A Collaborative Approach for Caching Dynamic Data in Portal Applications

Mehregan Mahdavi **John Shepherd** **Boualem Benatallah**

School of Computer Science and Engineering
University of New South Wales,
Sydney 2052, Australia,
Email: {mehrm,jas,boualem}@cse.unsw.edu.au

Abstract

Portals are one of the rapidly growing applications on the Web, providing a single interface to access different sources (providers). Providing fast response time is one of the critical issues in such applications. Dissatisfaction of users dramatically increases with increasing response time, resulting in abandonment of Web sites, which in turn could result in loss of revenue by businesses. In this paper we address the performance of such applications through caching techniques. We discuss the limitations of existing solutions and introduce a caching strategy based on collaboration between the portal and its providers. Providers trace their logs, extract information to identify good candidates for caching and notify the portal. Caching at the portal is mainly decided based on scores calculated by providers and associated with objects. We evaluate the performance of the collaborative caching strategy using simulation data. We also address the issue of heterogeneous scoring policies by different providers and introduce mechanisms to regulate caching scores.

Keywords: Web Caching, Caching Dynamic Data, Web Portal, Web Services, Mediator Systems

1 Introduction

Web Portals are emerging Internet-based applications enabling access to different sources (providers) through a single interface. Using Web portals can help users to effectively find the desired information, service, or product between a (large) number of providers without navigating through them individually. In other words, the portal is a *mediator* representing an *integrated service* which is the aggregation of all services provided by the individual providers. Business portals, such as *Amazon* (www.amazon.com) and *Expedia* (www.expedia.com), are examples of such applications where customers can search for services or products to use or buy on-line.

Providing fast response time is one of the critical issues in portal-enabled applications. Network traffic between the portal and individual providers, server workload, or failure at provider sites are some contributing factors for slow response time. Previous research (Wong 1999, Zona Research Inc. 2001) has shown that dissatisfaction of clients dramatically increases with increasing response time, resulting in abandonment of Web sites. This, in turn results in loss of revenue by businesses.

Caching is one of the key techniques which promises to overcome some of the portal performance issues. In particular, caching response messages (which we also refer to as *dynamic objects* or *objects*[1], for short) gives portals the ability to respond to some customer requests locally. As a result, response time to the client is improved, client satisfaction is increased and better revenue for the portal and the providers is generated. In addition, network traffic and the workload on the providers' servers are considerably reduced. This in turn improves scalability and reduces hardware costs.

Inherent in the notion of caching are the ideas that we should maintain as many objects as possible in the cache, but that the cache is not large enough to hold all of the objects that we would like. This introduces the notion that some objects are better candidates for caching than others. The best candidates for caching are objects which are requested frequently and not changed very often. For rapidly changing or infrequently accessed objects, it might not be beneficial to cache them at all.

A *caching policy* is required to determine which objects should be cached. Products such as *Oracle Web Cache* (Oracle Corporation 2001*b*), *IBM WebSphere Edge Server* (http://www.ibm.com) , and *Dynamai* (http://www.persistence.com/products/dynamai/) from *Persistence Software* enable system administrators to specify caching policies. This is done mainly by including or excluding objects or object groups (e.g., objects with a common prefix in the URI) to be cached, determining expiry date for caching objects or object groups, and etc. Server logs (i.e., access log, and database update log) are also used to identify objects to be cached.

Caching dynamic objects at portals introduces new problems to which existing techniques cannot be easily adapted. Since the portal may be dealing with a large number of providers, determining "cache-worthy" objects by an administrator or by processing logs is impractical. On one hand, an administrator cannot identify candidate objects in a highly dynamic environment where providers may join and leave the portal frequently. On the other hand, keeping and processing access logs in the portal is impractical due to high storage space and processing time requirements. Moreover, the providers' database update logs, which are critical in determining which objects have been modified, are not normally accessible to the portal.

In this paper we examine caching solutions to increase the performance in Web portals. We introduce a caching strategy based on collaboration between the portal and its providers to overcome the above-mentioned problems. Providers trace their logs, extract information to identify good candidates for caching and notify the portal. Providers associate a score with each response message which represents

[1] Dynamic objects are data items requested by the portal, such as XML or SOAP response messages.

the usefulness of caching this object.

Clearly, such a scheme imposes overheads on the providers, and leaves open the possibility that they may try to *subvert* the caching system by claiming that all of their pages are essential to remain in the cache. We show that the overheads are minimal and introduce mechanisms to regulate caching scores, to overcome these problems.

The remainder of this paper is organised as follows. Section 2 provides an overview of portals, their architecture, implementation and performance issues. In Section 3 we introduce a collaborative caching strategy based on a score associated with objects by providers. Experimental results to evaluate the effectiveness of this strategy are presented in Section 4. More details about existing caching solutions and related works are described in Section 5. Finally, some conclusions are presented in Section 6.

