Engineering and Hosting Adaptive Freshness-Sensitive Web Applications on Data Centers

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

NEC Laboratories America, Inc.
10080 North Wolfe Road, Suite SW3-350, Cupertino, California 95014, USA
Email:wen@sv.nec-labs.com Tel:408-863-6008 Fax:408-863-6099

ABSTRACT

Wide-area database replication technologies and the availability of content delivery networks allow Web applications to be hosted and served from powerful data centers. This form of application support requires a complete Web application suite to be distributed along with the database replicas. A major advantage of this approach is that dynamic content is served from locations closer to users, leading into reduced network latency and fast response times. However, this is achieved at the expense of overheads due to (a) invalidation of cached dynamic content in the edge caches and (b) synchronization of database replicas in the data center. These have adverse effects on the freshness of delivered content. In this paper, we propose a *freshness-driven adaptive dynamic content caching*, which monitors the system status and adjusts caching policies to provide content freshness guarantees. The proposed technique has been intensively evaluated to validate its effectiveness. The experimental results show that the freshness-driven adaptive dynamic content caching technique consistently provides good content freshness. Furthermore, even a Web site that enables dynamic content caching can further benefit from our solution, which improves content freshness up to 7 times, especially under heavy user request traffic and long network latency conditions. Our approach also provides better scalability and significantly reduced response times up to 70% in the experiments.

Categories and Subject Descriptors

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

General Terms

Performance, Reliability, Experimentation

Keywords

dynamic content, web acceleration, freshness, response time, network latency, database-driven web applications

1. INTRODUCTION

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

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

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

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

How to architect Web sites and tune caching policies dynam-

WWW2003, May 20–24, 2003, Budapest, Hungary.
ACM 1-58113-680-3/03/0005.

ically to serve fresh content in short response time is a complex problem. In this paper, we focus on the issue of how to maintain the best content freshness for a given set of user request rate, database update rate, and network latency parameters. We propose an *freshness-driven adaptive dynamic content caching* technique that monitors response time and invalidation cycle as feedback for dynamic adjustment of caching policy. By considering the trade-off of invalidation cycle and response time, it maintains the best content freshness to the users.

The rest of this paper is organized as follows. In Section 2 we describe a typical data center architecture for hosting database-driven Web application. In Section 3, we describe how to enable dynamic content caching for data center-hosted Web applications. In Section 4 we address some limitation of these architectures in supporting fresh dynamic content. In Section 5, we give an overview of our proposed solution for assuring content freshness. In Section 6 we describe the dynamic content invalidation schemes in the scope of NEC's CachePortal technology. In Section 7, we describe dependency between request response time and invalidation cycles. In Section 8 we describe how to engineer adaptive freshness-sensitive Web applications on data centers that balances response time and invalidation cycles. We also present experimental results that evaluate effectiveness of our proposed technique for ensuring content freshness as well as its benefit in accelerating request response. In Section 9 we summarize related work and compare with our approach. In Section 10 we give our concluding remarks.

2. DATA CENTER ARCHITECTURE FOR HOSTING WEB APPLICATIONS

A typical data center architecture for hosting Web applications requires a complete e-commerce Web site suite to be distributed along with the database replicas. The figure shows a configuration in which the WS/AS/DBMS *suite* is installed in network edges to serve non-transaction requests which require accesses to only read-only database replicas. In order to distinguish between the asymmetric functionality of master and slave DBMSs, we refer the mirror database in the data center as data cache or DB Cache. DB Cache can be a lightweight DBMS without the transaction management system and it may cache only a subset of the tables in the master database. Updates to the database are handled using a master/slave database configuration: all updates and transactions are processed at the master database at the origin site.

The scheme for directing user requests to the closest server is the same as what typical CDNs deploy. The interactions between these software components for hosting Web applications on data centers are summarized as follows:

1. A request is directed to a nearby data center based on network proximity.
2. If the request is non-transaction and the DB Cache has all the required data, the request is processed at the data center and the result page is generated dynamically and returned to the user.
3. Otherwise, the request is forwarded to the master database at the original site.
4. The changes of database contents are periodically migrated from the master database to the DB Cache at the data centers.

Either a pull- or a push-based method can be used to synchronize DB Cache with the master database. A 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 content freshness needs to be assured at a much higher standard and asynchronous update propagation is used. Although its popularity, this system architecture has the following drawbacks:

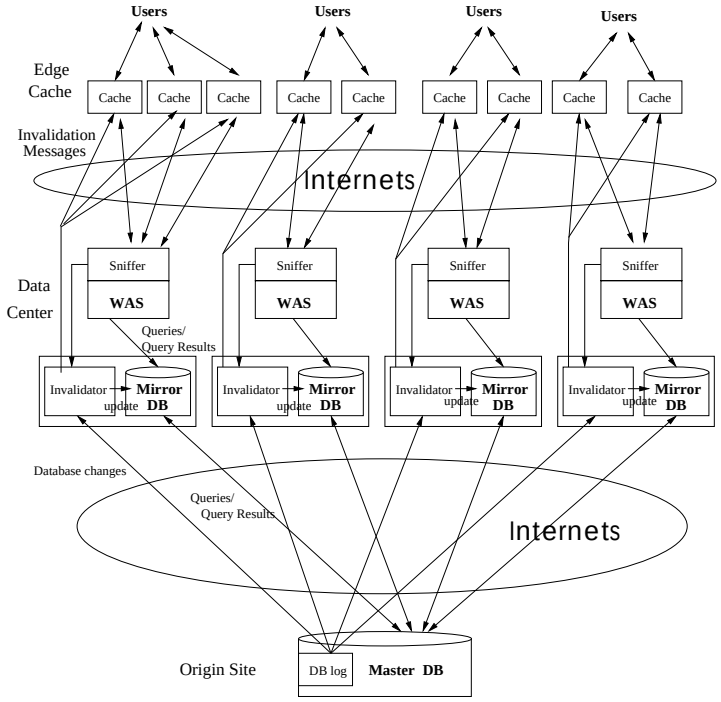


Figure 1: Data Center-hosted Database-driven Web Site with Deployment of CachePortal

- all dynamic content pages have to be generated at the data centers or the origin Web site as user requests arrive;
- only network latency between the data centers and origin Web site is reduced; the network latency between users and the data centers remain the same; and
- synchronization is pre-scheduled and the dynamic content pages may be generated based on outdated database content between two synchronizations.

3. PROPOSED SYSTEM ARCHITECTURE

In [3], we presented a theoretical framework for invalidating dynamic content. These works introduce two new architectural components:

- *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: (1) retrieving the database content change log from the sniffer at the master database and propagating the changes to the mirror database; (2) performing invalidation checking for the cached pages at the edge caches based on the database content change log, URL and database query mapping, and the content in the mirror database.

Note that the knowledge about dynamic content is distributed across multiple servers. In contrast to the other approaches [4, 5] which assume such mappings are provided by system designers, the construction of mapping between the database content and the corresponding Web pages is *automated*.

