Scalable Delivery of Dynamic Content using a Cooperative Edge Cache Grid *

Lakshmish Ramaswamy[◇], **Ling Liu**[†] and **Arun Iyengar**[♠]

[◇]Department of Computer Science
University of Georgia
Athens GA 30602
laks@cs.uga.edu

[†]College of Computing
Georgia Tech
Atlanta GA 30332
lingliu@cc.gatech.edu

[♠]IBM T. J. Watson Research Center
Yorktown Heights NY 10598
aruni@us.ibm.com

Abstract

In recent years edge computing has emerged as a popular mechanism to deliver dynamic web content to clients. However, many existing edge cache networks have not been able to harness the full potential of edge computing technology. In this paper, we argue and experimentally demonstrate that cooperation among the individual edge caches coupled with scalable server-driven document consistency mechanisms can significantly enhance the capabilities and performance of edge cache networks in delivering fresh dynamic content. However, designing large-scale cooperative edge cache networks presents many research challenges. Towards addressing these challenges, this paper presents cooperative edge cache grid (cooperative EC grid, for short) - a large-scale cooperative edge cache network for efficiently delivering highly dynamic web content with varying server update frequencies. The design of the cooperative EC grid focuses on the scalability and reliability of dynamic content delivery in addition to cache hit rates, and it incorporates several novel features. We introduce the concept of cache clouds as a generic framework of cooperation in large-scale edge cache networks. The architectural design of the cache clouds includes dynamic hashing-based document lookup and update protocols, which dynamically balance lookup and update loads among the caches in the cloud. We also present cooperative techniques for making the document lookup and update protocols resilient to the failures of individual caches. This paper reports series of simulation-based experiments to study the advantages of cooperation in edge cache networks, and the benefits and costs of the proposed architecture and techniques. The results show that the overheads of cooperation in the cooperative EC grid are very low, and our architecture and techniques enhance the performance of the cooperative edge networks.

1 Introduction

Recently, edge computing has emerged as a popular technique to efficiently deliver dynamic web content to the clients [1, 3, 4, 15, 25, 39]. Edge cache networks typically have several caches at various locations

*A shorter version of this paper appeared in the proceedings of the 25^{th} international conference on distributed computing systems (ICDCS-2005), Columbus, OH, June 2005 [27]

on the Internet, and the origin servers *offload* data and parts of the applications to these edge caches. However, many of the current edge cache networks have certain shortcomings that limit their ability to effectively deliver dynamic web content. Consider the problem of ensuring freshness of cached dynamic documents. Previous research has shown that due to the frequent changing nature of these documents, server-driven mechanisms, which can provide stronger consistency guarantees, are essential for maintaining their freshness [7, 35]. However, because of the high overheads of these techniques, most current day edge cache networks still rely upon the weaker *time-to-live* mechanism for maintaining consistency of cached documents. Furthermore, these systems cannot adequately handle sudden changes in request and update patterns of dynamic web content.

In this paper we argue that cooperation among the caches of an edge network can be a very effective tool in dealing with the challenges of caching dynamic web content. Previously researchers have studied cooperation in the context of caching static documents on client-side proxy caches [11, 13, 16, 22, 32]. A few commercial edge cache networks like Akamai [1] adopted similar cache cooperation models in their designs. However, in these systems cache cooperation is limited only to handling cache misses. Further, these systems only support the TTL-based weak document consistency mechanism.

1.1 Edge Cache Cooperation for Dynamic Content Delivery

Cooperation among edge caches is a very potent technique that can be used to enhance the capabilities of edge cache networks serving highly dynamic web content in multiple ways. First, when an edge cache receives a request for a document that is not available locally (i.e., the request is a local miss) it can try to retrieve the document from nearby caches rather than immediately contacting the remote server. Local misses occur either because of first-time requests for documents or more commonly because of requests for documents that were evicted due to the high costs of caching them. Irrespective of the causes of local misses, resolving them by retrieving documents from nearby cooperating caches can alleviate the loads on the origin servers and reduce the latency of local misses. Second, cooperation among the edge caches alleviates the load induced on the origin servers by server-driven document consistency mechanisms. In edge cache networks which incorporate cooperative consistency maintenance protocols, the servers can communicate the update message to only a few caches, which in turn distribute it to other edge caches. Third, cooperative edge cache networks can adopt cache management policies that are sensitive to the

cooperation among the edge caches, so that the available resources are optimally utilized. Fourth, when a cache fails, its load is collectively shared by the other caches in its vicinity. Furthermore, cooperative re-synchronization of caches that are recovering from failures ameliorates server loads.