2 Web Portals: Architectures and Performance Issues

Web portals enable access to different providers through a single interface. Providers register their services in the portal to commence a relationship between the portal and themselves. Providers fall into two broad categories, based on their relationship with each other:

- **Complementary:** Complementary providers are those who provide different elements of a composite service or product. For example, in a travel planner portal, services such as flight, accommodation, and car rental are complementary services. Another example is a computer manufacturing portal where each provider may provide different parts of a computer.
- **Competitor:** Competitor providers are those who provide the same service or product. They compete with each other either in providing better quality of Service (QoS), e.g., faster response time or a cheaper price. Fast response time through the portal is a QoS which both portal and provider try to provide. A computer selling portal is an example where providers compete with each other in selling their products, such as PCs or printers.

The relationship between the portal and the providers can be realized in two ways:

- **Centralized:** Each provider sends its content to the portal and the contents from different providers are combined and maintained by the portal. When the portal receives a query, e.g., a browse request, it processes the query locally. Each provider is responsible for providing updated content to the portal.
- **Distributed:** Each provider maintains its content. When the portal receives a query, it forwards the query to the appropriate provider(s). Each provider processes the query and returns its result to the portal. The provider may need to contact other providers, e.g., a service or product is made of different services or items which are provided by other providers. The final result will be integrated at the requesting portal and returned to the user.

The centralized approach has the advantage that all queries are carried out on the portal under a single integrated schema. This has some administrative and performance benefits (especially in the speed of answering queries), and works effectively in domains

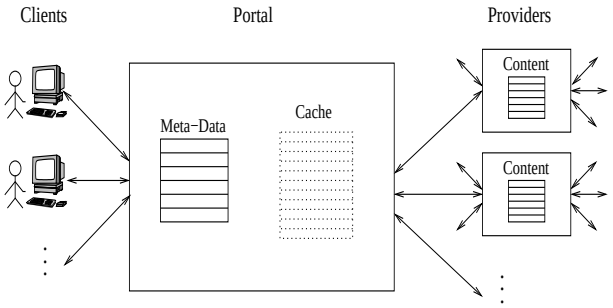


Figure 1: A Distributed Portal

where data is not highly dynamic (e.g. the catalogue of an on-line bookshop). However, it has a number of serious disadvantages. In particular, it is not suitable for application domains, such as ticket booking systems or purchasing systems, where "freshness" of data is critical; since providers are responsible for sending content updates, they will typically do this only periodically. Another potential problem arises if providers have relationships with a number of portals; it becomes costly for the provider to keep the content up-to-date at all portals. We focus on *distributed portals* for the remainder of this paper.

In a distributed portal, which may include complementary providers, competitor providers or both, the relationship between portal and providers is not necessarily fixed and pre-established. It can be rather highly dynamic where new providers can join or existing providers can leave the portal. Figure 1 shows the general architecture of a distributed portal along with the providers. Meta-data is information about providers, provided when they register their service. As can be seen in the figure, each provider may have a relationship with a number of portals. Moreover, each provider may have a number of sub-providers.

The distributed approach is more appropriate for applications which need fresher data, since providers are queried directly and always provide answers based on up-to-date content. However, this clearly introduces significant overheads into the query processing (message handling, extra network traffic, possibility of failure of provider sites), which may result in slow response to the user.

A potential solution to the response time problem for distributed portals is to cache response messages (e.g., SOAP or XML) from providers at the portal. Caching data from providers can reduce network traffic. It also reduces the workload on the provider Web application server and database server by answering some requests locally in the portal site. Less workload on the provider site leaves more processing power to process incoming queries which results in more scalability and reduces hardware costs.

In environments where the providers are mainly complementary, the performance of the portal is limited by the performance of the provider with the worst performance among the providers taking part in a composite service. Providing data from a shorter distance (i.e. locally on the portal) improves the response time to the user. This can in turn help in better user satisfaction and finally in better revenue for the portal and the provider. For example, if a fast browse session for composite services or products is provided, users might be more willing to continue shopping, which may finally lead to using the service or buying the product. Moreover, users will be more likely to come back later.

In competitor environments caching is of more interest for providers. If a portal lists the contents from different providers as they arrive, providers with bet-

ter response time have a better chance to be listed for the user. Assuming that in most cases users are only interested in the "Top N" results, failure to provide a fast response may result in a provider losing the chance to be listed in "Top N" results. This may result in less revenue for the business.

The use of a cache of provider objects in the portal immediately raises the issues of how many objects can/should be cached, which objects are the best candidates for replacement when the cache fills, and how is information provided to the portal in order to determine this. These issues are addressed by our collaborative caching strategy.

3 A Collaborative Caching Strategy

Caching a particular object at the portal depends on the available storage space, response time (QoS) requirements, access and update frequency of objects. The best candidates for caching are those who are: (i) accessed frequently, (ii) not changed very often, and (iii) expensive to compute or deliver (Kossmann & Franklin 2000, Florescu, Yagoub, Valduriez & Issarny 2000).

As mentioned earlier, due to the potentially large number of providers and dynamicity of the environment, it is not feasible to identify "cache-worthy" objects on the portal, either by a system administrator or by mining server logs: a human administrator cannot handle frequent changes to the collection of providers; maintaining and processing access logs in the portal imposes too much storage and processing overhead; database update logs from the providers are typically not accessible to the portal.