In this paper, we build on these results by developing a novel system architecture that accelerates data center-hosted Web applications through deployment of dynamic content caching solutions. The proposed system architecture is shown in Figure 1. It is similar to the typical data center architecture except the two new software modules. The interactions between the components in the data center-hosted Web site with edge caches are as follows:

1. A request is directed to the edge cache closest to the user based on the network proximity. If there is a cache hit, the requested page is returned to the user. Otherwise, the request is forwarded to the WAS in the closest data center.
2. If the request is non-transaction and the DB Cache has all the required data, the request is processed and the page is generated dynamically and returned to the user.
3. Otherwise, the request is forwarded to the master database at the original site.
4. The changes of database contents are periodically reflected from the master database at the origin Web site to the DB Cache at the data center for synchronization and invalidation. In our implementation, database update log is scanned every second and the new log is copied to the data center.
5. The invalidator reads the unprocessed database update log and performs the invalidation checking and synchronization tasks. These tasks are done as an invalidation/synchronization cycle. After one cycle is completed, the invalidator starts the next cycle immediately. Since the log scanning and invalidation/synchronization are performed in parallel, the invalidator does not need to wait for the completion of log scanning. We visualize the relationship among log scanning, edge cache invalidation, and DB Cache synchronization in Figure 2.
6. The dynamic content pages generated are cached in the edge caches.

As shown in Figure 1, a data center may deploy multiple edge caches depending on the user request rates. Dynamic content pages in the edge caches may be generated by the database content in the DB Cache or the master database. The IP addresses of edge caches and URL strings of cached pages are tracked by the invalidator for invalidation if it is applied.

3.1 System Parameters

The parameters that have an impact on the resulting freshness of delivered contents as follows:

Response time at edge cache servers: This is the round trip time for requests that are served by an edge cache server (as a result of a cache hit). The response time from edge caches is expected to be extremely fast.

Response time at the data center: This is the round trip time for requests that are served by a data center (as a result of an edge cache miss and DB Cache hit). The response time from the data center is expected to be fast, but is impacted by the request rate at the data center. The network latency between the end users and the data center has limited impact to the response time since the network latency is low.

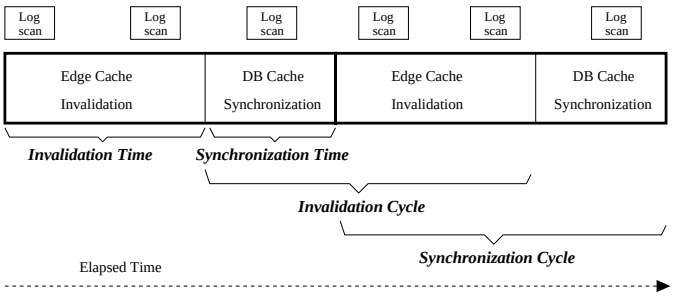


Figure 2: Visualization of the Relationship Among Log Scanning, Edge Cache Invalidation, and DB Cache Synchronization

Response time at origin Web sites: This is the round trip time for requests that are served at the origin Web servers (as a result of an edge cache miss and a DB Cache miss).

Invalidation time: This is the time required to process invalidation checks for all the pages in the edge caches.

Synchronization time: This is the time required to propagate database updates from the master database log to the DB Cache in the data center.

Invalidation cycle: This is the time required to process invalidation checks for all the pages in the cache servers **plus** propagating database updates. Note that the invalidation process requires the database update log and synchronization of the master database and DB Cache.

Synchronization cycle: The synchronization time could be shorter than the invalidation time; however, the synchronization cycle is the invalidation time plus synchronization time since synchronization and invalidation are interleaved.

Note that the invalidation time and synchronization time vary very little from one cycle to the other, the length of synchronization cycle and invalidation cycle are almost of the same (as shown in Figure 2) although the length of the synchronization time and invalidation time are different.

3.2 Advantages

A data center architecture with deployment of CachePortal has the following advantages:

- serving cached dynamic content pages is much faster than generating pages on demand;
- edge caches are deployed close to the end users; consequently, the network latency between end users and data centers is eliminated;
- since a bulk of the load is distributed to the edge caches, the WAS and the DB cache at the data center have lighter loads and thus they can generate requested pages faster.
- freshness of pages cached in the edge caches and those generated at the data center on demand is assured to be not older than the invalidation/synchronization cycle.

Since the freshness that can be provided depends on the length of the invalidation/synchronization cycle, on the other hand, this parameter has to be carefully adjusted. In the next section, we discuss challenge in greater detail.

4. ISSUES IN CONTENT FRESHNESS

In Figure 3, we illustrate response time and content freshness of three system architectures. For Web sites that do not deploy any dynamic content caching solution (Figure 3(a)), all pages need to be

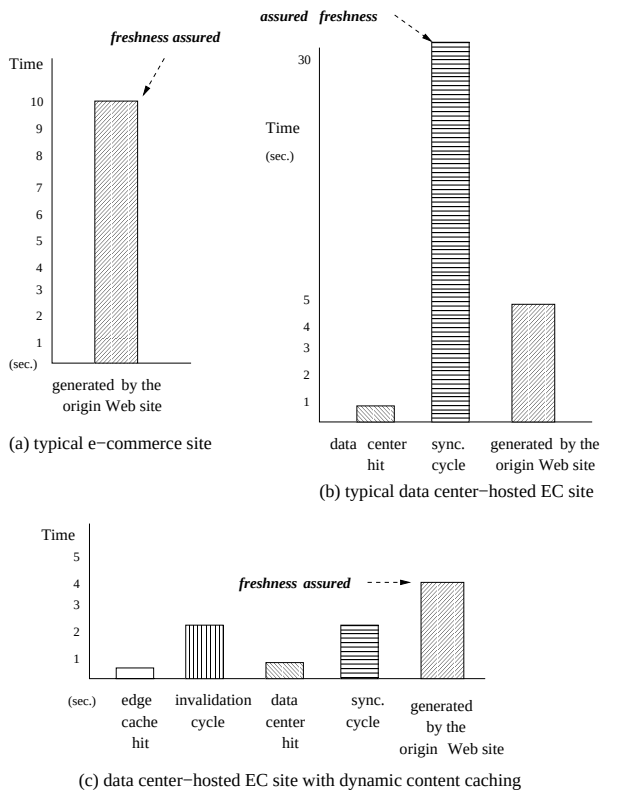


Figure 3: Response Time and Freshness of Delivered Content by Various Existing System Architectures

dynamically generated at the origin Web site. The content freshness such a system can assure is the response time at the Web sites. For example, if the average response time at a Web site is 10 seconds, even every Web page is generated based on up to date information in the database, the assured content freshness (i.e., *age*) of this Web site is 10 seconds since the database content may change after the Web page is generated.

For data center-hosted applications, as described in Section 2, response time can be improved by caching database content at the data centers (benefiting from lower network latency). However, in this case, database content must be synchronized in a timely manner so that pages are not generated based on outdated DB Cache content. The content freshness assured by this system architecture, then, is the maximum value among (1) DB Cache synchronization cycle, (2) response time at the data center, and (3) response time at the master database. For example, a DB Cache is set to be synchronized every minute (Figure 3(b)) and contents are served at both the data center and the origin Web site. Although the response time at the data center and the origin Web site could be as fast as less than 1 second and few seconds, respectively, the DB Cache content may be out of synchronization as long as 30 seconds. Therefore, the assured freshness of the delivered content is 30 seconds; even the average response time at the data center could be as low as few seconds.

