Cooperative Data Placement and Replication in Edge Cache Networks

Lakshmish Ramaswamy
Dept. of Computer Science
University of Georgia
Athens, GA 30602
laks@cs.uga.edu

Arun Iyengar
IBM TJ Watson Research Center
Hawthorne, NY 10532
aruni@us.ibm.com

Jianxia Chen
Dept. of Computer Science
University of Georgia
Athens, GA 30602
jxchen@uga.edu

***Abstract*— Cooperation among individual caches has proven to be an effective strategy to improve the scalability and performance of edge cache networks delivering dynamic web content. To date, research in the area of cooperative edge caching has mainly focused on serving client requests and maintaining freshness of cached documents. However, designing mechanisms to effectively manage the available resources is an important challenge that can have significant impact on the performance of an edge cache network. In this paper we propose a novel data placement scheme, called the *utility-based placement* scheme, which is not only sensitive to the ongoing cooperation in the edge cache but also takes into account the various costs and benefits of storing a data-item at an individual edge cache. At the heart of proposed scheme is a utility function that quantifies the usefulness of storing a data-item at a particular edge cache. Experiments show that the proposed scheme provides significant performance benefits.**

I. Introduction

The amount of dynamic content on the World Wide Web (web) continues to grow at a rapid phase. Unlike static web pages, the dynamic web pages are generated on the fly when a user request arrives at the web server infrastructure. While dynamic content has enabled several new kinds of applications, high costs of generating them coupled with their stringent freshness requirements have posed new challenges to the scalability of the web. Recently, edge computing has emerged as a popular technique to efficiently deliver dynamic web content to the clients [1], [14], [19]. The fundamental philosophy of edge computing is to move data, and possibly parts of the application closer to the user, so that client-requests reach the data/service they need by traversing just a few hops within the Internet.

However, many of the existing edge cache networks have not been able to fully harness the potentials of the edge computing technology, which has limited their ability to effectively deliver dynamic web content. In the cooperative edge cache grid project (henceforth cooperative EC grid), we are exploring the capabilities of cooperation among the caches of an edge network as a tool for efficient, scalable and reliable delivery of dynamic web content [16]. While the basic idea of collaboration among multiple caches has been studied by several previous works, cache cooperation in these systems is limited only to handling cache misses. In contrast, we utilize the principle of cache cooperation to enhance the performance of edge cache networks in several related, yet distinct ways, including collaborative miss processing, cooperative document freshness maintenance, cooperative failure handling and collaborative cache management. Further, unlike earlier cooperative caching systems that only support the TTL-based weak document freshness mechanisms, the cooperative EC grid incorporates stronger, server-driven document consistency schemes.

This paper considers the problem of cache management in cooperative edge cache networks. There are two primary approaches to cache management, namely, *data placement* and *data replacement*. While several document replacement policies have been proposed and studied [2], [3], [12], [13], the data placement problem has received very little research attention. Moreover, very few cache management schemes take into account the overheads of server-driven document consistency mechanisms. In addition, they are not sensitive to the cooperation that might be going on among the caches of the edge network. Most edge cache networks employ a very simple data placement scheme wherein a cache retrieving a data-item stores it locally irrespective of the overheads involved in maintaining its freshness or its availability in the cooperative cache group. However, this simple strategy induces high network loads, and it also increases resource contention at individual caches.

In this paper we propose a novel *utility-based data placement scheme* for cooperative edge cache networks. This technique quantifies the various costs and benefits of storing a given data-item at a particular edge cache, and it estimates the *utility* of the data copy to the edge cache network. A copy of the data-item is stored at the cache only if the benefits outweigh the costs. Furthermore, the proposed scheme explicitly considers the ongoing cooperation within the edge cache network while computing the costs-benefits tradeoff of storing the data-items. Our simulation-based experiments show that the utility-based placement scheme provides considerable performance benefits.

II. Cooperative Edge Cache Grid

Cooperative edge cache grid is a large-scale edge cache network for delivering dynamic web content, whose design

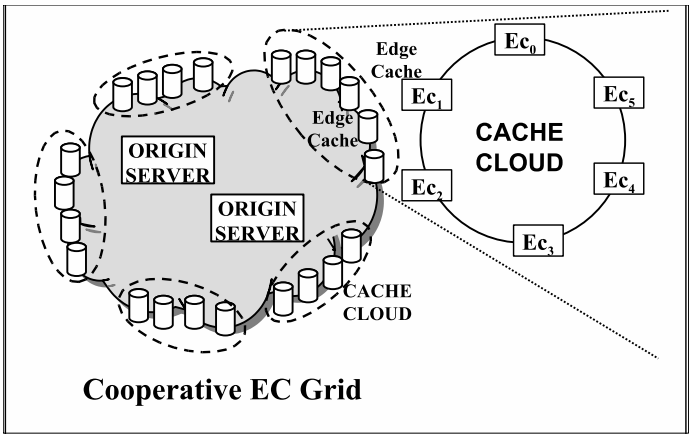

Fig. 1: The Cooperative Edge Cache Grid

incorporates several unique techniques for supporting cost effective cooperation among its caches. A cooperative EC grid typically contains a large number of edge caches and a few origin servers. Unlike many edge cache networks, the cooperative EC grid supports server-driven data consistency mechanisms, wherein the origin server initiates an update message when a data-item is modified, which is communicated to all the caches holding a copy of the data.