In order to provide effective caching in a distributed, dynamic portal environment, we propose a strategy based on the collaboration between the providers and the portal. A caching score (called *cache-worthiness*) is associated to each object, determined by the provider of that object. The cache-worthiness of an object, a value in the range [0,1], represents the usefulness of caching this object at the portal. A value of zero indicates that the object cannot be cached in the portal, while a value of 1 indicates that it is desirable to cache the object in the portal (Mahdavi, Benatallah & Rabhi 2003). The cache-worthiness score is sent by the provider to the portal in response to a request from the portal. The decision whether to cache an object or not is made by the portal, based on the cache-worthiness scores along with other parameters such as recency of objects, utility of providers, and correlation between objects.

The caching strategy is supported by two major tables, the *cache look-up table* used by the portal to keep track of the cached objects, and the *cache validation table* used by the providers to validate the objects cached at the portal(s). When a hit is detected at the portal for a provider's response message, a validation request message is sent to the relevant provider. The provider checks the freshness of the object by probing the cache validation table to find the relevant entry. If the cache validation table does not contain an entry for the object, it means that the corresponding object is not fresh anymore due to changes in the database. It is also possible that entries are removed for other reasons such as space limitations. After the object is sent back, the portal responds to the user request and a copy of the object may be cached at the portal. Clearly, this approach uses a pull-based consistency mechanism. Other mechanisms such as those presented in (Ramamritham, Deolasee, Kathar, Panchbudhe & Shenoy 2000, Deolasee, Katkar, Panchbudhe, Ramamritham & Shenoy 2001, Duvuri, Shenoy & Tewari 2000, Olston & Widom 2002, Cao & Ozsu 2002, Liu & Cao 1998) can be considered. However, this is not a focus of this paper.

Changes in the back-end database invalidate entries in the cache validation table. If changing the content of the database affects the freshness of any object, then the appropriate entry in the provider cache validation table will be removed. Solutions for detecting changes in the back-end database and/or invalidating the relevant objects are provided in (Anton, Jacobs, Liu, Parker, Zeng & Zhong 2002, Selcuk, Li, Luoand, Hsiung & Agrawal 2001, Challenger, Iyengar & Dantzig 1999), and *Dynamai*. For this purpose, the technique presented in *Dynamai* is used, where the incoming (update) requests are used to invalidate cached objects.

Server logs in the provider sites are used to calculate a score for cache-worthiness, using the following parameters:

- The **access frequency** is calculated by processing Web application server access log.
- The **update frequency** is calculated by processing database update log.
- The **computation cost** is calculated by processing the database request/delivery log and calculating the time elapsed between the request and delivery of the result from the database.
- The **delivery cost** is measured by the size of the object. Larger objects are more expensive to deliver in terms of time and network bandwidth consumption.

The score for cache-worthiness is computed as the aggregation of the above parameters.

Although, all providers may use the same strategy to score their objects, the scores may not be consistent. This is mainly due to the fact that: (i) each provider uses a limited amount of logs to extract required information which varies from one to another, (ii) each provider may use different weight for each term, (iii) the computation cost may depend on the provider hardware and software platform, workload and etc., (iv) providers may use other mechanisms to score the objects, and (v) malicious providers may claim that all of their own objects should be cached, in order to "freeze out" other providers.

To achieve an effective caching strategy, the portal should detect such inconsistencies and regulate the scores given by different providers. For this purpose, the portal uses a regulating factor $\lambda(m)$ for each provider. When the portal receives a cache-worthiness score, it multiplies it by $\lambda(m)$ and uses the result in the calculation of the overall caching score. For each provider, $\lambda(m)$ can be set by the administrator at the beginning. It is adapted dynamically by tracing the performance of the cache for individual providers.

Adapting $\lambda(m)$ dynamically is done through the following process carried out by the portal:

- When the number of false hits (i.e., hits occurred at the portal while the object is already invalidated) for a provider exceeds a threshold, then the portal decreases $\lambda(m)$ for the provider.[2]
- When the number of real hits (i.e., hits occurring at the portal while the object is still fresh and can be served by the cache) for a provider exceeds a threshold, then the portal increases $\lambda(m)$.

[2]Note that this provides regulation of cache-worthiness scores to counter providers who seek to gain some advantage by claiming that all of their content should be cached in the portal.

Our experimental results show that the above collaborative caching strategy can improve performance in terms of throughput and network bandwidth usage. They also show that a cache replacement strategy based on eviction of objects with least cache-worthiness performs better than other strategies such as FIFO and LRU.

4 Experiments

In order to evaluate the performance of the collaborative caching strategy, we have built a simulation platform. This platform enables us to simulate the behaviour of a portal with different number of providers, different message sizes, etc. and allows us to evaluate performance measures such as *network bandwidth usage* and *throughput*. To evaluate the performance, we run two simulations for each collection of parameter settings, one with caching and one without caching, and compare the performance results. In the rest of this paper we call them `CacheCW` and `NoCache`, respectively. We also test the performance of the cache replacement strategy based on cache-worthiness (used in `CacheCW`). We compare its performance with two other cache replacement strategies, FIFO and LRU. In the rest of this paper `CacheFIFO` and `CacheLRU` stand for the collaborative caching strategy using FIFO and LRU for cache replacement.