For a data center that deploys dynamic content caching solutions, as shown in Figure 1, requested pages may be delivered from the edge cache servers; or dynamically generated at the data center or the origin Web site. The dynamic pages in the edge caches need to be invalidated and the content in the DB Cache needs to be synchronized periodically to ensure the freshness of delivered content. In this architecture, the content freshness that can be assured is the *maximum* of (1) the response time from origin Web sites; (2) the response time from edge caches; (3) the response time at the data center; (4) the edge cache invalidation cycle; and (5) the DB Cache synchronization cycle. Note that since the response time from the edge caches and data centers is much lower than the response time from the origin Web site, the invalidation cycles for the edge cache content, and the synchronization cycle of DB Cache, the content freshness that can be assured is the maximum of (1) the response time at origin Web sites; (2) the invalidation cycle of edge cache content; and (3) the synchronization cycle of DB Cache content. This is illustrated in Figure 3(c).

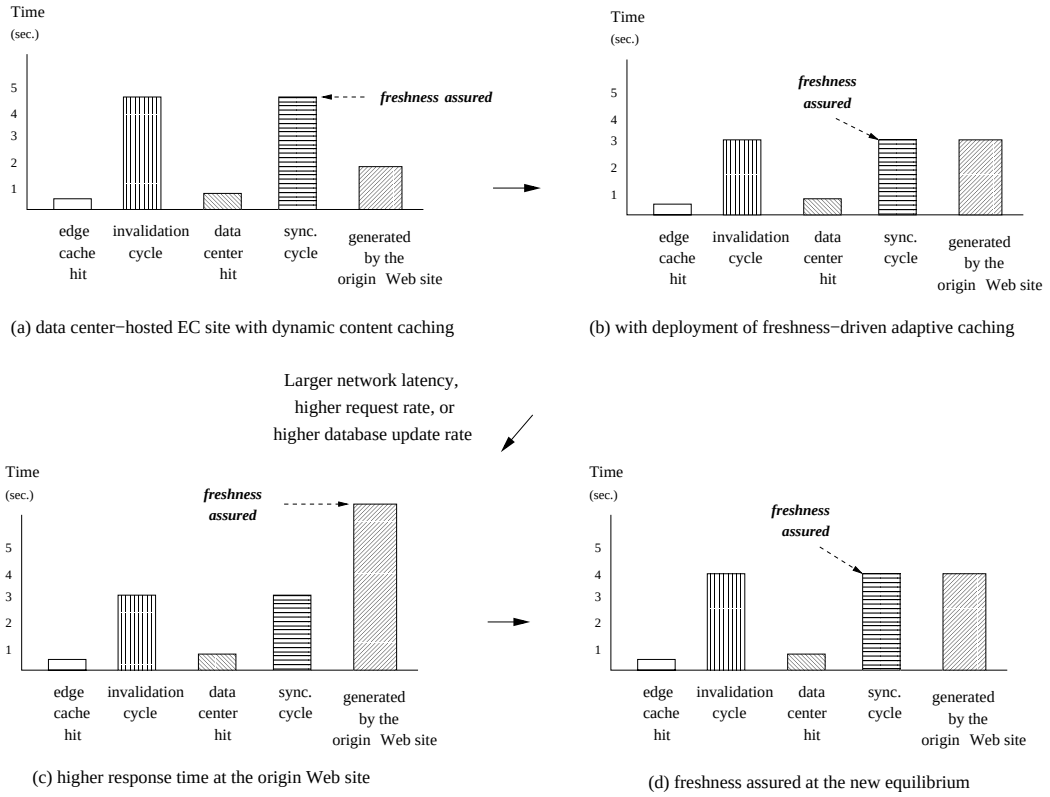
As we can see in Figure 3, the system architectures of a typical Web site and data center-hosted Web applications are not suitable for applications that require assurance on the response time and it is difficult to estimate the TTL (Time To Live).

5. SOLUTION OVERVIEW

In Figure 3(c), we show a system configuration that caches few pages or database content. As a consequence, it is expected to have a short invalidation cycle and DB Cache synchronization time; at the expense of slow response time at the origin Web site when there is a cache miss at the edge caches and DB Cache. On the other hand, the system configuration in Figure 4(a) has a different caching policy that caches a large number of Web pages at edge caches and/or more database content at the DB Cache. This configuration provides fast response time: when there is a cache hit, the response time is naturally very fast; when there is a cache miss, the response is still reasonable as the Web site has lighter work load since most of the requests are served directly from the edge cache servers or DB Caches. However, this configuration and its caching policy has potentially low content freshness since it will take longer time to complete the necessary invalidation check for all the pages in the edge caches and to synchronize the DB Caches in the data centers.

In this paper, we propose a *freshness-driven adaptive dynamic content caching* technique. The proposed technique aims at maintaining the best content freshness that a system configuration can support and assure. Our technique does not blindly attempt to maintain the lowest average response time nor average content freshness. Instead it dynamically tunes the caching policy: the hit rate is tuned to a point where the response time from the origin site and invalidation cycle for the cached pages are equivalent as shown in Figure 4(b). At this point, the freshness of requested pages can be ensured to be at the highest level.

For the configuration in Figure 4(b), the content freshness that can be assured is equal to the length of the the edge cache invalidation cycle (or the DB Cache synchronization cycle). Given that the response time at the origin Web site is lower, the proposed freshness-driven adaptive caching would tune its caching policy by decreasing the number of cached pages and database content so that the edge cache invalidation cycle and DB Cache synchronization cycle get shorter accordingly. The number of cached pages and database content are decreased until the response time at the origin Web site is close to the invalidation cycle (and synchronization cycle) as shown in Figure 4(b). At this equilibrium point, the assured freshness of delivered content is optimal. In addition, the response time at the Web site is also assured. Thus, in Figure 4(b), the freshness of dynamic content and response time for the Web site are assured to be less than 3 seconds.



(a) data center−hosted EC site with dynamic content caching

(b) with deployment of freshness−driven adaptive caching

(c) higher response time at the origin Web site

(d) freshness assured at the new equilibrium

Figure 4: Response Time and Freshness of Delivered Content with Deployment of Freshness-driven Adaptive Caching

The equilibrium point in Figure 4(b), however, may change when the values of the influential system parameters change. If network latency, database update rates, and user request rates increase, the response time at the origin Web site will increase accordingly (Figure 4(c)). To reduce the response time at the origin Web site, the freshness-driven adaptive caching will increase the number of cached pages at the edge cache and/or database content at the DB Cache. Consequently, the request rate at the origin Web site would be reduced (so would be the response time) at the expense of invalidation cycle and synchronization cycle. Note that the equilibrium point in Figure 4(d) is higher than the equilibrium point in Figure 4(b).

The proposed freshness-driven adaptive caching has various advantages. First, it yields and assures the best content freshness among these system architectures. Second, it also yields fast response times. Third, it provides assurance for both freshness and response time as follows:

- the system does not serve the content that is older than the *assured freshness*; and
- the system does not serve the content slower than the *assured response time*.

6. INVALIDATION CHECKING PROCESS

In this section, we describe the invalidation schemes used in the proposed system architecture. Assume that the database has the following two tables: `Car(maker, model, price)` and `Mileage(model, EPA)`. Say that the following query $Query1$ has been issued to produce a Web page, $URL1$:

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

Now a new tuple $(Toyota, Avalon, \$25000)$ is inserted into the table Car. Since $Query1$ accesses two tables, we first check if the new tuple value can satisfy the condition associated with only the table Car stated in $Query1$. If it does not satisfy, we do not need to test the other condition and we know the new insert operation does not impact the query result of $Query1$ and consequently $URL1$ does not need to be invalidated or refreshed.