Cache clouds form the fundamental unit of cooperation in the cooperative EC grid. A cache cloud is a group of several edge caches that are located in close network proximity. The caches belonging to a cloud collaborate with one another primarily for the purposes of resolving misses and maintaining consistency of cached data. A cache which does not have the data to serve an incoming request (i.e. the cache suffers a local miss) first attempts to obtain it from other caches in the cloud that might have the data, rather than contacting the origin server immediately. When a data-item undergoes modification, the origin server sends an update message to one of the caches in each cloud, which is then collaboratively disseminated among the caches belonging to each cloud. In our design, all caches in a cloud share the lookup and update of the data-items being cached in the cloud. A cache that is currently handling the lookup and update operations of a data-item Dc is called Dc's *beacon point*. We have designed a *dynamic hashing mechanism* to assign data-items to beacon points. The dynamic hashing mechanism and the data lookup and update protocols are described in detail in [16]. Currently, cooperative EC grid supports caching at the granularity of entire web documents or parts of web documents known as fragment.

III. Problem Statement

In this section we formally state the data placement problem. Consider a cache cloud with M caches represented as $CCloud_i = \{Ec_0, Ec_1, \ldots, Ec_{M-1}\}$. Suppose the cache Ec_l receives a client request for data-item Dc, which is not available locally (i.e., the request is a local miss). The data-item may be available in one or more caches within the cloud, or it might have to be retrieved from the origin server. Suppose the data-item is available in the cloud, then should Ec_l make a local copy of the data-item Dc? On the contrary, if the data-item is not available at any of the caches in the cloud, should it be stored in one of the caches in the cloud, and if so, where (in which cache) should it be stored? In contrast, the problem of cache replacement is to decide which data-items that are currently in the cache should be evicted in order to create disk-space for incoming data. Since, the cooperative EC grid caches data at the granularity of documents or parts of documents, the terms *data-item* and *document* and the terms *data placement* and document placement are used interchangeably in the remainder of this paper.

The straightforward solution to the data placement problem, which we refer to as the *ad hoc data placement* is to store the document at every cache that receives a client request for it. However, this simple scheme can lead to *uncontrolled data replication* within the cache cloud, which in turn can result in several performance limitations. First and foremost, the data consistency maintenance traffic in the clouds becomes very heavy, thereby imposing high communication and processing overheads. Second, if the storage-space at each cache is limited, then each additional copy of a document causes the eviction of other documents from the caches. Hence, the number of unique documents available within each cache cloud shrinks. Further, the average amount of time the documents live in the cache before they are evicted decreases. The cumulative effect of these two is the decrease in the cumulative hit rates of the cache clouds.

These performance problems are the manifestations of two important shortcomings of the ad-hoc document placement scheme. First, the ad-hoc scheme does not consider the costs of storing document copies. Storing an additional copy of the document within a cache cloud has two associated costs: (1) The document update message has to be communicated to

an additional edge cache; and (2) The new copy consumes extra storage space. The ad hoc scheme ignores these costs while making document placement decisions. The second shortcoming of the ad hoc scheme is that it does not take into account the possibility of the document being available in other caches in the cache cloud − it is essentially *blind* to the contents of the other caches in the cloud.

In order to overcome these limitations, this paper proposes the utility-based cooperative placement scheme which is described in the next section.

IV. Utility-based Cooperative Placement Scheme

As the name suggests, the placement decisions in the utility-based cooperative scheme rely upon the usefulness of a document-copy to the cache storing it, and to the entire cache cloud. The usefulness of the document of a document copy is termed as the *utility value* of the document copy, and is represented as $Utility(Dc)$ for document copy Dc. Our document placement scheme is also sensitive to the cooperation among the caches belonging to a cloud. In fact, our formulation of the utility value function takes into account the number of copies of the document, if any, that currently exist in the various caches of the cooperative cloud. When a cache retrieves a document it calculates the document's utility value. Based on this utility value the cache decides whether or not to store the document.

The utility of document copy Dc estimates the benefit-to-cost ratio of storing and maintaining the new copy. A higher value of utility indicates that benefits outweigh the costs, and vice-versa. As mentioned before, the costs of caching dynamic documents are two fold, namely, the *consistency maintenance costs* and the *storage costs*. Consistency maintenance costs correspond to the overheads involved in maintaining the freshness of the cached copy of a document. Storage costs correspond to physical storage space needed for storing the document copy. On the other hand caching a document copy benefits the edge network by reducing the server load, network load and the client latency.

Our formulation of the utility function has four components. Each of these components quantifies one aspect of the interplay between benefits and the costs. We now briefly discuss each of these components. Throughout this discussion we assume that a cache Ec_l has retrieved the document copy Dc, and is calculating its utility value to decide whether to store it locally.

A. Document Availability Component

The document availability component, represented as $DAIC(Dc, Ec_l)$, quantifies the improvement in the availability of the document in the cache cloud achieved by storing the document copy at Ec_l. Availability of a document Dc is defined as the probability that an in-coming request for the document can be served by one of the caches in the cache cloud. Improving the availability of a document increases the probability that a future request for the document would be served within the cache cloud.