The requests to objects from providers are generated based on random selection according to a *Zipf* distribution (Breslau, Cao, Fan, Phillips & Shenker: 1999). Scores for cache-worthiness are assigned to the objects in such a way that larger values are assigned to more popular objects and smaller values to less popular ones. The most important variable parameters used in the experiments are:[3]

- Average number of providers
- Average number of cacheable objects per provider (e.g., depending on the affordable processing and space overhead needed for calculating and storing cache-worthiness scores by providers)
- Average size of objects
- Cache size
- Network bandwidth
- The ratio of read requests to write requests

The experimental results shown in this paper are extracted using 20 providers with an average of 10000 cacheable objects each, network bandwidth equal to 10Mb/s, and the ratio of read requests to write requests equal to 90%. Average object size and cache size are varied between 8KB/16KB/32KB and 20MB/100MB/1GB in different experiments. In the rest of this section, we present experimental results which evaluate the performance of the caching strategy based on network bandwidth usage and throughput.

4.1 Network Bandwidth Usage

In the first experiment, we study the amount of network bandwidth usage per request. We run the simulation for 120 minutes. We vary the average size of objects between 8KB, 16KB, and 32KB among the simulation instances and generate the graph in Figure 2. A cache size equal to 1GB is assumed for all three simulation instances.

[3]The simulation platform allows us to set the value for these parameters.

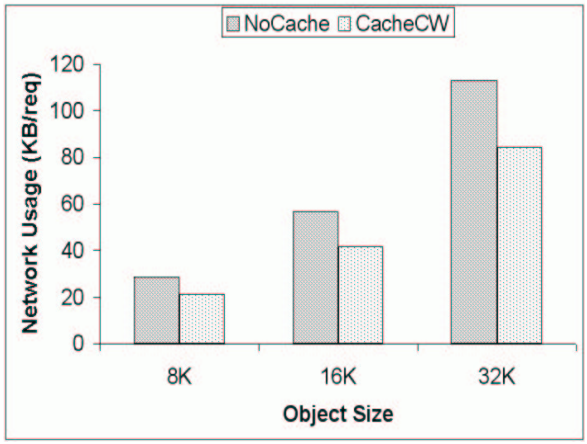


Figure 2: Network Bandwidth Usage per Request

As Shown in Figure 2, the caching strategy reduces the network bandwidth usage per request. According to the figure, `CacheCW` decreases the bandwidth usage per request, 1.32, 1.35, and 1.4 times for 8KB, 16KB, and 32KB sizes, respectively. It should be mentioned that each user request generates a number of sub-requests to providers. This is a random number and the maximum can be varied in the simulation. In other words, each request results in a number of response messages to be sent by providers. That is why the amount of network usage per request is a number of times more than the average size of response messages.

4.2 Throughput

In the second experiment, we study the throughput. Figure 3 shows the throughput measured as the number of performed requests per minute for different size of objects, i.e., 8K, 16K, and 32K. The results are based on 120 minutes simulation with a cache size equal to 1GB.

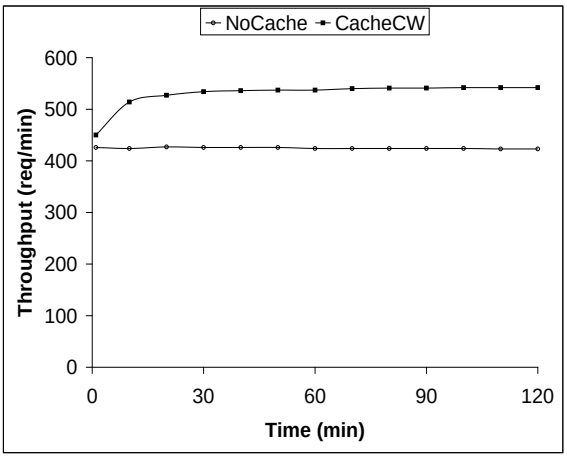
The throughput of `NoCache` stays almost constant at all the times. For `CacheCW`, it takes a while till the throughput reaches a stable value. This is because the cache is empty at the beginning and it takes some time until the cache is filled up. The caching strategy improve throughput 1.3, 1.32, and 1.35 times for 8KB, 16KB, and 32KB sizes, respectively.