If the newly inserted tuple does satisfy the condition associated with the table Car, we cannot determine whether or not the query result of $Query1$ has been impacted unless we check the rest of the condition associated with the table $Mileage$. To check whether or not the condition $Car.model = Mileage.model$ can be satisfied, we need to access the table $Mileage$. To check this condition, we need to issue the following query, $Query2$, to the database:

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

If the result of $Query2$ 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 referred as *polling queries*.

Now assume that we observe the following three queries, $Query3$ and $Query4$, in the URL/database query mapping to generate user requested pages:

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

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

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

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

Therefore, multiple query instances can have the same bound query type; and, multiple bound query types may have the same query type. We can create a temporary table `Query_Type` to represent the above two query instances as follows:

```
 QUERY_ID    QUERY_INSTANCE
---------- ------------------
  Query3         Honda
  Query4         Ford
---------- ------------------
```

Let us also assume that the following four tuples are inserted to the database:

```
(Acura, TL, $30000)
(Honda, Accord,  $20000)
(Lexus, LS430, $54000)
```

A temporary table `Delta` to represent the above three tuples. We can consolidate a number of invalidation checks into more compact form through transformation. For example, a single polling query, $Query5$, can be issued as follows:

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

$Query\_Type.QUERY\_ID$ is a list of query results that need to be invalidated. In this example, $Query3$ will be invalidated. For details of the invalidation scheme, please see [6].

We summarize important characteristics of the consolidated invalidation checking schemes as follows: (1) the invalidation cycle is mainly impacted by the number of cached query types since it determines the number of polling queries executed for invalidation checking; and (2) database update rates and the number of query instance per query type would have relatively lower impact on the invalidation cycle since they only increase query processing cost.

7. DEPENDENCY BETWEEN RESPONSE TIME AND INVALIDATION CYCLE

In this section, we examine the dependency between the response time and the invalidation cycle length. We have conducted experiments to verify this dependency. We first describe the general experiment setup that consists of Web servers, application servers, DBMS, and network infrastructure that are used in the experiments.

7.1 General Experimental Setting

We used two heterogeneous networks that are available in the NEC's facility in Cupertino, California: one is used by the C&C Research Laboratories (referred to as CCRL) and the other one is used by *cacheportal.com* (referred to as CP). Users, edge cache servers, Web server, application server, and DB Caches are located in the CP network while the master databases are located in the CCRL network. The average round trip time on the CCRL-CP connections is around 250 ms while the round trip time within the same network is negligible. To summarize, connectivity within the same network is substantially better than that across the Internet and there is notable network latency.

BEA WebLogic 7.0 is used for the WAS. Oracle 9i is used as the DBMS. The database contains 7 tables with 1,000,000 rows each. The database update rate is 600 rows per table per minutes. The Squid server is modified to use as edge cache servers. The maximum number of pages that can be cached is 1,000,000 pages. These pages are generated by queries that can be categorized into 1,000 query types. Thus, on average, there are 1,000 pages (query instances) per query type. Among these 1,000 query types, 200 query types are non-cacheable (i.e. queries involved transactions, privacy, or security). All servers are located in dedicated machines. All of the machines are Pentium III 700Mhz one CPU PCs with 1GB of memory. They are running Redhat Linux 7.2.

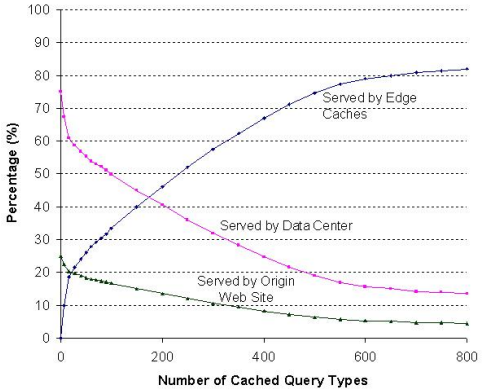


Figure 5: Correlation between Number of Cached Query Types and Load Distribution

7.2 Correlation between Number of Cached Query Types and Edge Cache Hit Rates

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 rate is an important factor since a high invalidation rate will lead to a potentially high cache miss rate in the future. Another consideration is that there is overhead for invalidation checking process; a well-tuned cache management should cache only a small portion of pages to serve most of the requests.

We have developed a cache replacement algorithm that takes into consideration (1) user access patterns, (2) page invalidation pattern, (3) temporal locality of the requests, and (4) response time. Consequently, we are able to select only a small number of query types (i.e., pages generated by these query types) to cache, but maintain a high hit rate at edge caches.

Figure 5 shows the correlation between the number of *selected* cached query types and the request distribution percentages in the edge caches, data centers, and origin Web site. As we can see in the figure, when we choose to cache 200 query types (25% of all query types), the cache hit rate is close to 48%. However, when we increase cached query types from 500 to 800, the cache hit rate can only be improved by an additional 5%. Note that user requests are distributed among edge caches, data centers, and the origin Web site. Thus, when there is a high edge cache hit rate, the load at both the data center and the master database is reduced. The figure shows that approximately 75% of the user requests due to edge cache miss can be handled by the data center and the master database handles the rest 25% of the requests, including requests due to edge cache misses and data center misses, as well as all transactions. The ratio of 75%:25% is independent from the edge cache hit rate.

7.3 Effects of Request Rates and Cache Hit Rates on Request Response Time

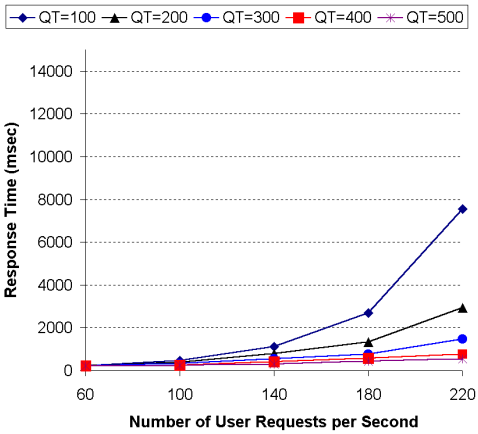


Figure 6: Effects of Request Rates and Number of Cached Query Types on Request Response Time at the Data Center

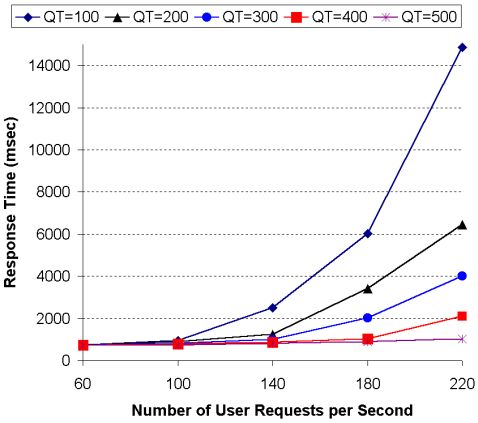


Figure 7: Effects of Request Rates and Number of Cached Query Types on Response Time at the Origin Web Site

The next experiment we conducted is to measure the effects of cache hit rates (i.e., number of cached query types) and request rates on the response time. Note that we do not present the effects of cache hit rates and request rates on the response time at the edge cache server since the response time at edge cache server fast.