Motivated by the benefits of edge cache cooperation we are developing techniques and system-level facilities for utilizing the power of low cost cache cooperation to efficiently and scalably deliver dynamic web content in large-scale edge cache networks. Towards this end, in this paper we present the design and evaluation of *cooperative edge cache grid (cooperative EC grid, for short)*, - a scalable, efficient, and failure resilient cooperative edge cache network for delivering highly dynamic web content with varying server-update frequencies. Concretely, this paper makes three technical contributions:

1. We experimentally study the costs and benefits of edge cache cooperation for delivering dynamic web content, demonstrating that edge cache cooperation can provide an order of magnitude improvement in the server loads, and with careful design it is possible to ensure that the overheads of cooperation are low.

2. This paper presents the architecture of cooperative EC grid, whose design incorporates various mechanisms for promoting cost effective cache cooperation and for dealing with the constantly evolving nature of dynamic web content. We also introduce the concept of cache clouds as a generic framework of cooperation in the EC grid.

3. We present our design of individual cache clouds, which includes novel, *dynamic hashing*-based cooperation protocols for efficient document retrievals and updates. These protocols also balance the document lookup and update loads dynamically among the caches of a cloud, thereby handling sudden changes in document access and update patterns. Further, we also discuss two strategies to make the protocols resilient to the failures of individual caches.

We present several simulation-based experiments to evaluate the proposed cooperative EC grid architecture and the associated techniques. The results indicate that cooperative EC grid supports effective and low-overhead cooperation among its caches.

The rest of the paper is organized as follows. In Section 2, we discuss the architecture of the cooperative EC grid. Section 3 describes the design of individual cache clouds and Sections 4 explains

our failure resilience mechanisms. Section 6 discusses the experimental evaluation of the proposed architecture and mechanisms. We discuss the related work in Section 7 and conclude in Section 8.

2 Cooperative Edge Cache Grid

This section provides an overview of the design architecture of the cooperative edge cache grid (cooperative EC grid) discussing the salient features of its design.

2.1 Design Challenges

Designing efficient cooperative edge networks poses several research challenges: First, an effective mechanism is needed to decide the scale and configuration of the edge cache network in terms of the number of caches, and the locations where these edge caches are placed. We refer to these challenges as *cache placement problems*. The challenge is to optimize the setup and maintenance, while ensuring that client latency is within desirable limits. Deciding which caches should cooperate with one another, or in other words, grouping caches into cooperative structures such that the efficiency and effectiveness of cooperation are simultaneously optimized is the second major challenge in constructing edge cache networks. Third, a dynamic and adaptive cooperation architecture and a set of highly efficient and failure resilient techniques are needed to deal with the constantly evolving nature of dynamic content delivery like continually changing document update and user request patterns. The fourth challenge is to design cache management (document placement and replacement) policies such that the various resources of the cache group are optimally utilized.

We are investigating various techniques to address these challenges [27, 28]. In this paper we focus on the architectural design of the cooperative EC grid discussing its novel features and mechanisms

2.2 Cooperative EC Grid: Design Architecture

A typical cooperative EC grid consists of several geographically distributed edge caches, and one or more origin servers. In this paper, we assume that the client requests are routed to a nearby cache that is likely to provide the best performance, by employing a technique like URL rewriting or application-layer request redirection [5, 17, 29]. The origin servers incorporate the infrastructure (web servers, application servers and backend databases) necessary for generating the dynamic content. Further, the origin sites also have document update engines, which continuously monitor the changes occurring

at the backend databases and initiate appropriate update messages. In this paper, our description of the cooperative EC grid assumes a proactive approach for maintaining document consistency, wherein the new version of the modified document is communicated to the caches (update-based model for consistency). However, all the techniques and mechanisms described in this paper work well even with an invalidation-based document consistency model. Further, for conceptual simplicity, our discussion in the rest of the paper assumes that the cooperative EC grid contains a single origin server, although all of the proposed techniques can be applied for scenarios with multiple origin servers.

