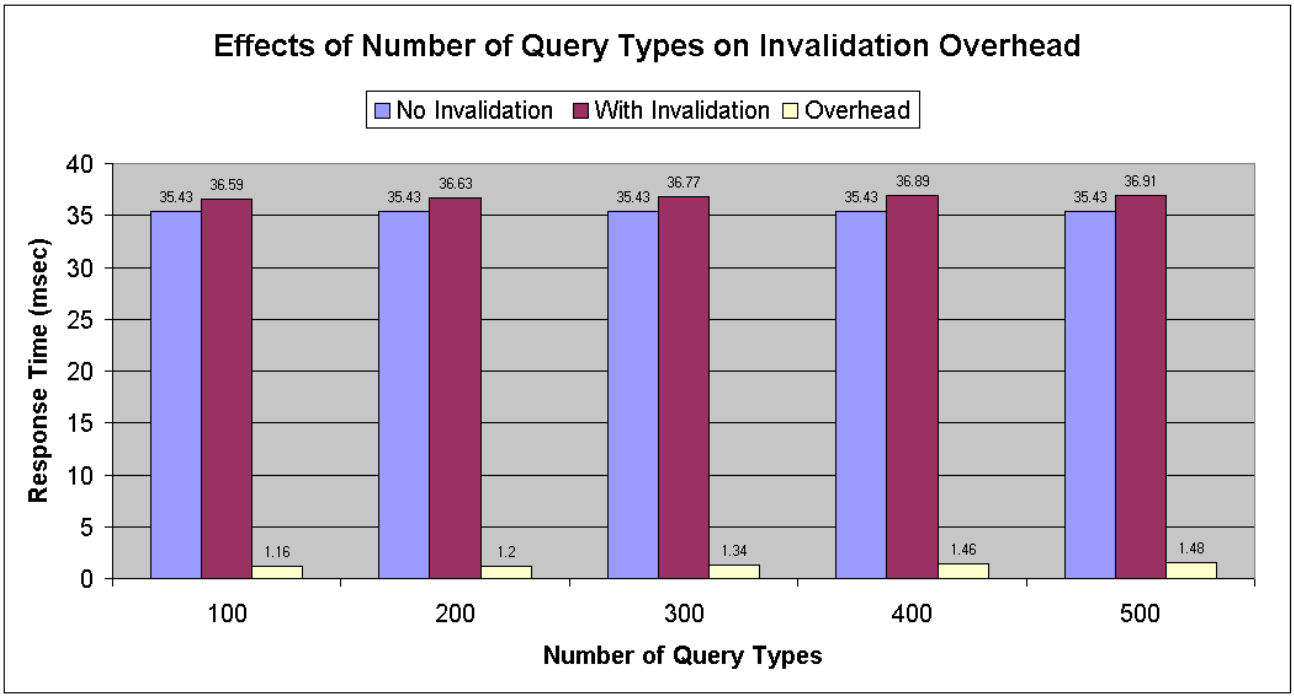

Figure 7: Effects of Number of Query Types on Invalidation Overhead

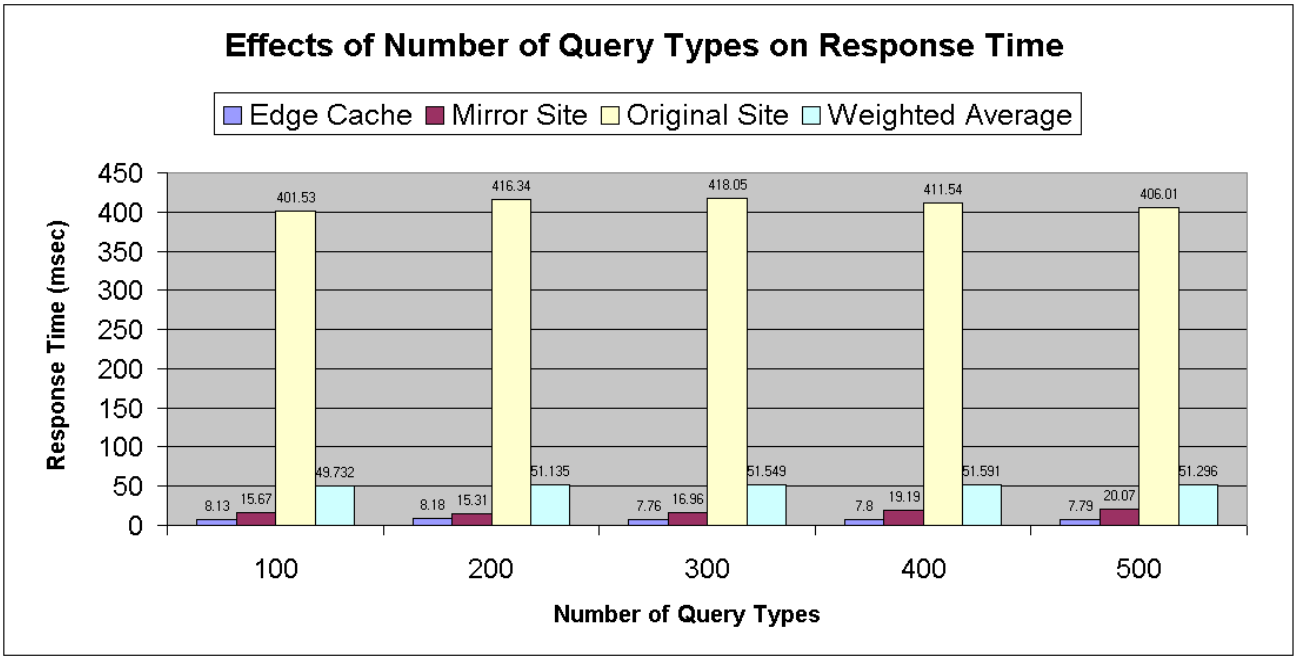

Figure 8: Effects of Number of Query Types on Response Time

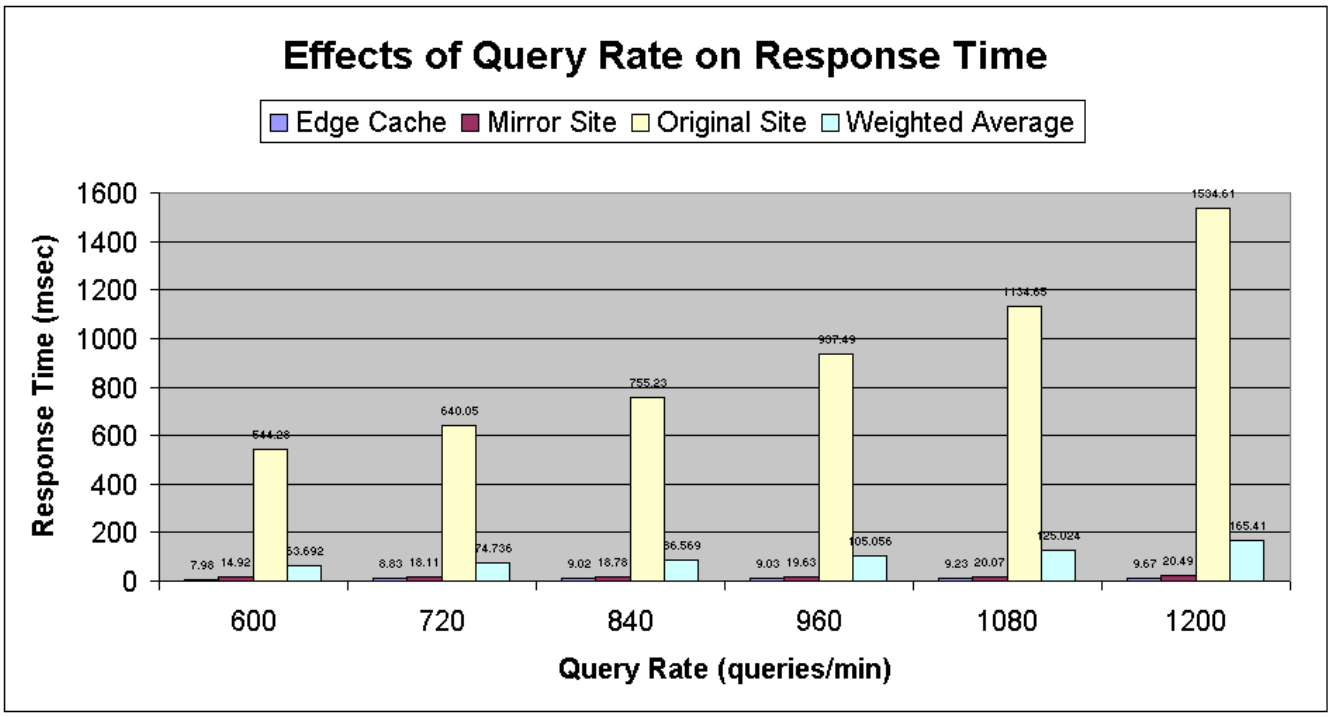

Figure 9: Effects of Hit Rate on Response Time

5.4 Evaluations of Scalability

Figures 6, 7, and 9 show the experimental results for evaluation of the scalability of our proposed solutions. We summarize the results as follows:

- Effects of update rate on invalidation overhead (shown in Figure 6): The update rate has impacts on the computing time of each polling query; however, the number of polling queries remains the same. Since the polling queries are submitted to the same machine and indexing schemes have been applied, the impact of update rate changes is very limited.
- Effects of number of query types on invalidation overhead (shown in Figure 7): The number of query types has impact on the number of polling queries need to be executed; however, the computation cost of each polling query remains the same. Since the polling queries are submitted to the same machine, the impact of update rate changes is very limited.