4.3 Cache Replacement Strategy

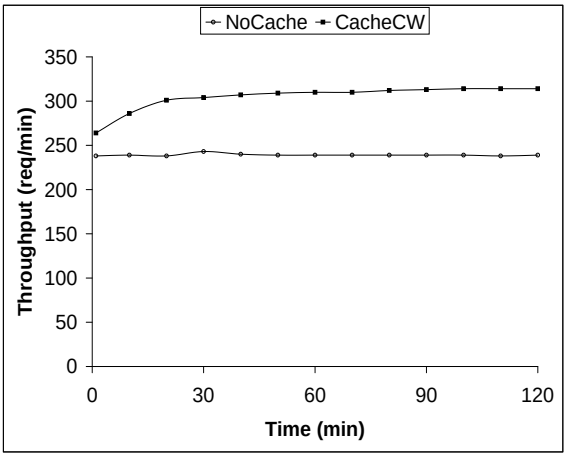
In the third experiment, we compare the cache replacement strategy based on cache worthiness with two other strategies, FIFO and LRU. In this experiment an average size of objects equal to 16KB is assumed. We vary the cache size between 1GB, 100MB, and 20MB and compare hit ratios (i.e., number of hits divided by number of accesses) of three strategies. Figure 4 shows that where there is not much limitation on the cache size (e.g., 1GB), all three strategies perform similarly. When there is cache size limitation (e.g., 100MB, or 20MB) `CacheCW` performs better followed by `CacheLRU` and then `CacheFIFO`. Figure 4 (c) shows that with more limitation on the cache size the difference becomes significant. For the cache size equal to 20MB, the hit ratios are 0.23 for `CacheCW`, followed by 0.17 for `CacheLRU`, and 0.14 for `CacheFIFO`.

4.4 Cache Size

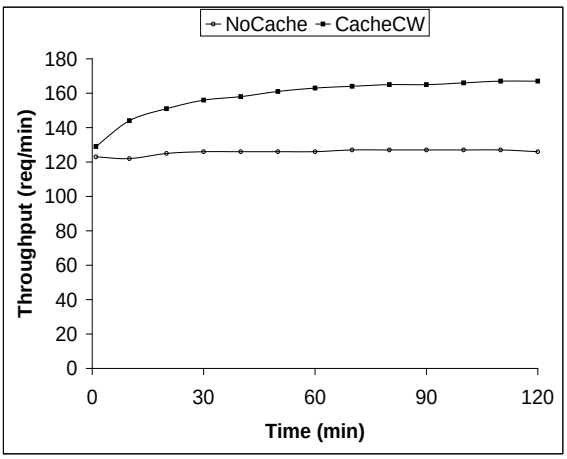
In the last experiment, we study the effect of cache size on the overall performance (i.e., network band-