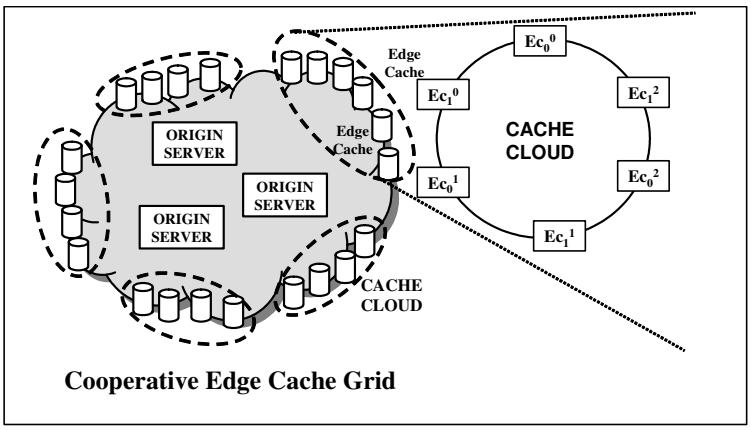

Figure 1: Architecture of Cooperative Edge Cache Grid

The design of the cooperative EC grid is based on the concept of *cache clouds*. A cache cloud is a group of edge caches that are located in close network proximity. The caches belonging to a cache cloud cooperate with one another for four basic purposes, namely, *document freshness maintenance*, *miss processing*, *resource management*, and *failure resilience and recovery handling*. The cooperative EC grid typically contains several cache clouds. The appropriate number of cache clouds and the size of each cloud depend on various factors including the scale of the EC grid and the document request and update patterns. Figure 1 illustrates the high-level design architecture of the cooperative edge cache grid with 22 caches organized into 5 cache clouds. The notation Ec_l^j represents an edge cache in the cooperative EC grid. The exact meaning of the notation will be explained in Section 3.

Previously, researchers have demonstrated the benefits of cooperative miss handling in the context of proxy caching of static documents [11, 13, 16, 22, 32]. In contrast, server-driven document consistency schemes are uniquely important to the performance and scalability of dynamic content delivery, and the cooperative EC grid provides significant benefits in this regard. In contrast to general edge cache networks wherein the origin server has to send update messages to each cache that holds a copy of the

document being modified, in the cooperative EC grid, the origin server is required to send only one update message per cache cloud, which is then cooperatively disseminated to all the caches within each cloud that hold a copy of the document. Further, the server is not required to maintain the state of any cache or cache cloud. Hence, the cooperative EC grid ameliorates the high overheads of server-driven consistency mechanisms.

The cache clouds serve as a generic framework of cooperation among closely located edge caches. This framework subsumes different cooperative architectures such as hierarchical and distributed caching architectures. In this paper we propose a flat cache cloud architecture in which the caches belonging to a cloud interact with one another as peers. Our architecture provides a scalable, efficient, and reliable infrastructure for delivering most common types of dynamic web content. However, any other architecture might also be employed, if required for specific circumstances.

3 Design of Cache Clouds

Since the cache clouds form the fundamental units of cooperation, the strategies adopted for their design would have a significant impact on the performance of the cooperative EC grid. We briefly analyze the important issues that need to be addressed while designing cache clouds. First, in order to collaboratively handle misses, caches should be able to locate the copies of requested documents (if any) existing within the cache cloud. We refer to the mechanism of locating document copies within a cache cloud as the *document lookup protocol*. Second, the cache cloud design should also incorporate a *document update* protocol through which the document update messages are communicated to the caches in the cloud that are currently containing the respective documents. In addition, we also need to design effective *cache management policies* and *failure handling mechanisms*.