- Effects of number of query types on response time (shown in Figure 9): The number of query types has impact on the number of polling queries need to be executed. As the experimental results in Figure 7 indicate: the impacts of the number of query types on the overhead is really limited, the results in Figure 9 further confirm our observation since the response time does not get impacted by the invalidation overhead.

5.5 Effects of Cache Hit Edge Cache Hit Rates on Response Time

The next experiment we conducted is to examine the impact of cache hit on the response time for the system configuration with `CachePortal`. In this experiment, we assume that hit rate at the mirror database is 75 percent for the traffic hits the mirror database. The hit rate at the edge cache was varied from 40 percent to 90 percent. Figure 9 depicts that the average response time for the master database decreases sharply as the hit rates of the edge cache increase. However, the response times for the requests served at the edge caches and the mirror databases do not change significantly. The experimental results imply that the response time is very sensitive for the changes of traffic across networks. The figure quantifies the advantages of increasing the edge cache hit ration to improve response time.

6 Comparisons of Content Freshness

In this section, we discuss the issues associated with freshness. There are two ways to measure: *maximum age* and *average age* of the out dated information that the users may receive. We use the examples in Figure 10 to explain:

- Figure 10(a) shows a sequence of operations occurring in a data center-hosted distributed Web site. The synchronization between the mirror database and the master database is scheduled to take place every 60 seconds. And the synchronization takes 50 seconds to complete. In this example, the *maximum age* of the out dated content is 110 seconds (i.e. the content change occurs at $T = 0$ and the content gets refreshed at $T = 110$). And *average age* of the out dated content is 55 seconds (i.e. (55/2)+(60/2)).