(a) Average Size: 8KB

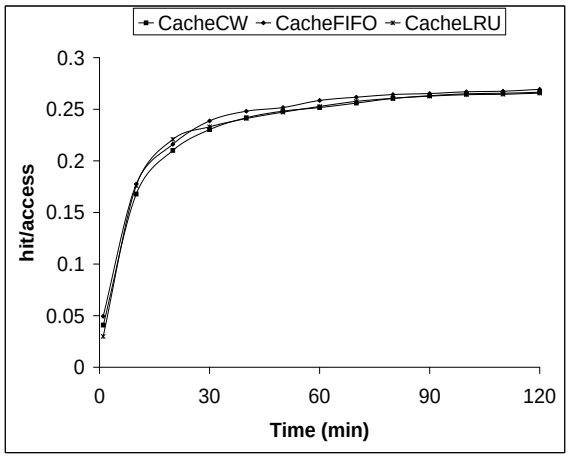


(b) Average Size: 16KB

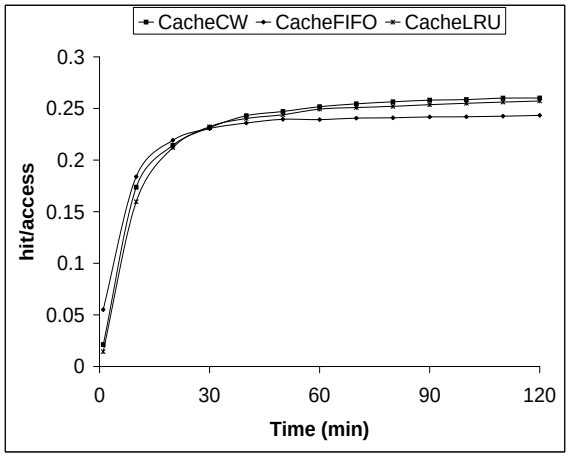


(c) Average Size: 32 K

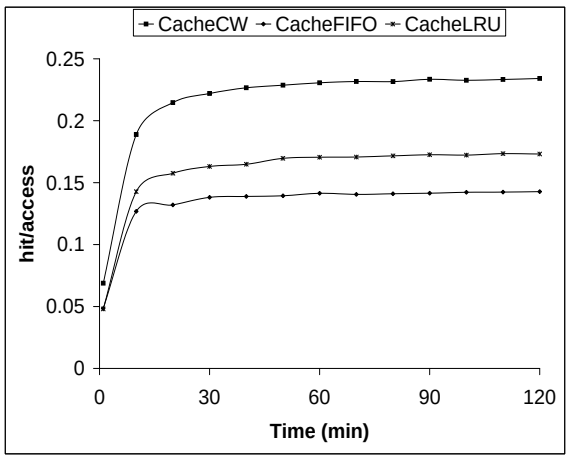
Figure 3: Throughput



(a) Cache Size: 1GB



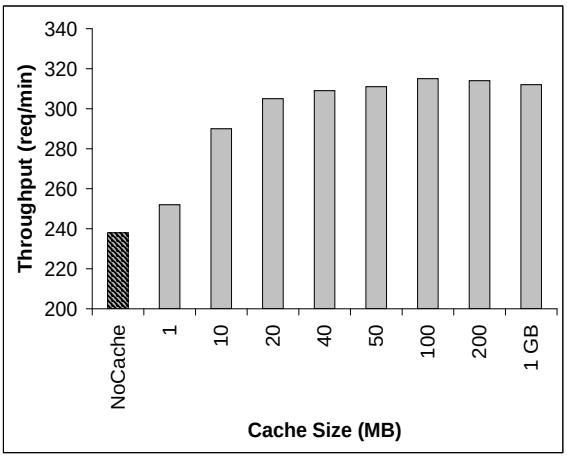
(b) Cache Size: 100MB



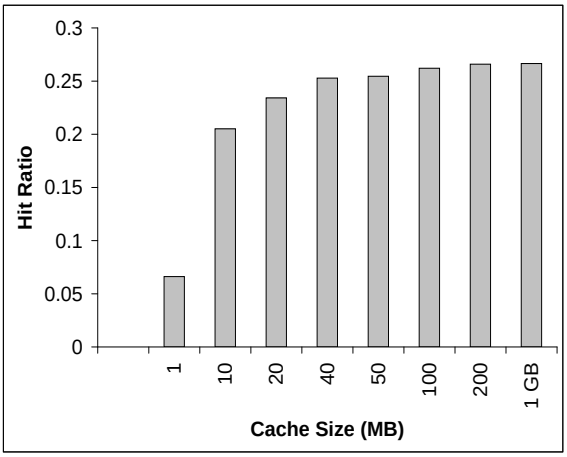
(c) Cache Size: 20MB

Figure 4: Hit Ratio of Cache Replacement Strategies

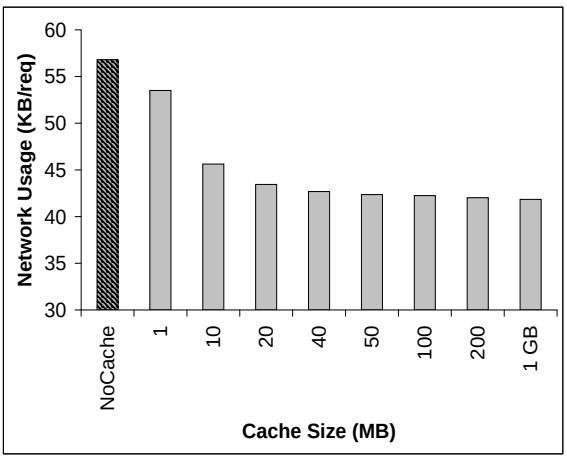
width usage, and throughput). We run `CacheCW` with different sizes of cache and compare the results together. Figure 5 (a) shows that a cache size equal to 100MB will result in maximum throughput (i.e., 315 request per minute). Increasing the cache size has a slightly negative impact on the throughput. This is mainly because of the more overhead incurred for probing larger *Cache Look-up Table* by the portal when the cache size increases. Figure 5 (b) proves this claim. For cache sizes larger than 100MB there is a very slight improvement in cache hit ratio. But as said before, due to the overhead, it does not impact the throughput positively. According to Figure 5 (c) cache sizes larger than 100MB have an insignificant effect on reducing network bandwidth usage. The experiments show that the optimal value for cache size



(a) Throughput



(b) Hit Ratio



(c) Network Bandwidth Usage

Figure 5: Effect of Cache Size on Performance

is around 100MB. However, this value depends on the system parameters such as the number of providers, number of objects per provider, access and update patterns.

5 Related Work

In (Luo & Naughton 2001), caching dynamic Web pages at a proxy server is enabled by sending and caching some programs on the proxy server. These programs generate the dynamic part of some Web pages while the static part can be directly provided from the cache.

A database accelerator increases the performance by reducing the communication between the application server and the database server. It also reduces the load on the back-end database resulting in more scalability (Oracle Corporation 2001*a*, TimesTen Inc. 2002). Products such as *Oracle Application Server Database Cache* (Oracle Corporation 2001*a*) and *TimesTen* (TimesTen Inc. 2002) enable this kind of caching.

Caching the result of dynamic Web pages at Web application server can reduce the workload on the Web application server and back-end database (Selcuk et al. 2001, Anton et al. 2002). Under a hit, the Web application server answers the request using the cache if it is still valid. Changes in back-end database invalidate relevant Web pages that use the modified data. Current application servers such as *BEA WebLogic Application Server* (http://www.bea.com) , *IBM WebSphere Application Server* , and *Oracle Application Server* support caching dynamic Web pages.

Cache servers can be deployed in front of Web application servers. This type of caching solution is known as *server acceleration*. It intercepts requests to the Web application server and either answers the request (if the result is cached) or forwards the request to the origin server. After a cache miss, the server accelerator caches any cacheable result returned by the origin server and forwards the reply back to the requester. Some examples include *IBM WebSphere Cache Manager* , and *Oracle 9i AS Web Cache* (Oracle Corporation 2001*b*). They promise caching dynamic and personalized Web pages.