The availability of a document depends upon the number of copies of the document existing within the cloud. Further, the availability of the document is also directly proportional to the *reliability* of the caches storing those copies. Reliability of a cache denotes the probability that the cache is alive at any given point in time. Let $Rblty(Ec_p)$ denote the reliability of the cache Ec_p. The current document availability of document Dc is computed as $CAvblty(Dc) = \sum_{Ec_p \text{contains} Dc} Rblty(Ec_p)$. Note that $CAvblty(Dc)$ becomes 0 when Dc is not currently stored at any cache in the cache cloud. If an additional copy of the document is stored at the cache Ec_l, it improves the document availability by $Rblty(Ec_l)$. Document availability improvement component is now defined as:

$$DAIC(Dc, Ec_l) = \begin{cases} MaxValue & \text{If CAvblty(Dc) = 0} \\ \frac{Rblty(Ec_l)}{CAvblty(Dc)} & \text{Otherwise} \end{cases}$$

Here $MaxValue$ denotes a large positive real number. Observe that $DAIC(Dc, Ec_l)$ acquires a very high value if there are no existing copies of Dc within the cache cloud, whereas its value would be very small if several copies already exist in the cloud.

B. Disk-Space Contention Component

This component captures the storage costs of caching the document copy at Ec_l. An important point that needs to be noted is that the cost associated with storing a given document may vary from cache to cache. It is much more expensive to store the same document in a smaller (or heavily accessed cache) than a larger (or a less busy cache), since in the first scenario documents that are more recently accessed would have to be evicted. Hence any reasonable quantification of storage costs should take into account the contention at the cache where the document is stored. Accordingly, we quantify this component in terms of the disk-space contention at Ec_l.

The disk-space contention at the cache Ec_l determines the time duration for which the document can be expected to reside in the cache Ec_l before it is replaced. Suppose the cache cloud already contains a copy of the document at cache Ec_p. If the disk space contention at that cache is lower than that of cache Ec_l, then even if we were to make a new copy at the cache Ec_l, the new copy would be removed earlier than the existing copy. Hence, the benefits of storing the new copy are limited, whereas it adds to the disk-space contention at Ec_l. On the other hand if the disk-space contention at Ec_l is significantly lower than that of Ec_p, the new copy is expected to remain in the cache much longer than the copy at Ec_p benefiting the cache cloud for a longer duration. We use the concept of *cache expiration age* [15] to accurately quantify the disk space contention at edge caches. The higher the cache expiration age of a cache, the lower is its disk space contention, and vice versa.

If the cache Ec_l retrieves the document Dc from another cache in the cache cloud, say Ec_p, then the disk-space component for document Dc is defined as the ratio of the

expiration age of Ec_l to the expiration age of Ec_p. However, if the document is not available in the cache cloud, the disk-space component is assigned a large positive value, since it is always advantageous to store a copy of Dc.

$$DsCC(Dc, Ec_l) = \begin{cases} \frac{CacheExpAge(Ec_l)}{CacheExpAge(Ec_p)} & \text{If } Dc \text{ is retrieved from} Ec_p \\ MaxValue & \text{If } Dc \text{ is retrieved from server} \end{cases}$$

$DsCC(Dc, Ec_l)$ acquires a higher value when the expiration age of Ec_p is smaller and vice versa. A higher value of $DsCC(Dc, Ec_l)$ implies that the new document copy is likely to remain in the cache cloud for a longer duration than the old one, and it is beneficial to store this copy.

C. Consistency Maintenance Component

Denoted by $CMC(Dc, Ec_l)$ this component accounts for the costs incurred for maintaining the consistency of the new document copy at Ec_l, and the advantages that are obtained as a result of storing Dc at Ec_l by avoiding the cost of retrieving the document from other caches on each access. Suppose we observe the access and the update patterns of the document Dc at cache Ec_l for t_w time units. Let there be $AccCount$ accesses during this time period. An access A_v is termed as an *updated-access* if the document Dc is updated between the accesses A_{v-1} and A_v, and as *nonupdated-access* otherwise. Let $NonUpCount$ represent the number of nonupdated-accesses within the time duration t_w. The consistency maintenance component is obtained as:

$$CMC(Dc, Ec_l) = \frac{NonUpCount}{AccCount}$$

A high value of $CMC(Dc, ECl)$ indicates that the document Dc is accessed more frequently than it is updated, and vice-versa.

D. Access Frequency Component

The final component of our utility function quantifies how frequently the document Dc is accessed in comparison to other documents in the cache. If access frequency of Dc at the cache Ec_l is high when compared to other documents in the cache, it is advantageous to store it. Let $ReqCount(Dc, Ec_l, T_z)$ indicate the number of requests to the document Dc at cache Ec_l in the past T_z time units, $TotReqs(Ec_l, T_z)$ denote the total number of client requests received at cache Ec_l in the past T_z time units, and let $NumDocs(Ec_l)$ represent the total number of documents currently cached at Ec_l. Therefore, the mean of the number of requests received by the documents in the cache is given by $MnReqs(Ec_l, T_z) = \frac{TotReqs(Ec_l, T_z)}{NumDocs(Ec_l)}$. The access frequency component of the utility function is computed as below.

$$AFC(Dc, Ec_l) = \frac{ReqCount(Dc, Ec_l, T_z)}{MnReqs(Ec_l, T_z)}$$

E. Document Placement Algorithm

The above four components form the building blocks of the utility function. We observe that for each component, a higher value implies that benefits of storing Dc are higher than the overheads, and vice-versa. We define the Utility of storing the document Dc at cache Ec_l, denoted as $Utility(Dc, Ec_l)$, to be a weighted linear sum of the above four components.

$$Utility(Dc, Ec_l) = W_{DAIC} \times DAIC(Dc, Ec_l) + W_{DsCC} \times DsCC(Dc, Ec_l) + W_{CMC} \times CMC(Dc, Ec_l) + W_{AFC} \times AFC(Dc, Ec_l)$$

In the above equation W_{DAIC}, W_{DsCC}, W_{CMC}, and W_{AFC} are positive real constants such that $W_{DAIC} + W_{DsCC} + W_{CMC} + W_{AFC} = 1$. These constants are assigned values reflecting the relative importance of the corresponding component of the utility function to the performance of the system. For example, if the documents in the system have high update rates, then W_{CMC} is assigned a high value. Similarly if disk-space availabilities at the caches are limited W_{CMC} would be set to a high value.