Fundamentally, there are two approaches to cache cloud design, namely, centralized and distributed [27]. We have adopted the distributed approach, since it provides better scalability, load-balancing, and failure resilience properties. In our design each cache is responsible for handling the lookup and update operations for a set of documents assigned to it. In a cache cloud, if the cache Ec_l^j is responsible for the lookup and update operations of a cached document Dc, then we call the cache Ec_l^j as the *beacon point* of Dc. The beacon point of a document maintains the up-to-date lookup information, which includes a list of caches in the cloud that currently hold the document. A cache that

needs to serve a document Dc and the server that needs to update Dc utilize this lookup information as described later in the section. Thus, in our design each edge cache in the cloud plays dual roles. As a cache it stores documents and responds to client requests. As a beacon point it provides lookup and update support for a set of documents.

Now the problem is to decide which of the caches in the cloud should act as the beacon point of a given document. We refer to this as the beacon point assignment problem. In designing the cache cloud architecture, our goal is to assign beacon points to documents in such a manner that the following important properties are satisfied: (1) The caches within the cloud and the origin servers can efficiently discover the beacon point of any document. (2) The document lookup and the update operations are resilient to beacon point failures. (3) The load due to document lookups and updates is well distributed among all beacon points in the cache cloud. (4) The beacon point assignment and the load balancing scheme should be dynamic and adaptive to the variations of the lookup and the update patterns.

The simple solution of assigning documents to beacon points through a random hash function (henceforth referred to as *static hashing scheme*) neither provides resilience to beacon point failures, nor can it ensure good load balancing among the beacon points since the lookup and update loads often follow the highly skewed Zipf distribution. In this paper, we propose a dynamic hashing-based mechanism for assigning the beacon point of a document which supports very efficient lookup and update protocols, provides good load balancing properties, and can adapt to changing load patterns effectively.

3.1 Beacon Rings-based Dynamic Hashing Scheme

As the name suggests, in the dynamic hashing scheme, the assignment of documents to beacon points can vary over time so that the load balance is maintained even when the load patterns change. Consider a cache cloud with M edge caches. We organize the edge caches of a cache cloud into substructures called *beacon rings*. A cache cloud contains one or more beacon rings, and each beacon ring has two or more beacon points. Figure 2 shows a cache cloud with 4 beacon rings, where each beacon ring has 2 beacon points. The notation Ec_l^j denotes the cache that acts as the l^{th} beacon point in the j^{th} beacon ring. All the beacon points in a particular beacon ring are collectively responsible for maintaining the lookup information of a set of documents. In our scheme each document is uniquely mapped to a beacon ring through a random hash function. Suppose the cache cloud has K beacon rings numbered from 0 to

$K-1$. A document Dc is mapped to beacon ring j where $j = MD5(URL(Dc))$ Mod K. Here $MD5$ represents the MD-5 hash function, and URL(Dc) represents the unique identifier of the document Dc.

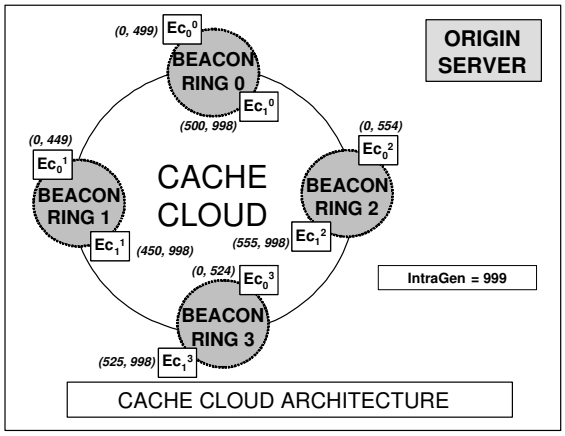

Figure 2: Architecture of Edge Cache Cloud

Suppose the document Dc is mapped to the beacon ring j containing L caches. One of these L caches would be assigned to serve as the *primary beacon point* for the document Dc. The primary-beacon point of a document maintains the up-to-date lookup information of the document, and is responsible for handling the lookup and updates of the document. All the other caches in the beacon ring serve as backup nodes for failure resilience purposes. Details of the failure resilience mechanism will be discussed in detail in Section 4.