- Figure 10(b) shows a sequence of operations occurring in a data center-hosted distributed Web site with deployment of `CachePortal`. The invalidation of cached contents is scheduled to take place every 60 seconds and the invalidation can be completed in 10 seconds. In this example, the *maximum age* of the out dated content is 70 seconds (i.e. the content change occurs at $T = 0$ and the content gets refreshed at $T = 70$). And *average age* of the out dated content is 35 seconds (i.e. (10/2)+(60/2)).

- Figure 10(c) shows another sequence of operations occurring in a data center-hosted distributed Web site with deployment of `CachePortal`. The invalidation of cached contents is scheduled to take place every 30 seconds and the invalidation can be completed in 8 seconds. In this example, the *maximum age* of the out dated content is 38 seconds (i.e. the content change occurs at $T = 0$ and the content gets refreshed at $T = 38$). And *average age* of the out dated content is 19 seconds (i.e. (8/2)+(30/2)).

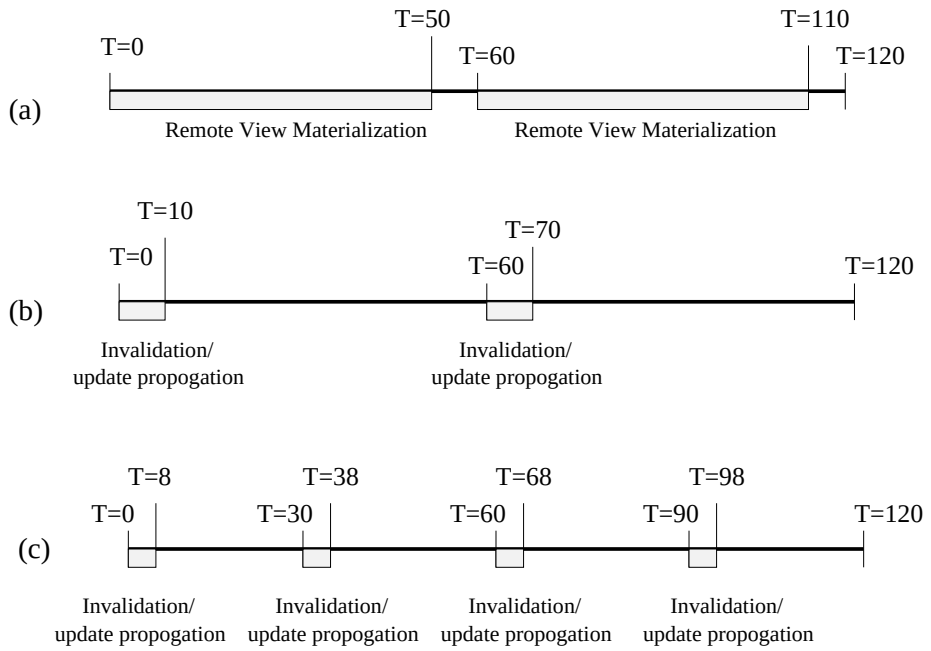

Figure 10: Examples for Calculating *Maximum Age* and *Average Age*

Now we will compare the maximum age and average age for the system configurations with and without `CachePortal`. We assume the database size is 100,000 rows per table and the database update rate is 50,000 rows per table per minute. In Figure 11, we show the results of experiments to evaluate the time required to propagate database changes (various numbers of rows) to the mirror databases of various sizes. Our observations are as follows:

- The larger the mirror database size is, the longer time is required for synchronization.
- The larger the number of rows needed to be propagated to the mirror database, the longer time is required for synchronization.

The time required to propagate 10,000 rows to a mirror database whose size is 100,000 rows is about 60 seconds (as indicated in our experiments in Figure 11). As a result, the system configuration without `CachePortal` can provide the freshness level in which the *Maximum Age* and *Average Age* of updated contents are 120 seconds and 60 seconds respectively.