If the value of the utility function $Utility(Dc, Ec_l)$ exceeds a threshold, represented as $UtlThreshold(Ec_l)$, then Dc is stored at cache Ec_l. Otherwise the edge cache Ec_l does not store a local copy of Dc. Concretely, suppose a client-request for document Dc encounters a local miss at cache Ec_l. Then Ec_l retrieves the document either from another cache in the cache cloud, or from the origin server. If the document Dc does not currently exist in the cache cloud, Ec_l stores the document Dc and registers it with the beacon point of Dc. This is because, in this scenario the $DAIC$ and the $DsCC$ components of its utility function assume very high values. If at least one cache in the cache cloud contains Dc, then Ec_l does not decide immediately whether to store the document. Instead, it monitors the pattern of requests and updates to the document for fixed time duration in order to evaluate its utility value. At the end of this time duration, the cache evaluates the utility function $Utility(Dc, Ec_l)$. The cache stores the document Dc, only if $Utility(Dc, Ec_l) \geq UtlThreshold$, in which case it registers the new copy with its beacon point. The pseudo-code of the utility-based document placement is outlined in Algorithm 1.

We note that the utility function accurately quantifies the costs and benefits of caching dynamic documents, and it can also be adopted as the cost function in cost-based document replacement policies such as the Greedy-dual size [3], Greedy-dual* [2] algorithms.

F. Discussion

In this section we compare and contrast two approaches to cache management, namely, the document placement policies and document replacement policies. As described in Section III, the problem of document placement is to decide whether a document Dc that has been retrieved by a cache Ec_l, should be stored at the cache Ec_l, or whether it should be stored at a different cache in the cloud, or should just be discarded after serving the user request. In contrast the

Algorithm 1 Utility-based Document Scheme: Algorithm performed by a cache on receiving a request for document Dc

```
if Dc is available locally then
  Serve request and update document statistics
else
  Contact Dc's beacon point and obtain lookup information
  if Dc is not available within cache cloud then
    Obtain Dc from the origin server
    Store Dc locally
    Register the new copy with beacon point
  else
    Obtain Dc from one of the caches currently holding Dc
    Along with Dc obtain the responding cache's Expiration Age
    if A DocumentRecord for Dc exists in DocumentMonitorList then
      Update the DocumentRecord
      if DocumentRecord of Dc has resided in the DocumentMonitorList for more than MonitorDurationThreshold then
        Remove DocumentRecord of Dc
        Compute the utility function Utility(Dc, Ecl)
        if Utility(Dc, Ecl) ≥ UtlThreshold(Ecl) then
          Store Dc locally
          Register the new copy with beacon point
        else
          Discard the DocumentRecord of Dc
        end if
      end if
    else
      Create a DocumentRecord for Dc and store it in DocumentMonitorList
    end if
  end if
end if
```

document replacement problem can be summarized as follows: Suppose the cache Ec_l wants to store a document Dc locally, but its storage-space is already full. Now the cache has to evict some of the documents that are currently stored by it, so that the incoming document can be stored. In this scenario, deciding which documents to evict is known as the document replacement problem. While researchers have proposed many techniques to address the document replacement problem [2], [3], [12], [13], the studies on the document placement problem are relatively few [9], [15], [18].

Both document placement and document replacement problems are concerned with managing the available resources such as disk-space and network bandwidth effectively. The goal of both problems is to maximize the benefits of caching. The document placement approach may be considered as a proactive approach to resource management, whereas document replacement can be regarded as a reactive approach, since it is triggered when the available resources such as disk-space become insufficient. We contend that caches need to implement a good document placement as well as a good document replacement scheme in order to effectively manage their resources.

In the context of cooperative edge cache networks, incorporating a proactive approach through good document placement schemes provides some unique benefits. First, utilizing resources judiciously even when the resource consumption at individual caches have not reached their respective limits is likely to benefit the performance of the cooperative cache group. For example, suppose a cache Ec_l retrieves a document Dc which is being modified very frequently. If the cache stores this document, Dc's beacon point would have to communicate Dc's updates to Ec_l until it is replaced, which would place significant load on the beacon point and on the network. This cost could have been avoided had Ec_l made the placement decision judiciously. Similarly, suppose the cache Ec_l retrieves a document Dc_1, which is available at many other caches in the cloud. Then making an additional copy at Ec_l blindly, could result in the replacement of one or more documents, which may not be available in any other caches within the cache cloud. This affects the cumulative hit rate of the cache cloud.