The next question that needs to be addressed is: how is the primary beacon point assignment done? Or in other words for any document, say Dc, which is mapped to beacon ring j, how do we decide which of the L caches belonging to the beacon ring would act as its primary beacon-point? We propose a *dynamic hashing scheme*, which not only provides very good load balancing, but is also adaptive to changing load patterns.

Let us suppose that the L caches of the beacon ring j are represented as $\{Ec_0^j, Ec_1^j, \ldots, Ec_{L-1}^j\}$. The dynamic hashing technique uses an *intra-ring hash* function for distributing the documents to the L beacon points, which works as follows. An integer which is relatively large compared to the number of beacon points in the beacon ring is chosen and designated as the intra-ring hash generator (denoted as $IntraGen$). Each document's URL is hashed to an integer value between 0 and $(IntraGen - 1)$. This value is called the document's *intra-ring hash value* or ***IrH value*** for short. For example, for a document Dc, the IrH value would be $IrH(Dc) = MD5(URL(Dc))$ Mod $IntraGen$, where $MD5$ represents the MD-5 hash function, $URL(Dc)$ represents the URL of the document Dc and Mod represents the modulo function. Various beacon rings of a cache cloud may be configured to have different $IntraGen$ values. However, setting the same $IntraGen$ value for all beacon rings simplifies

cache cloud configuration and management.

Further, the range of intra-ring hash values $(0, IntraGen - 1)$ is divided into L consecutive non-overlapping sub-ranges represented as $\{(0, MaxIrH_0^j), (MinIrH_1^j, MaxIrH_1^j), \ldots, (MinIrH_{L-1}^j, IntraGen - 1)\}$. Each cache within the beacon ring is assigned to be responsible for one such sub-range, and this cache will be chosen as the primary beacon point for all the documents hashed into that sub-range. For example, the beacon point Ec_l^j is assigned to be responsible for the range $(MinIrH_l^j, MaxIrH_l^j)$. $MinIrH_l^j$ denotes the beginning and $MaxIrH_l^j$ denotes the end of the intra-ring hash values assigned to the cache Ec_l^j. Ec_l^j will serve as the primary beacon point of a document Dc, if $IrH(Dc)$ lies within the sub-range currently assigned to it. Figure 2 shows sub-ranges assigned to each beacon point.

3.2 Determining the Beacon Point Sub-Ranges

This section outlines the mechanism of dividing the intra-ring hash range into sub-ranges such that the load due to document lookups and updates is balanced among the beacon points of a beacon ring. This process is executed periodically (in cycles) within each beacon ring and it takes into account factors such as the beacon point capabilities and the current loads upon them. Any beacon point within the beacon ring may execute this process. This beacon point collects the following information from all other beacon points in the beacon ring.

Capability: Denoted by Cp_l^j, it represents the power of the machine hosting the cache Ec_l^j. Various parameters such as CPU capacity or network bandwidth may be used as measures of capability. These parameters can be obtained by the hardware/software specifications of the beacon points. For example, the maximum update/lookup throughput can provide a good estimate of the beacon point's capability. Various tools also exist to dynamically monitor these parameters. In this paper, we assume a more generic approach wherein each beacon point is assigned a positive real value to indicate its capability.

Current Sub-Range Assignment: Denoted by $(CMinIrH_l^j, CMaxIrH_l^j)$, it represents the sub-range assigned to the beacon point Ec_l^j in the current cycle.

Current Load Information: Represented by $CAvgLoad_l^j$, it indicates the cumulative lookup and update load averaged over the duration of the current period. The scheme can be made more accurate if

the beacon points also collect load information at the granularity of individual IrH values. Denoted by $CIrHLd_l^j(p)$, it indicates the load due to all documents whose IrH value is p. However, the $CIrHLd$ information is not mandatory for the scheme to work effectively.

The aim of the periodical sub-range determination process is to update the sub-ranges such that the load a beacon point is likely to encounter in the next cycle is proportional to its capability. For each beacon point, we verify whether the fraction of the total load on the beacon ring that it is currently supporting is commensurate with its capability. If the fraction of load currently being handled by a beacon point exceeds its share, then its sub-range shrinks for the next cycle, and vice-versa.