Companies such as *Akamai* (http://www.akamai.com), and *Digital Island* (http://www.digitalisland.com) have been providing Content Delivery/Distribution Network (CDN) services for several years. CDN services are designed to deploy cache/replication servers at different geographical locations called *edge servers*. The first generation of these services was designed to cache static objects such as HTML pages, image, audio and video files. Nowadays, *Edge Side Includes* (ESI) (http://www.esi.org) enables the definition of different cacheability for different fragments of an object. Processing ESI at these servers enables dynamic assembly of objects at edge servers which otherwise may be done at server accelerator, proxy server or browser.

Some applications may need a customized caching technique. Therefore, the existing caching solutions might be insufficient. Application level caching is normally enabled by providing a cache API, allowing application writers to explicitly manage the cache to add, delete, and modify cached objects (Challenger et al. 1999, Degenaro, Iyengar & Ruvellou 2001).

The performance of individual cache servers increases when they collaborate with each other by replying each other's misses. Protocols such as *Internet Communication Protocol* (ICP) (Wessels & Claffy 1997), *Summary Cache* (Fan, Cao & Broder 2000), and *Cache Array Routing Protocol* (CARP) (Microsoft Corporation 1997) enable collaboration between proxy servers to share their contents. *ICP* was developed to enable querying of other proxies in order to find requested Web objects. In *Summary Cache*, each cache server keeps a summary table of

the content of the cache at other servers to minimize the number of ICP messages. *CARP* is rather a routing protocol which uses a hash function to determine the owner of a requested object in an array of proxy servers.

Maintaining cache consistency has been studied in (Ramamritham et al. 2000, Deolasee et al. 2001, Duvuri et al. 2000, Olston & Widom 2002, Cao & Ozsu 2002, Liu & Cao 1998).

Web Cache Invalidation Protocol (WCIP) enables maintaining consistency using invalidations and updates. In server-driven mode, cache servers subscribe to invalidation channels maintained by an invalidation server. The invalidation server sends invalidation messages to channels. These invalidation messages will be received by cache servers. In the client-driven mode cache servers periodically synchronize the objects with the invalidation server. The interval depends on consistency requirements (Li, Cao & Dahlin 2001).

Caching policies for Web objects are studied in (Aggrawal, Wolf & Yu 1999, Cao & Irani 1997, Cheng & Kambayashi 2000).

WEAVE (Florescu et al. 2000, Yagoub, Florescu, Valduriez & Issarny 2000) is a Web site management system which provides a language to specify a customized cache management strategy.

Triggers can be deployed to detect changes on the back-end database and invalidate cached objects. *Oracle Web Cache* (Oracle Corporation 2001*b*) uses a time-based invalidation mechanism (Anton et al. 2002).

Server logs can be used to detect changes and invalidate relevant entries, as proposed in *CachePortal* (Selcuk et al. 2001). It intercepts and analyzes three kinds of system logs to detect changes on base data and invalidate the relevant entries. These logs include HTTP request/delivery logs to determine requested page, query instance request/delivery logs to determine the query issued on the database based on the user query, and database update logs. A sniffer module finds the map between query instances and URLs based on HTTP and query instance logs and generates a QI/URL map table. An invalidator module uses the database update logs and invalidates the cached copies based on the updates and the QI/URL map table.

The *Data Update Propagation (DUP)* algorithm (Challenger et al. 1999) uses object dependence graph for the dependence between cached objects and the underlying data. The cache architecture is based on a cache manager which manages one or more caches. Application programs use an API to explicitly manage caches to add, delete, and update cache objects.

Dynamai uses an event-based invalidation for cache objects. Two kinds of events may change and therefore invalidate a dynamic Web page in the cache: First, the content of the database may change by the application through the Web interface. Second, it can be changed by an external event such as system administrator or another application. In the first case, the application can monitor incoming requests, and if they cause an update on the database, affected query instances will be invalidated. In the second case, the invalidation mechanism will be programmed in a script file and executed by the administrator to invalidate appropriate query instances. For request-based invalidation, some dependency and event rules are defined by the application developer or system administrator. They identify all dependencies between Web pages and all the events that a request may cause which in turn may invalidate other Web pages.

6 Conclusions and Future Work

In this paper, we have studied portals as a growing class of Web-based applications and addressed the importance of providing fast response time. We discussed the limitations of existing solutions in providing an effective caching solution in portals, and introduced a collaborative caching technique to address these limitations.

The experimental results show that the collaborative caching strategy can improve the performance by reducing the network bandwidth usage and increasing the throughput. However, its performance depends on effective collaboration between the portal and providers.

Further experiments are needed to determine the most effective strategies for regulating heterogeneous caching scores from different providers. Moreover, the maximum number of objects each provider should calculate the cache-worthiness for and keep an entry in *cache validation table* needs to be further investigated. This number will depend on the affordable processing and space overhead for calculating and storing cache-worthiness scores by providers, along with the overhead for storing and updating the entries in the *cache validation table*.