The second advantage of the document placement approach for cache management is the ease and efficiency of its implementation in a cooperative cache group setting. For example, collaborative cache management schemes require information about the availability of documents in various caches of the group. A typical cooperative edge cache network maintains this information in order to facilitate the document retrieval process (in the cooperative EC grid this information is maintained at the document's beacon point). This information can be piggy-backed on the document lookup data sent to a cache that is attempting to retrieve the corresponding document. Therefore, the edge caches can obtain all the information needed to make the placement decisions at no extra-cost. In contrast, if a cache has to implement a cooperative replacement scheme, it has to obtain the availability information for a subset of documents that would be the probable candidates for eviction which could involve additional communication costs.

V. Experimental Evaluation

This section reports the experimental study we have conducted to evaluate the proposed document placement scheme. We compare the utility-based document placement scheme to two other schemes. The first scheme is the ad-hoc document placement mechanism, wherein a cache stores every document for which it receives a client-request. The second is the beacon-point placement. In this scheme, each document is only cached at its beacon point. Any other cache receiving a request from a client retrieves the document from its beacon point and serves the request. The experimental study was conducted through trace-based simulations of the cooperative EC grid. The simulator can be configured to simulate any of the above three document placement schemes. Each cache in the edge cache network receives requests continuously from a request trace file. If the requested document is available within the cache it is recorded as a local hit. If the document is retrieved from another cache, it is a group hit. A request is a miss, if none of the caches in the cloud has the document. The *local hit rate* of a cache cloud is defined as the ratio of the sum of the local hits at the caches belonging to the cloud to the total number of requests received at those caches. The terms *group hit rate* and *miss rate* are analogously defined.

For our experimental study, we use a dataset which is derived from real request and update traces from a major

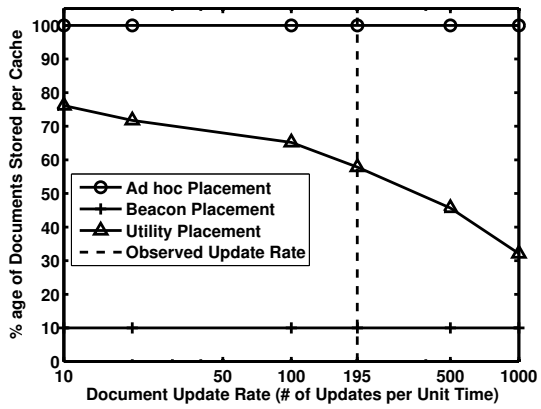

Fig. 2: Percentage of Documents Stored at Varying Update Rates (DsCC Turned Off)

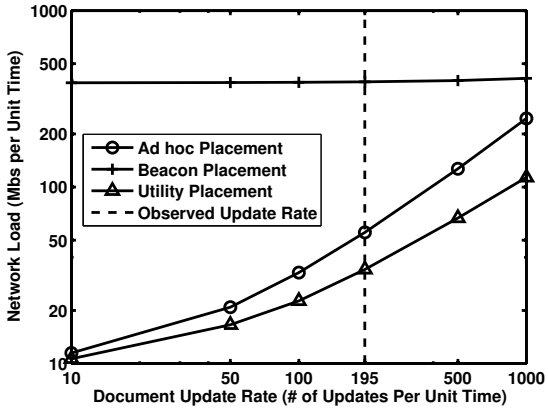

Fig. 3: Network Loads of Different Placement Schemes at Varying Update Rates (DsCC Turned Off)

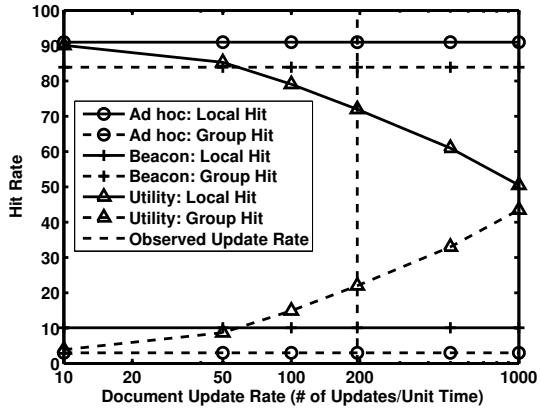

Fig. 4: Hit Rates of Different Placement Schemes at Varying Update Rates (DsCC Turned Off)

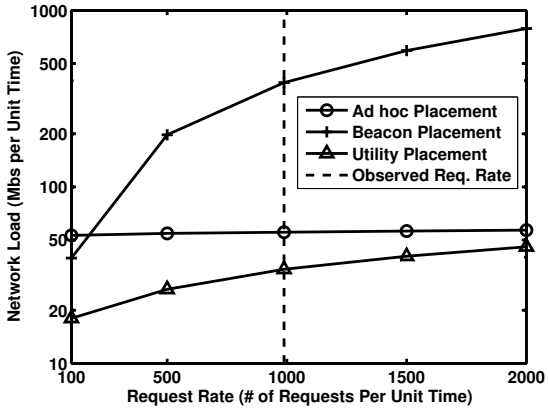

Fig. 5: Network Loads of Different Placement Schemes at Varying Request Rates (DsCC Turned Off)

IBM sporting event website[1](henceforth referred to as *Sydney*). The logs contained 52527 unique documents with the average size of each document being 65.3 Kilobytes. The requests were segregated based on their client-ids, and the requests from a few random clients were combined to generate the request-logs. Each cache was driven by one such request log. In these experiments we have simulated a cooperative EC grid consisting of 120 caches that were organized into 12 cache clouds each containing 10 caches. Since, the document placement schemes work at the level of individual cache clouds, we report the results on a representative cache cloud.