In Figures 6 and 7, we plot the response time at the data center and the origin Web site with respect to various request rates (i.e., from 20 to 120 requests per second) and numbers of cached query types (i.e., from 100 to 500). The experimental results indicate the following:

- the origin Web site is much more sensitive to the request rate than the data center caused by the network latency between the users and the origin Web site. The master database needs to keep more connections open for a longer period and many requests are queued waiting for processing.
- the response time at the origin Web site is higher than that at the data center. This is also due to the network latency;
- when the request rate reaches a certain threshold, the response time increases sharply. We observe this effect in both figures. This is because the requests start to accumulate after the request rate reaches a certain level; and
- the response time is reduced when the number of cached query types is increased. We observe such effects in both figures (at the data center and the origin Web site).

7.4 Effects of Network Latency on Request Response Time at the Origin Web Site

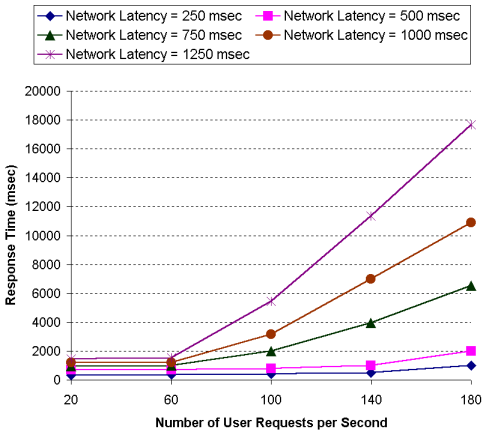


Figure 8: Effects of Request Rates and Network Latency on Request Response Time at the Origin Web Site

In earlier experiments, the network round trip latency between users and the origin Web site was set around 250 ms. In this experiment, to measure the effects of network latency, we fixed the edge cache hit rate (i.e. the number of cached query type) but varied the request rate and the network latency between users and the origin Web site. The round trip time is altered as 250 ms, 500 ms, 750 ms, 1000 ms, and 1250 ms.

In Figure 8, we plot the response time at the origin Web site with respect to a set of different user request rates and network latency settings. The experimental results in the figure indicate that request response time at the origin Web site is very sensitive to the request rate and network latency when the user request rates reach a certain level. When the request rate and network latency reach a certain threshold, the response time increases sharply. Again, this is because request accumulation frequently occurs while the system experiences heavy request load and long network latency. We also observe that the response time increases at a faster rate with longer network latency; as we can see from the figure, the slope of the plot for the network latency of 1250 ms is much steeper than that for the network latency of 250 ms.

8. FRESHNESS-DRIVEN CACHING

In this section, we describe the proposed freshness-driven adaptive dynamic content caching followed by the discussion and analysis of the experimental results.

8.1 Adaptive Caching Policy

To achieve the best assured dynamic content freshness, we need to tune the caching policy so that the response time is close to the invalidation cycle in an equilibrium point. However, the response time and invalidation cycle can only be improved at the expense of each other. In Section 7, we found that response time can be impacted by (1) network latency, (2) user request rates, (3) database update rates, (4) invalidation cycle (frequency), and (5) edge cache hit rates. Network latency, request rates, and database update rates

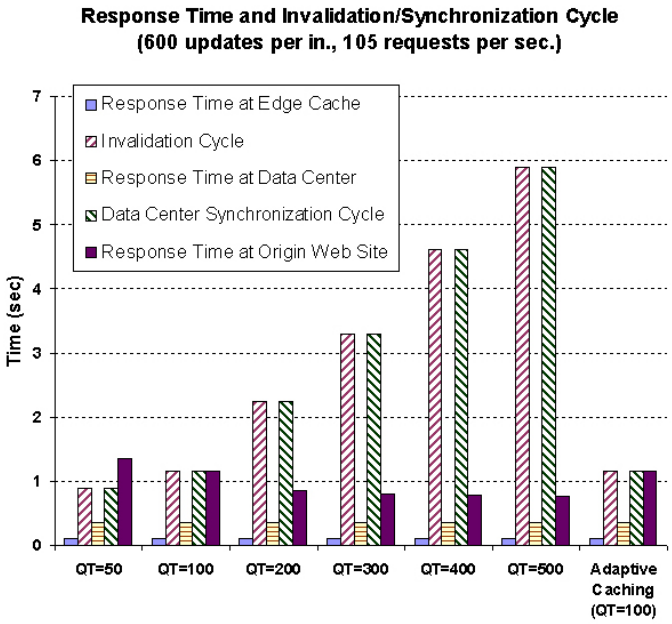


Figure 9: Effects of the Number of Cached Query Types on Response Time and Invalidation/Synchronization Cycle

depend on the network infrastructure and application characteristics and hence can not be controlled. However, we can tune the edge cache hit rate (to a certain degree) by adjusting the number of cached query types; and consequently affect response time.

Figure 9 summarizes the effect of the number of query types on response time, invalidation cycle, and synchronization cycle. In this configuration, the database update rate is 120 updates per second and user request rate is 105 requests per second. We vary the number of cached data types between 50, 100, 200, 300, 400, and 500. We plot the response time at the edge cache, data center, and origin Web site. We also plot the length of the invalidation and data center synchronization cycle (note that the invalidation cycle and data center synchronization cycle are identical since invalidation and synchronization are performed together).

When the number of cached query types decreases from 500 to 100, the invalidation/synchronization cycle and response time at the origin Web site move to an equilibrium point. At the equilibrium point (i.e., $QT = 100$), the assured freshness is higher than the response times at the edge cache and the data center. Since the response times at the edge cache and data center are much lower than the response at the origin Web site, 1.2 seconds is the best freshness we can assure. The value will increase when the network latency between users and the origin Web site or user request rate increase and vice versa.

We derive the following adaptive caching policy for maintaining request response time and invalidation cycle close to an equilibrium point:

- If the response time at the origin Web site is larger than the length of the invalidation cycle, the response time can be reduced by increasing the number of cached query types until the request response time and invalidation cycle reach an equilibrium point. Note that when we increase the number of cached query types, the edge cache hit rate will increase. As a result, the request rates at the data center and the origin Web site are reduced. Consequently, the response time at both the data center and the origin Web site is improved.
- Similarly, if the invalidation cycle is longer than the request response time, we can lower the invalidation cycle by decreasing the number of cached query types until the request response time and invalidation cycle reach an equilibrium point.

In the current implementation, the adaptive caching policy is deployed at the edge server. The response time is measured at the cache server assuming that the round trip between users and the cache server (i.e., functioning as a user side proxy) is negligible.

8.2 Experiments on Assuring Freshness

We conducted a series of experiments to evaluate the proposed freshness-driven adaptive dynamic content caching technique. In these experiments, we created setups by varying request rates and database update rates (every 10 minutes) for the first 60 minutes of experiment. In the first 60 minutes, the network latency is stable at 500 ms delay on average. After 60 minutes, the database update rate and request rate are fixed while the network latency is varied. We observed the effects of our freshness-driven adaptive caching technique on maintaining the best freshness that can be assured for a given system configuration and setup. In Figure 10, we plot the response time at the origin Web site as a dotted line and invalidation cycle as a solid line. The numbers next to the dotted line are the number of cached query types. In this figure, we do not plot the response time at the edge cache and the data center since the response times are very low. The response times measured are the response time at the origin Web site.