Now let us measure the freshness level that can be supported by the system configuration with without `CachePortal`. Figure 12 shows the experimental results that we measure the relationships between update rates and invalidation cycle times. The number of cached pages is 1,000,000 and that belong to 100 query types

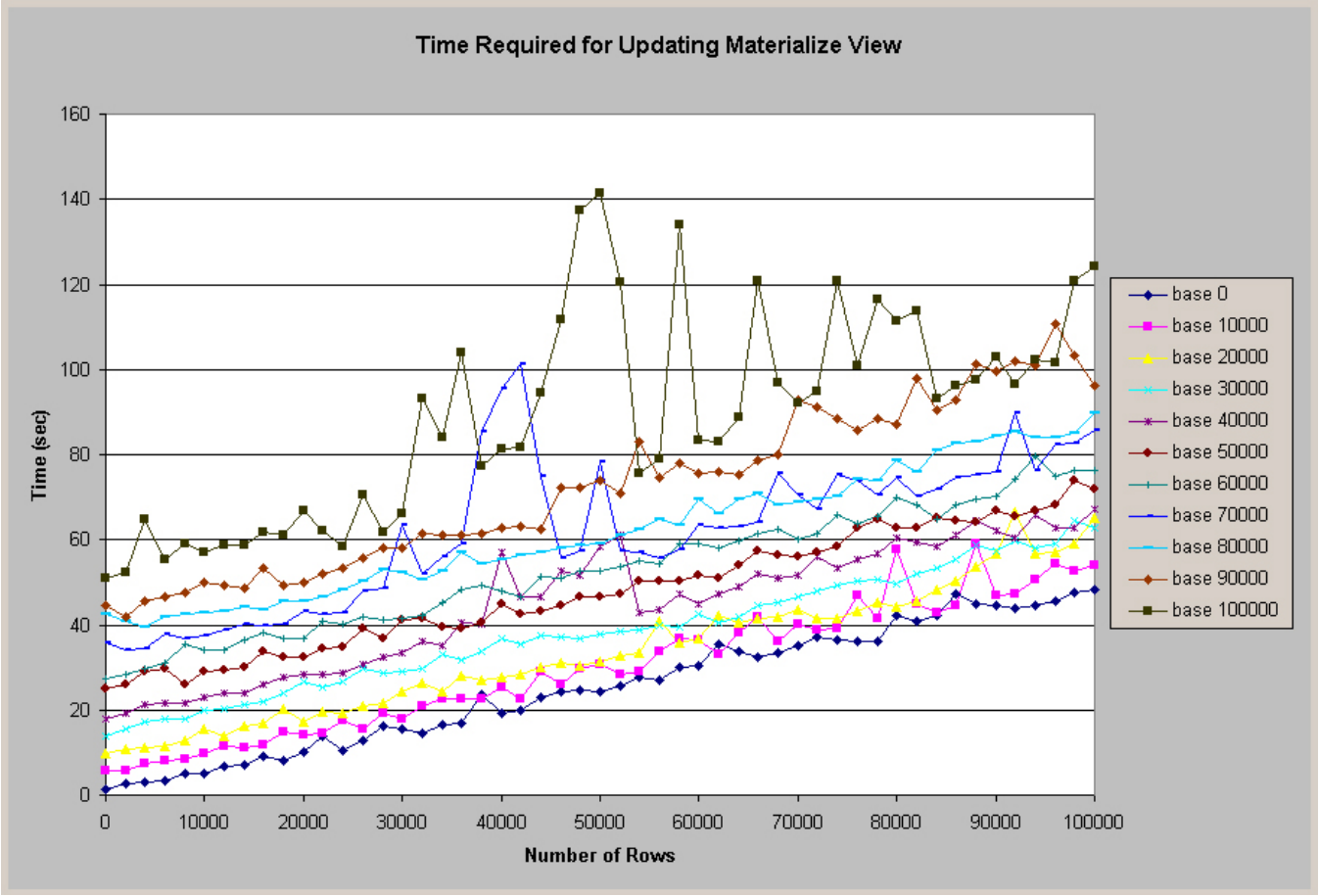

Figure 11: Typical Data Center-hosted Database Update Migration Cycles

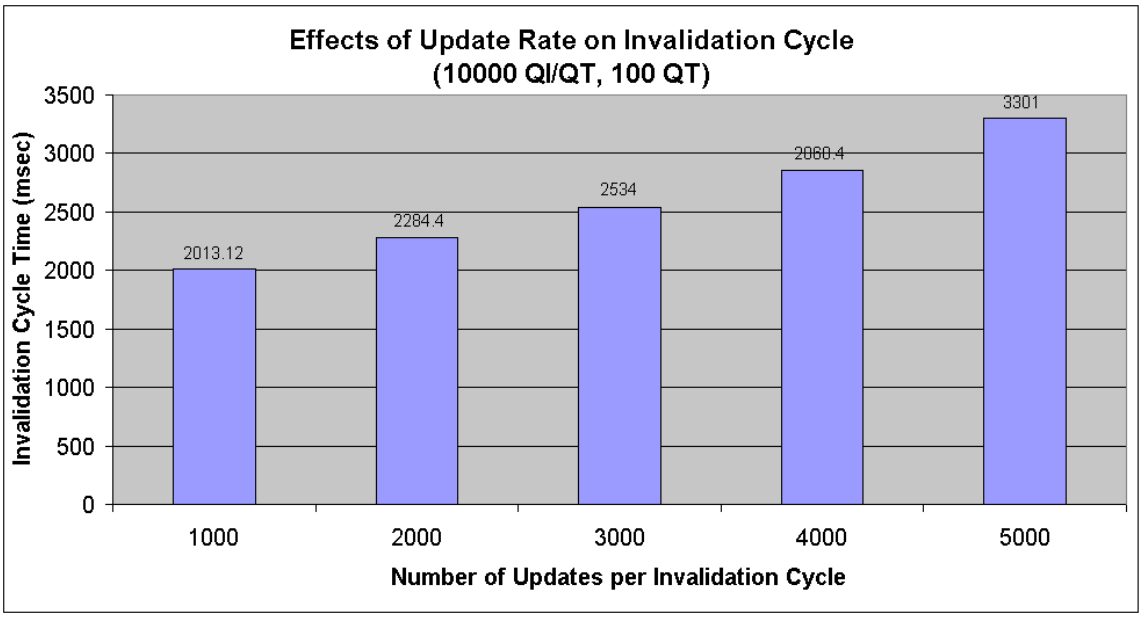

Figure 12: The Effects of Update Rate on Invalidation Cycles

with a uniform distribution of 10,000 query instances for query type. The sizes of the master database and the mirror database are 100,000 rows and the update rate is 10,000 rows per minute. Note that the update rate is considered high for e-commerce applications. WE use the high update rate to show the scalability. In the experiment, we measure that the cycle time required for invalidation is 3.3 seconds (including update propagation). The *Maximum Age* and *Average Age* of updated contents are 63.3 seconds and 31.65 seconds respectively, which are substantially better than 120 seconds and 60 seconds that can be supported in the standard data center-hosted Web sites. Furthermore, the system configurations with `CachePortal` still support an average of 7 times speed up on the user response time.

7 Related Work

Issues related to caching of dynamic data have received significant attention recently [9, 10]. Applying caching solutions for Web applications and content distribution has also received a lot of attentions in the Web and database communities[6, 11, 7, 12, 13, 14]. Dynamai [15] 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. [16, 17, 18] 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.

Other related works include [19, 20], where authors propose a diffusion-based caching protocol that achieves load-balancing, [21] which uses meta-information in the cache-hierarchy to improve the hit ratio of the caches, [22] which evaluates the performance of traditional cache hierarchies and provides design principles for scalable cache systems, and [8] which highlights the fact that static client-to-server assignment may not perform well compared to dynamic server assignment or selection.