In the first set of experiments the caches are assumed to have unlimited amounts of disk-space. This implies that the storage costs of caching documents are minimal. Therefore the disk-space component of the utility function is turned off by setting W_{DsCC} to 0. The weights of availability, consistency maintenance, and access frequency components are all set to 0.33. In the first experiment the access rates at caches are fixed, whereas we vary the document update rate to study the effect of the five document placement policies.

The graph in Figure 2 shows the percentage of the total documents in the log that are stored at each cache in the cache cloud at various document update rates. The X-axis represents the document update rate in number of updates per minute on the log scale, and the Y-axis represents the percentage of documents cached. The vertical broken line indicates the observed document update rate in the real trace. As the ad hoc policy places each document at every cache which receives a request, almost all documents are stored at all caches. In contrast the beacon point placement stores each document only at its beacon point. Hence, each cache stores around 10% of the total documents. The percentage of documents stored per cache in the utility-based scheme varies with the update rate, indicating its sensitivity towards document update costs. In the utility-placement scheme, when the update rates are low, a large percentage of documents are stored at each cache, owing to the small consistency maintenance cost. As the update rate increases, the consistency maintenance component

[1]The Sydney 2000 Olympic Games Website

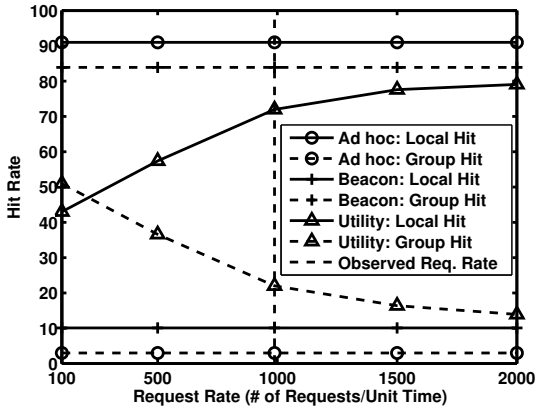

Fig. 6: Hit Rates of Different Placement Schemes at Varying Request Rates (DsCC Turned Off)

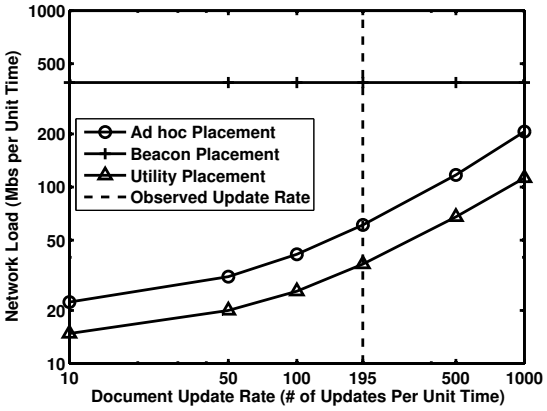

Fig. 7: Network Loads of Different Placement Schemes at Varying Update Rates(DsCC Turned On)

value (defined in Section IV-C) of the utility function decreases for all the documents, thereby leading to a decrease in the percentage of documents stored at each cache.

In order to demonstrate the benefits of being sensitive to the update costs when making document placement decisions, we plot the total network traffic in the clouds generated by various document placement policies in Figure 3. The results indicate that the utility-based document placement creates the least network traffic at all update-rates. The improvements provided by the utility-based placement scheme over the ad hoc placement scheme increase with increasing update rate. This is because while the number of replicas present in the cache cloud essentially remains a constant in the ad hoc placement scheme, the utility-based scheme creates fewer replicas at higher update rates, thereby reducing the consistency maintenance costs significantly. The network traffic produced by the beacon point caching is very high at all update-rates, as in this scheme only one copy of each document is stored per cache cloud. Hence, most of the network traffic is due to caches repeatedly accessing the single copies of the documents.

Figure 4 indicates the local and the group hit-rates of ad-hoc, beacon-point and utility-based placement schemes. The local hit rates and the group hit rates of ad hoc and beacon point placement schemes remain constant at all document update rates, since these schemes are not sensitive to document update costs. In contrast to these schemes the hit rate of the utility scheme varies with the document update rates. As the document update rate gets higher, the local hit rate of the utility scheme drops and its group hit rate increases.

In the second experiment of this set, we fix the document update rate to 195 updates per unit time and study the impact of varying document access rates on the performance of the three document placement schemes. Figure 5 indicates the total network traffic in the clouds induced by the three document placement policies when the access rates at the caches vary from 100 accesses per unit time to 2000 requests per unit time. The network load of the ad hoc placement remains almost constant at around 55 MBs per unit time. In this scheme, every cache stores all requested documents. Hence, the load due to document access is very low, and the network load is predominantly due to the document updates. The network load for beacon placement starts from a low value, but increases at a very rapid pace. The utility-based placement scheme takes into account the tradeoff between access and update rates, thereby resulting in significantly lower network loads at all request rates. Figure 6 shows the local and the remote hit rates of the three document placement schemes at various request rates. Similar to the previous experiment, the local and the group hit rates of the ad hoc and the beacon point placement schemes remain constant, while the local hit rate of the utility-based placement scheme increases and its group hit rate drops as the request rate increases. This is because the utility-based placement scheme stores more documents at each cache as the document access rate increases, thus increasing the fraction of requests that are served by the cache that receives the client request.