References

Aggrawal, C., Wolf, J. L. & Yu, P. S. (1999), Caching on the World Wide Web, *in* 'IEEE TKDE', Vol. 11.

Anton, J., Jacobs, L., Liu, X., Parker, J., Zeng, Z. & Zhong, T. (2002), Web Caching for Database Aplications with Oracle Web cache, *in* 'ACM SIGMOD'. Oracle Corporation.

Breslau, L., Cao, P., Fan, L., Phillips, G. & Shenker:, S. (1999), Web Caching and Zipf-like Distributions: Evidence and Implications, *in* 'IEEE INFOCOM', pp. 126–134.

Cao, P. & Irani, S. (1997), Cost-Aware WWW Proxy Caching Algorithms, *in* 'The USENIX Symposium on Internet and Systems'.

Cao, Y. & Ozsu, M. T. (2002), 'Evaluation of Strong Consistency Web Caching Techniques', *WWW* **5**(2), 95–124.

Challenger, J., Iyengar, A. & Dantzig, P. (1999), A Scalable System for Consistently Caching Dynamic Web Data, *in* 'IEEE INFOCOM'.

Cheng, K. & Kambayashi, Y. (2000), LRU-SP: A Size-Adjusted and Popularity-Aware LRU Replacement Algorithm for Web Caching, *in* 'IEEE Compsac', pp. 48–53.

Degenaro, L., Iyengar, A. & Ruvellou, I. (2001), Improving Performance with Application-Level Caching, *in* 'International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR)'.

Deolasee, P., Katkar, A., Panchbudhe, A., Ramamritham, K. & Shenoy, P. (2001), Adaptive Push-Pull: Disseminating Dynamic Web Data, *in* 'The Tenth World Wide Web Conference (WWW-10)'.

Duvuri, V., Shenoy, P. & Tewari, R. (2000), Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web, *in* 'IEEE INFOCOM'2000'.

Fan, L., Cao, P. & Broder, A. (2000), Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol, *in* 'IEEE/ACM Transactions on Networking', Vol. 8.

Florescu, D., Yagoub, K., Valduriez, P. & Issarny, V. (2000), WEAVE: A Data-Intensive Web Site Management System, *in* 'The Conference on Extending Database Technology (EDBT)'.

Kossmann, D. & Franklin, M. J. (2000), Cache Investment: Integrating Query Optimization and Distributed Data Placement, *in* 'ACM TODS'.

Li, D., Cao, P. & Dahlin, M. (2001), 'WCIP: Web Cache invalidation Protocol', http://www.ietf.org/internet-drafts/draft-danli-wrec-wcip-01.txt.

Liu, C. & Cao, P. (1998), Maintaining Strong Cache Consistency in the World-Wide Web, *in* 'International Conference on Distributed Computing Systems'.

Luo, Q. & Naughton, J. F. (2001), Form-Based Proxy Caching for Database-Backed Web Sites, *in* 'VLDB Conference', pp. 191–200.

Mahdavi, M., Benatallah, B. & Rabhi, F. (2003), Caching Dynamic Data for E-Business Applications, *in* 'International Conference on Intelligent Information Systems (IIS'03): New Trends in Intelligent Information Processing and Web Mining (IIPWM)'.

Microsoft Corporation (1997), 'Cache Array Routing Protocol and Microsoft Proxy Server 2.0', http://www.mcoecn.org/WhitePapers/Mscarp.pdf. White Paper.

Olston, C. & Widom, J. (2002), Best-Effort Synchronization with Source Cooperation, *in* 'ACM SIGMOD'.

Oracle Corporation (2001*a*), Oracle9i Application Server: Database Cache, Technical report, Oracle Corporation, http://www.oracle.com.

Oracle Corporation (2001*b*), Oracle9iAS Web Cache, Technical report, Oracle Corporation, http://www.oracle.com.

Ramamritham, K., Deolasee, P., Kathar, A., Panchbudhe, A. & Shenoy, P. (2000), Dissemination of Dynamic Data on the Internet, *in* 'DNIS 2000'.

Selcuk, K., Li, W.-S., Luoand, Q., Hsiung, W.-P. & Agrawal, D. (2001), Enabling Dynamic Content Caching for Database-Driven Web Sites, *in* 'ACM SIGMOD Conference'.

TimesTen Inc. (2002), Mid-Tier Caching, Technical report, TimesTen Inc., http://www.timesten.com.

Wessels, D. & Claffy, K. (1997), 'Application of Internet Cache Protocol (ICP), version 2', Network Working Group, Internet-Draft.

Wong, S. (1999), 'Estimated $4.35 Billion Ecommerce Sales at Risk Each Year', http://www.zonaresearch.com/info/press/99-june30.htm.

Yagoub, K., Florescu, D., Valduriez, P. & Issarny, V. (2000), Caching Strategies for Data-Intensive Web Sites, *in* 'VLDB Conference', Cairo, Egypt.

Zona Research Inc. (2001), 'Zona Research Releases Need for Speed II', http://www.zonaresearch.com/info/press/01-may03.htm.