SPREAD[23], 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 [23] focuses on static content and describes techniques to synchronize static content, which gets updated periodically, across Web servers. Therefore, in a sense, the invalidation messages travel horizontally across Web servers. Other works which study the effects of *invalidation* on caching performance are [24, 25]. Consequently, there has been various cache consistency protocol proposals which rely heavily on *invalidation* [23, 26, 27]. 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 front-end and edge cache servers as well as JDBC database access layer caches.

8 Concluding Remarks

Response time and scalability are essential to many Web applications. With availability and advance content delivery networks (CDN), many database-driven Web applications start to utilize data centers to host applications and database contents. This kind of Web site infrastructure has advantages of generating user requested pages from a location close to the users to avoid network latency. On the other hand, it has additional cost of database synchronization and invalidation. In this paper, we describe the deployment of NEC's `CachePortal` dynamic content caching technology on a database-driven Web site in a data center-based distribution infrastructure. The proposed system architecture has been prototyped and experimental evaluated. The experimental results show that our proposed architecture has advantages of both the scalability and performance gain up to 7 times while supporting higher freshness level in term of average age of update contents.

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

[4] Digital Island, Ltd. *Information available at* `http://www.digitalisland.com/`.

[5] Oracle Corp. http://www.oracle.com/.

[6] 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.

[7] Wen-Syan Li, K. Seluk Candan, Wang-Pin Hsiung, Oliver Po, Divyakant Agrawal, Qiong Luo Wei-Kuang Wayne Huang, Yusuf Akca, and Cemal Yilmaz. Cache Portal: Technology for Accelerating

Database-driven E-commerce Web Sites. In *Proceedings of the 2001 VLDB Conference*, Roma, Italy, September 2001.

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

[9] Fred Douglis, Antonio Haro, and Michael Rabinovich. HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, 1997.

[10] Ben Smith, Anurag Acharya, Tao Yang, and Huican Zhu. Exploiting Result Equivalence in Caching Dynamic Web Content. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, 1999.

[11] 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.

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

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] A. Heddaya and S. Mirdad. WebWave: Globally Load Balanced Fully Distributed Caching of Hot Published Documents. In *Proceedings of the 1997 IEEE International Conference on Distributed Computing and Sytems*, 1997.

[20] A. Heddaya, S. Mirdad, and D. Yates. Diffusion-based Caching: WebWave. In *Proceedings of the 1997 NLANR Web Caching Workshop*, 1997.

[21] M.R. Korupolu and M. Dahlin. Coordinated Placement and Replacement for Large-Scale Distributed Caches. In *Proceedings of the 1999 IEEE Workshop on Internet Applications*, 1999.

[22] Renu Tewari, Michael Dahlin, Harrick M. Vin, and Jonathan S. Kay. Design Considerations for Distributed Caching on the Internet. In *Proceedings of the 19th International Conference on Distributed Computing Systems*, 1999.

[23] P.Rodriguez and S.Sibal. Spread: Scaleable platform for reliable and efficient automated distribution. In *Proceedings of the 9th World-Wide Web Conference*, pages 33–49, Amsterdam, The Netherlands, June 2000.

[24] P. Cao and C. Liu. Maintaining Strong Cache Consistency in the World Wide Web. *IEEE Transactions on Computers*, 47(4), 1998.

[25] J. Gwertzman and M. Seltzer. World-Wide Web Cache Consistency. In *Proceedings of 1996 USENIX Technical Conference*, pages 141–151, San Diego, CA, USA, January 1996.

[26] H. Yu, L. Breslau, and S. Shenker. A Scalable Web Cache Consistency Architecture. In *Proceedings of the ACM SIGCOMM'99 Conference*, Boston, MA, USA, September 1999.

[27] D. Li, P. Cao, and M. Dahlin. WCIP: Web Cache Invalidation Protocol, 2000. http://www.ietf.org/internet-drafts/draft-danli-wrec-wcip-00.txt.