In the second set of experiments, we study the performance of the three document placement policies when the disk spaces available at the edge caches are limited. In these experiments the disk space at each cache is set to 25% of the sum of sizes of all documents in the trace. As the disk space is a limiting factor in this series of experiments, we turn on the disk-space component of the utility function. The weights of all the utility function components are set to 0.25.

Figure 7 indicates the total network traffic generated by the three document placement policies at various update rates. As in the previous experiment, the utility-based document placement places the least load on the network. However, the results in this experiment differ from the previous experiment considerably. The percentage improvement in the network load provided by the utility scheme over the ad hoc scheme is higher in the limited disk-space case at low document update rates. However, the percentage improvement in the unlimited disk space case grows much faster in the limited

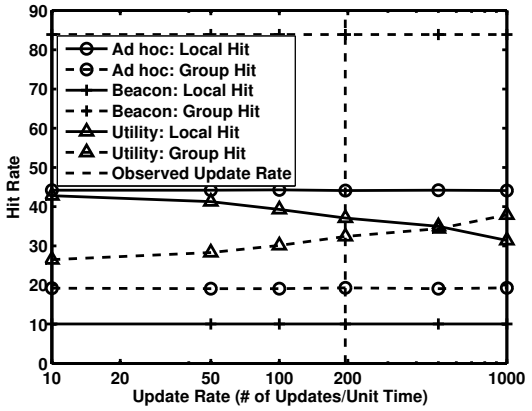

Fig. 8: Hit Rates of Different Placement Schemes at Varying Update Rates (DsCC Turned On)

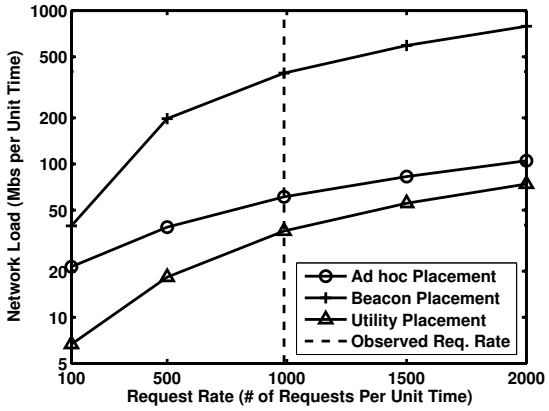

Fig. 9: Network Loads of Different Placement Schemes at Varying Request Rates (DsCC Turned On)

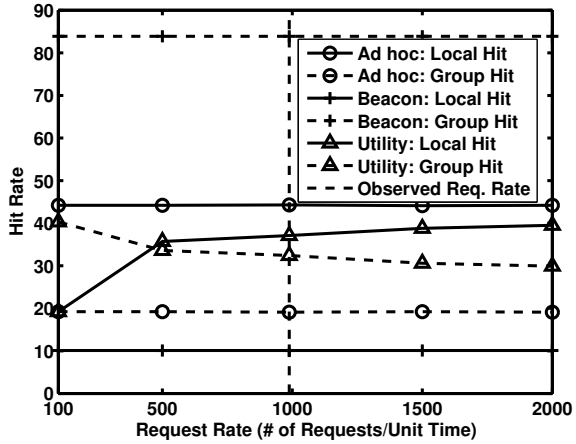

Fig. 10: Hit Rates Different Placement Schemes at Varying Request Rates (DsCC Turned On)

disk space scenario. These observations are the manifestations of the different roles the utility placement scheme is playing at different update rates in the limited disk space scenario. At low document update rates the utility scheme assumes the predominant role of reducing disk space contention at individual caches. Whereas at higher update rates its predominant effect is to reduce consistency maintenance costs.

Figure 8 shows the hit rates of ad-hoc, beacon-point and utility-based placement schemes for the limited disk space experiment at different update rates. As in the unlimited disk space scenario, the local hit rate of the utility scheme drops as the update rate increases, whereas its group hit rate starts to increase. A crucial observation here is that the cumulative hit rate (the sum of local and group hit rates) of the utility scheme is around 5.5% higher than that of the ad hoc scheme. This shows that when disk space becomes a limiting factor, the utility scheme uses the aggregate disk space available in the cache cloud more efficiently.

The final experiment evaluates the three document placement policies when the update rate is fixed while the request rate varies from 100 requests per unit time to 2000 requests per unit time. Figure 9 shows the network load induced by the three schemes, and Figure 10 indicates their local and group hit rates. The results are similar to the previous experiment, except that the local hit rate of the utility scheme increases and its group hit rate decreases as the request rate grows.

In summary, these experiments show that being sensitive to the various costs and benefits of dynamic content caching while making document placement decisions can significantly enhance the performance of the cooperative edge cache network.

VI. Related Work

The issue of cache management has received considerable attention from the web caching community. However, most of the research has focused on the document replacement problem [2], [3], [12], [13]. Of the several document replacement strategies that have been proposed in the literature, the cost-based document replacement schemes such as Greedy-dual size [3] and Greedy-dual* [2] are the ones that are most related to the work presented in this paper.

Although both the greedy-dual size algorithm and our utility-based document placement scheme adopt a cost-benefit approach for cache management, there are important differences between them. While the greedy-dual size algorithm is a document replacement scheme, the utility-based scheme is a document placement scheme which takes a proactive approach to cache management. Further, most current formulations of the greedy-dual size algorithm's cost function do not consider the document consistency costs, and hence, they are not directly applicable to caching dynamic documents. In this context, we note that our utility function can be adapted in a framework such as the greedy-dual size, to yield highly effective document replacement strategies for caches storing dynamic web content.