A sudden change in the request rate and network latency will cause temporary imbalance of response time (at the origin Web site) and invalidation/synchronization cycle (at the edge caches and the data center). As the imbalance is detected, the freshness-driven adaptive caching technique makes necessary adjustments to the number of cached query types, and this impacts the cache hit rate and the response time. As time elapses, the response time and invalidation cycle shifts to a new equilibrium point which supports the best content freshness that can be assured. For example, when the database update rate suddenly changes from 40 updates per second to 120 updates per second at 20^{th} minute of the experiment, the number cached query types is decreased to 100, where a new equilibrium point is reached. This compensates the sharp increase of invalidation/synchronization cycle due to the additional 80 updates per seconds; the number cached query types is decreased to 100.

The request response time at the origin site can be adjusted fairly quickly since it is very sensitive to the cache hit rate. And, we also observe in all our experiments that when we change the number of cached query types to compensate temporary imbalance of response time and invalidation cycle, both response time at the origin Web site and invalidation cycle moves toward a new equilibrium point at the same time. The observations in the experiments are consistent with our experimental results in Section 7.

8.3 Experiments on Adaptability

Next, we compare the content freshness that can be assured by four system configurations as follows:

1. a data center that does not deploy any dynamic content caching solution;
2. a data center with dynamic content caching but the numbers of cached query types are preset as 200;
3. a data center with dynamic content caching but the numbers of cached query types are preset as 400; and

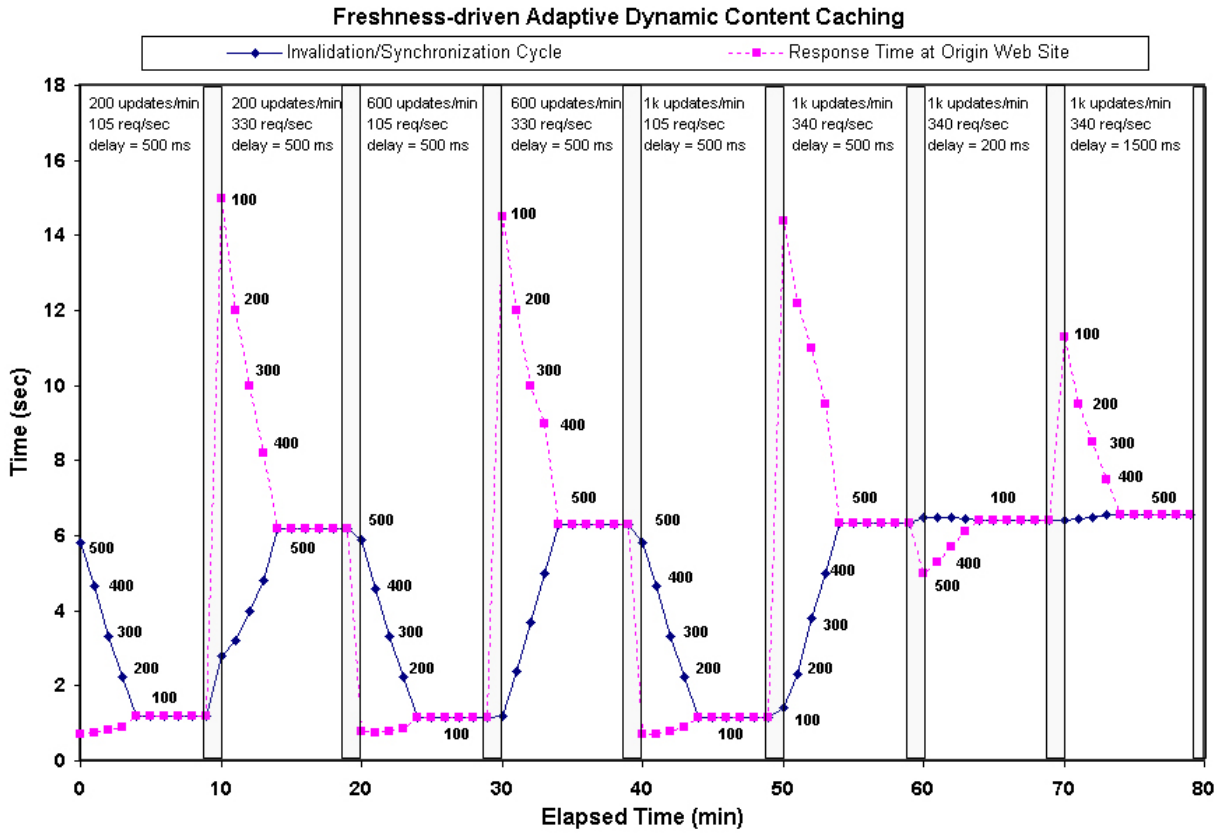


Figure 10: Impact of Adaptive Caching on Content Freshness

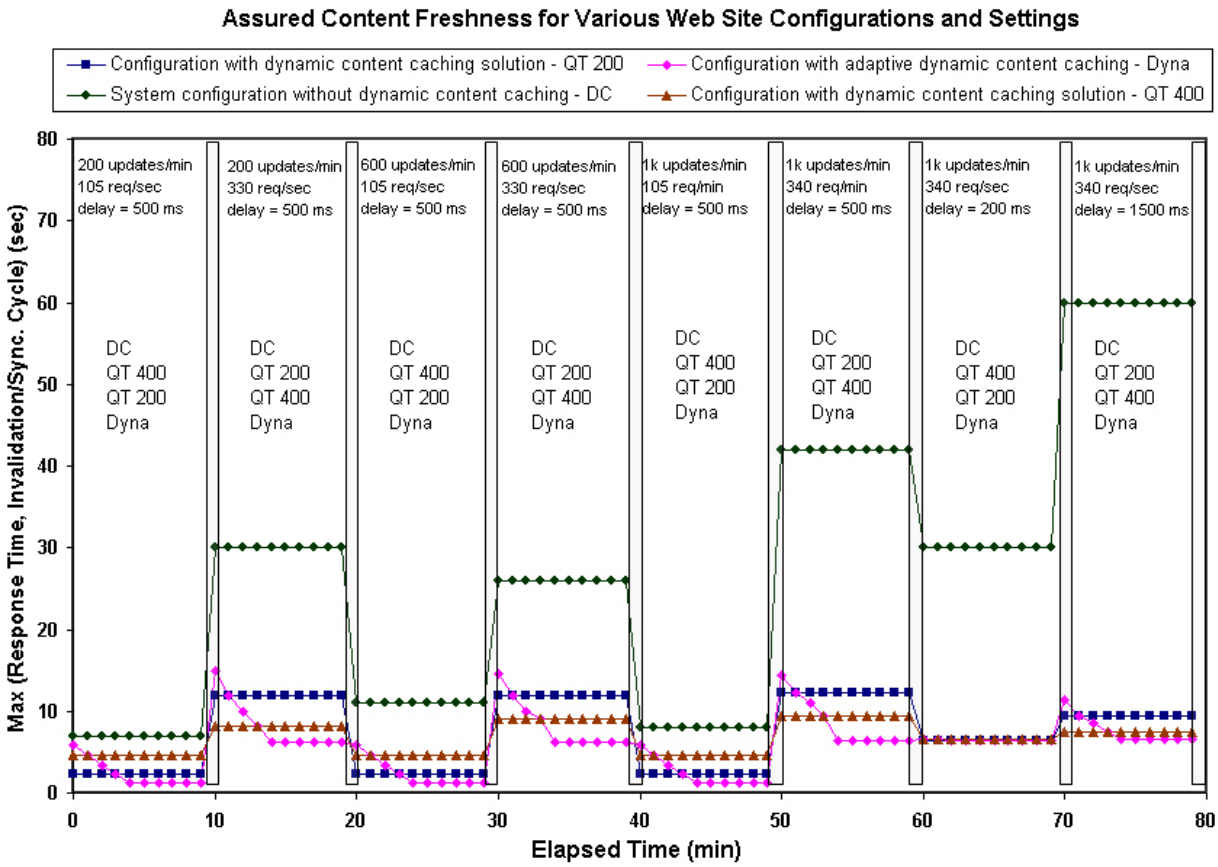


Figure 11: Comparisons of Assured Content Freshness for Three System Configurations

4. a data center that deploys dynamic content caching and the proposed freshness-driven adaptive caching technique.

In Figure 11 we plot the larger of the response time and the invalidation/synchronization cycle length at data centers of the four system configurations. This value gives the content freshness that a given system configuration can support for given request and database update rates. In the middle of each period in the figure, we indicate the ranking of four configurations in term of freshness they assure (the lower the better).

The figure shows the huge benefit provided by deploying Cacheportal. The three configurations with CachePortal (i.e., $QT200$, $QT400$, and $Dyna$) consistently provides much fresher content than the typical data center configuration. Especially, during the heavy traffic conditions in the periods of 10-20, 30-40, and 50-60 minutes and the long network latency delay condition in the period of 70-80 minutes, the system configuration with the freshness-driven adaptive caching supports content freshness up to 15 times better than those of the typical data center.

As we can see, some time (i.e., the periods of 10-20, 30-40, 50-60, and 70-80 minutes) the configuration 2 ($QT200$) performs better than the configuration 4 ($QT400$) and opposite is true in other times (i.e., the periods of 0-10, 20-30, and 40-50 minutes). However, the adaptive caching can consistently tune the caching policy to provide the best assured freshness feasible for a specific set of conditions and system configuration. In our experiments, the adaptive caching supports content freshness up to 10 times better than even those systems that already deploy dynamic content caching solutions. The experiments strongly show the effectiveness and benefits of the proposed freshness-driven adaptive caching technique in providing fresh dynamic content.

8.4 Experiments on Accelerating Response Time

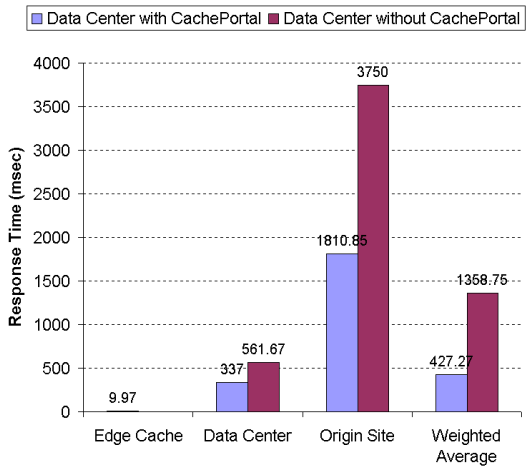


Figure 12: Response Time of Data Center-hosted Web Applications with and without CachePortal

The next experiment we conducted is to measure the performance gain (in terms of the response time observed by the users) achieved through our proposed approach. In this experiment, the baseline setting is used (i.e., database update rate is 600 tuples per table per minute; the network latency is 250 ms round trip; the number of cached query types is 300; and the request rate is 100 requests per second). We set up two system configurations: (1) typical data center architecture; and (2) data center with deployment of the CachePortal and freshness-driven adaptive dynamic content caching. For the first system configuration, the request rates distributed to the data center and the master database (i.e., origin Web site) are 75 requests per second and 25 requests per second, respectively. For the second system configuration, the request rates distributed to the edge caches, the data center ,and the master database (i.e., origin Web site) are 40, 45, and 15 requests per second, respectively. The characteristics of request distribution are described in Section 7.2 and Figure 5.

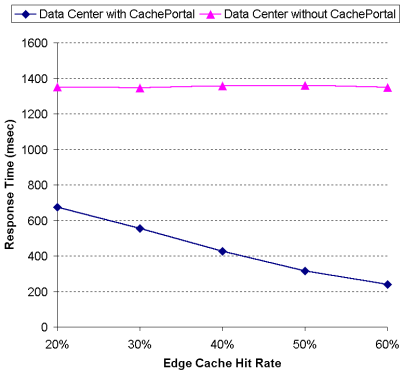


Figure 13: Effects of Edge Cache Hit Rates (i.e., Number of Cached Query Types) on Weighted Average Response Time

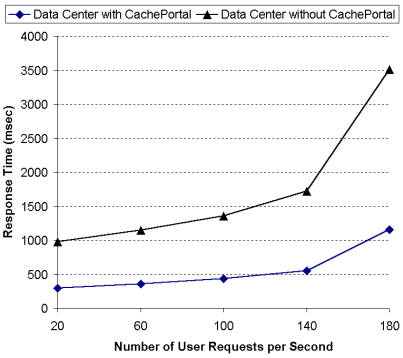


Figure 14: Effects of User Request Rate on Weighted Average Response Time

Figure 12 shows the response times measured at the edge caches, data centers, and origin Web sites for two system configurations. These figures also show the weighted average response time. As we can see, with deployment of CachePortal, we can reduce the average response time by almost 70%. When the system caches 300 query types, the edge cache hit rate reaches 40%. In Figure 13, we show the average response time of the second system configuration with various edge cache hit rates (from 20% to 60%). We observe that when the edge cache hit rate increases from 20% to 60%, the average response time can be reduced by 70%. This is because delivering dynamic content from the edge caches is much faster than garnering content on demand. Thus, when the edge cache hit rate increases, the weighted average response time can reduce significantly.

The next experiment we conducted is to measure the effects of the request rate on response time for these two system configurations. In Figure 14, the request rate is increased from 20 requests per second to 180 requests per second while the other parameters remain the same. As we can see, when the request load to the system increases 8 times, the average response time of the first system configuration increases 3.5 times, which indicates that the data center architecture is reasonably scalable. On the other hand, we see

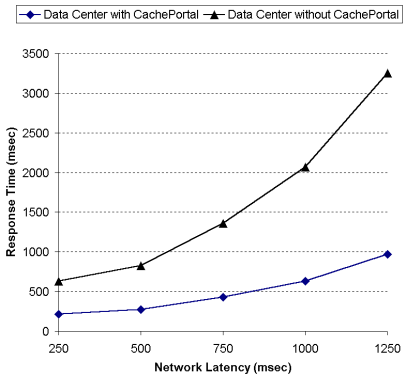


Figure 15: Effects of Network Latency on Weighted Average Response Time

that the average response time by the data center architecture with deployment of Cacheportal (i.e., the second system configuration) increases at a slower rate.

We also measure the effects of network latency on the average response time. We increase the network latency between the data center and the origin Web site (the master database) from 250 ms to 1250 ms. We observe the the second system configuration is less sensitive to the network latency as the average response time increases at a slow rate as shown in Figure 15. This is because the data center architecture with deployment of Cacheportal serves a higher percentage of requests from the edge than the typical data center architecture.