The schemes proposed by Korupolu et al. [9], and by Wu et al. [18] are among the few document placement strategies proposed in the literature. However, these schemes are targeted for caching static web content, and they do not consider the document consistency costs while making document placement decisions.

Research in the general area of cooperative web caching has been concentrated mainly on issues such as quantifying benefits of cooperation among proxy caches, designing cache-sharing protocols and cooperative architectures aimed at improving hit rates and document access latencies [5], [7], [8], [10], [17]. However, most of these schemes only support the weaker, TTL-based document consistency mechanisms. In contrast, Ninan et al. [11] propose a cooperative strategy for implementing lease-based consistency schemes. In addition there is a vast body of literature on delivery of dynamic content [4], [6], [14], [19]. However, very few of these works address the document placement problem.

Previously, we had outlined the utility-based document placement framework when describing the overall design architecture of cache clouds [16]. However, that paper does not discuss the exact mathematical formulations of the various components of the utility function that are presented here. Further, the study on the impact of request patterns on the performances of the various document placement schemes has not been previously reported. In short, the work presented in this paper is unique and it addresses an important problem in the area of edge cache networks.

VII. Conclusions

Cooperation among the caches of an edge cache network is an effective strategy to improve the scalability and efficiency of delivering dynamic web content. One of the important challenges in designing a cooperative edge network is to develop effective mechanisms for cache management. Towards this end, this paper proposed a novel utility-based data placement scheme in which the data placement decisions are based upon the costs and benefits of storing a data-item at a given cache. Furthermore, our scheme also takes into account the cooperation among the various caches of the edge cache network. We also presented a utility function comprised of four components each of which quantifies one aspect of the costs-benefits tradeoff of caching dynamic data. An experimental study of the proposed scheme has been reported in the paper, the results of which show that the utility-based placement scheme can yield significant performance benefits.

Acknowledgements

This research is supported by a grant from the UGA research foundation (UGARF).

[1] Akamai Technologies Incorporated. http://www.akamai.com.

[2] A. Bestavros and S. Jin. Popularity-Aware Greedy Dual-Size Web Proxy Caching Algorithms. In *Proceedings of the* 20^{th} *International Conference on Distributed Computing Systems(ICDCS-2000)*, April 2000.

[3] P. Cao and S. Irani. Cost-Aware WWW Proxy Caching Algorithms. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, 1997.

[4] J. Challenger, P. Dantzig, A. Iyengar, M. S. Squillante, and L. Zhang. Efficiently Serving Dynamic Data at Highly Accessed Web Sites. *IEEE/ACM Transaction on Networking*, 12(2), April 2004.

[5] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worell. A Hierarchical Internet Object Cache. In *Proceedings of the 1996 USENIX Technical Conference*, January 1996.

[6] A. Datta, K. Dutta, H. Thomas, D. VanderMeer, Suresha, and K. Ramamritham. Proxy-Based Acceleration of Dynamically Generated Content on the World Wide Web: An Approach and Implementation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, June 2002.

[7] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. In *Proceedings of ACM SIGCOMM 98*, September 1998.

[8] Internet Cache Protocol: Protocol Specification, Version 2, September 1997. http://icp.ircache.net/rfc2186.txt.

[9] M. R. Korupolu and M. Dahlin. Coordinated Placement and Replacement for Large-Scale Distributed Caches. *IEEE Transactions on Knowledge and Data Engineering*, 14(6), 2002.

[10] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson. Adaptive Web Caching: Towards a New Global Caching Architecture. *Computer Networks and ISDN Systems*, 30(22-23), November 1998.

[11] A. Ninan, P. Kulkarni, P. Shenoy, K. Ramamritham, and R. Tewari. Scalable Consistency Maintenance in Content Distribution Networks Using Cooperative Leases. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(4), July 2003.

[12] S. Podlipnig and L. Boszormenyi. A Survey of Web Cache Replacement Strategies. *ACM Computing Surveys*, 35(4), December 2003.

[13] K. Psounis and B. Prabhakar. A Randomized Web-Cache Replacement Scheme. In *Proceedings of IEEE-INFOCOM-2001*, April 2001.

[14] M. Rabinovich, Z. Xiao, and A. Aggarwal. Computing on the Edge: A Platform for Replicating Internet Applications. In *Proceedings of the* 8^{th} *International Workshop on Web Content Caching and Distribution*, September 2003.

[15] L. Ramaswamy and L. Liu. An Expiration Age-Based Document Placement Scheme for Cooperative Web Caching. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 16(5), May 2004.

[16] L. Ramaswamy, L. Liu, and A. Iyengar. Cache Clouds: Cooperative Caching of Dynamic Documents in Edge Networks. In *Proceedings of the* 25^{th} *International Conference on Distributed Computing Systems(ICDCS-2005)*, June 2005.

[17] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet. In *Proceedings of International Conference on Distributed Computing Systems*, May 1999.

[18] K.-L. Wu and P. S. Yu. Replication for Load Balancing and Hot-Spot Relief on Proxy Web Cache with Hash Routing. *Distributed and Parallel Databases*, 13(2), 2003.

[19] C. Yuan, Y. Chen, and Z. Zhang. Evaluation of Edge Caching/Offloading for Dynamic Content Delivery. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 16(11), November 2004.