8.5 Experiments on Scalability

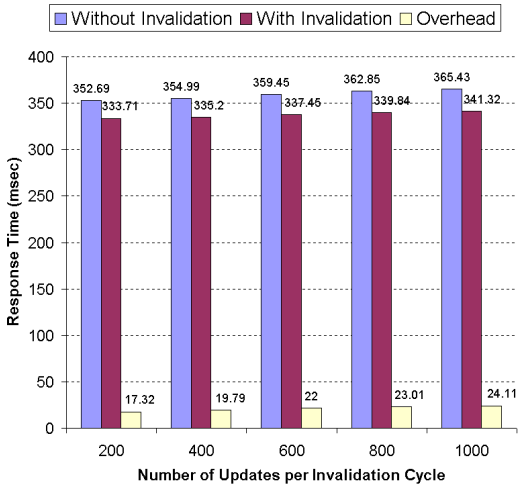


Figure 16: Effects of Update Rate on Invalidation Overhead

In Section 6, we have analytically concluded that the invalidation cycle is mainly impacted by the number of cached query types, and that database update rates have relatively lower impact on the length of the invalidation cycle. We have conducted experiments to validate our analysis.

Figures 16 and 17 show the results for the evaluation of the scalability of our proposed solutions. We first look at the effects of the update rate on invalidation overhead. As shown in Figure 16, the update rate has an impact on the computing time of polling queries in the invalidation as well as synchronization time. Since the polling queries are submitted to the same machine and since appropriate indexing schemes are used, the impact of the increase

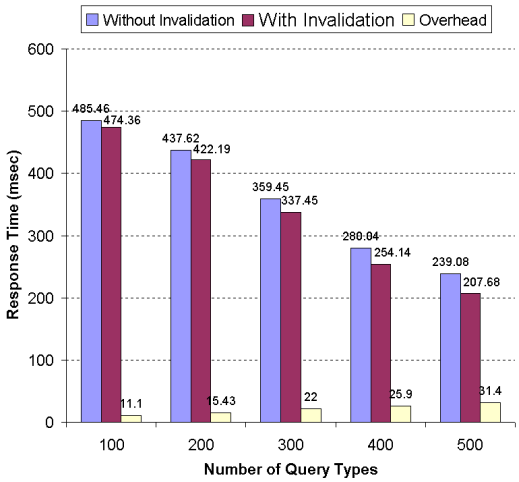


Figure 17: Effects of Cached Query Types on Invalidation Overhead

in the update rate is limited.

We also evaluate the effects of the number of cached query types on the invalidation overhead. The number of cached query types determines the number of polling queries that 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 the increase in cached query types is limited, although the effect of the number of cached query types is more notable than the effect of database update rate. This is shown in Figure 17. We can see that as we increase the number of cached query types, the reduction of response time is much higher than the increase of invalidation overhead.

9. RELATED WORK

Applying caching solutions for Web applications and content distribution has received a lot of attention in the Web and database communities[7, 8, 9, 10, 11, 12, 3]. These provide various solutions to accelerate content delivery as well as techniques to assure the freshness of the cached pages. Note that since Web content is delivered through the Internet, the content freshness can only be *assured* rather than being *guaranteed*.

WebCQ [13] is one of the earliest prototype systems for detecting and delivering information changes on the Web. However, the change detection is limited to ordinary Web pages. Yagoub et al. [14] have proposed caching strategies for data intensive Web sites. Their approach uses materialization to eliminate dynamic generation of pages but does not address the issue of view invalidation when the underlying data is updated. Labrindis and Roussopoulos [15] present an innovative approach to enable dynamic content caching by maintaining *static* mappings between database contents and Web pages, and therefore requires a modification to underlying Web applications.

Dynamai [4] from Persistence Software is one of the first dynamic caching solution that is available as a product. However, Dynamai relies on proprietary software for both database and application server components. Thus it cannot be easily incorporated into existing e-commerce framework. Levy et al. [5] at IBM Research have developed a scalable and highly available system for serving dynamic data over the Web. The IBM system was used at Olympics 2000 to post sport event results on the Web in timely manner. This system utilizes database triggers to generate update

events as well as intimately relying on the semantics of the application to map database update events to appropriate Web pages.

Heddaya and Mirdad [16], where authors propose a diffusion-based caching protocol that achieves load-balancing, [17] which uses meta-information in the cache-hierarchy to improve the hit ratio of the caches, [18] which evaluates the performance of traditional cache hierarchies and provides design principles for scalable cache systems, and [19] which highlights the fact that static client-to-server assignment may not perform well compared to dynamic server assignment or selection.

SPREAD [20], 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 [20] 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.

10. CONCLUDING REMARKS

In this paper, we propose a *freshness-driven adaptive dynamic content caching* technique that maintains the best content freshness that an application-hosting data center configuration can support. The technique monitors the response time and the length of the invalidation cycle and dynamically adjusts the caching policy accordingly. By balancing invalidation cycle length and response time our technique is able to maintain the best content freshness that can be assured. The experiments validate the effectiveness of our technique. Under heavy traffic, the freshness-driven adaptive caching supports content freshness up to 20 times better than those data center-hosted applications without dynamic content caching. It also supports content freshness up to 7 times better than those data center-hosted applications that deploy dynamic content caching solutions. Furthermore, our approach also provides faster response times and better scalability.

11. REFERENCES

[1] Akamai Technology. `http://www.akamai.com/`.

[2] B. Krishnamurthy and C.E. Wills. Analyzing factors that influence end-to-end web performance. In *Proceedings of the 9th World-Wide Web Conference*, pages 17–32, Amsterdam, The Netherlands, May 2000.

[3] 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*, Hong Kong, China, August 2002.

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

[5] 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.

[6] Wen-Syan Li, Wang-Pin Hsiung, Dmitri V. Kalashnikov, Radu Sion, Oliver Po, Divyakant Agrawal, and K. Selçuk Candan. Issues and Evaluations of Caching Solutions for Web Application Acceleration. In *Proceedings of the 28th Very Large Data Bases Conference*, Hong Kong, China, August 2002.

[7] 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.

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

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

[10] Anoop Ninan, Purushottam Kulkarni, Prashant Shenoy, Krithi Ramamritham, and Renu Tewari. Cooperative Leases: Scalable Consistency Maintenance in Content Distribution Networks. In *Proceedings of the 2002 World-Wide Web Conference*, Honolulu, Hawaii, USA, May 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] 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.

[13] Ling Liu, Calton Pu, and Wei Tang. WebCQ: Detecting and Delivering Information Changes on the Web. In *Proceesings of International Conference on Information and Knowledge Management*, Washington, D.C., November 2000.

[14] Khaled Yagoub, Daniela Florescu, Valrie Issarny, and Patrick Valduriez. Caching Strategies for Data-Intensive Web Sites. In *Proceedings of the 26th VLDB Conference*, Cairo, Egypt, 2000.

[15] A. Labrindis and N. Roussopoulos. Self-Maintaining Web Pages - An Overview. In *Proceedings of the 12th Australasian Database Conference (ADC)*, Queensland, Australia, January/February 2001.

[16] 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.

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

[18] 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.

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

[20] 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, May 2000.