Specifically, the scheme proceeds as follows. First, we calculate the total load being experienced by the entire beacon ring (represented as $BRingLd^j$), and the sum of the capabilities of all the beacon points belonging to the ring (represented as $TotCp^j$). Then for each beacon point we calculate its appropriate share of the total load on the beacon ring as $AptLd_l^j = \frac{Cp_l^j}{TotCp^j} \times BRingLd^j$. For each beacon point, we compare its $CAvgLd$ with its $AptLd$. If $CAvgLd_l^j > AptLd_l^j$, then the scheme shrinks the sub-range of the beacon point for the next cycle by decreasing its $CMaxIrH_l^j$ by a value t such that $\sum_{p=CMaxIrH_l^j-t}^{CMaxIrH_l^j} CIrHLd_l^j(p) \approx (CAvgLd_l^j - AptLd_l^j)$. When the sub-range of a beacon point Ec_l^j shrinks, some of its load would be pushed to the beacon point Ec_{l+1}^j. The scheme takes into account this additional load on the beacon point Ec_{l+1}^j when deciding about its new sub-range.

If $CAvgLd_l^j < AptLd_l^j$ then the scheme expands the sub-range of the beacon point Ec_l^0 by increasing its $CMaxIrH$ value. The amount by which $CMaxIrH$ is increased is determined in a very similar manner as the shrinking case discussed above. After determining the sub-range assignments for the next cycle, all the caches in the cache cloud and the origin server are informed about the new sub-range assignments. Thus, the caches are aware of the current sub-range assignments of all other caches within its cache cloud. The sub-range determination scheme is described in more detail in [27]

Discussion

We now discuss a few important issues with respect to the dynamic hashing mechanism and the ways in which it can be extended for further performance improvements. Some beacon points in a beacon ring might find it costly to maintain the $CIrHLd$ information for each of the hash value within its sub-range. For such beacon points $CIrHLd$ for all hash values in the sub-range are approximated by

averaging $CAvgLd$ over the sub-range of IrH values. Thus, it is possible for a beacon ring to have a few beacon points that just maintain the total load information and others that maintain load information at the granularity of individual hash values. It is also possible that every beacon point of a ring maintains only the total load information. In either case, the quality of load balancing might be slightly below what would have been achievable if all beacon points were to maintain load information at the granularity of hash values, or the dynamic hashing mechanism might take a few load balancing cycles to converge at the best-possible range apportioning.

The dynamic hashing mechanism can be augmented with the *reorder* operation [14] to reduce the computational costs of load balancing, especially when beacon rings are large and the beacon points experience serious load imbalance conditions. However, the reorder operation does not alter the quality of load balancing [14]. Further, since beacon rings generally contain small numbers of caches, the dynamic sub-range determination operation suffices for most scenarios.

The dynamic hashing mechanism essentially balances the lookup and the update loads among the caches belonging to a beacon ring. Larger beacon rings are expected to yield better load balancing, since load balancing occurs across more machines. In fact, by creating a single beacon ring for each cache cloud, it is possible to balance loads across all the caches belonging to the cloud. However, as our experiments show, increasing the size of beacon rings beyond two caches per ring provides incremental improvements to load balancing. The load balancing process itself becomes complicated for large beacon rings because of the need for coordination among many beacon points. Considering the above pros and cons, we believe that it is better to configure the cooperative EC grid such that each beacon ring contains a few beacon points. In most scenarios this approach of balancing loads within each beacon ring is sufficient to yield very good load distribution across the entire cache cloud. Previous works in the area of local load balancing algorithms have also shown similar results, albeit in a different context [6]. In the rare scenario in which the load balancing obtained by this strategy is inadequate, the dynamic hashing scheme can also be extended to a bi-level scheme wherein load balancing is also done at the level of beacon rings. One possible approach is to map documents into a relatively large number of buckets and assign multiple buckets to each beacon ring depending upon its cumulative capability.

In the dynamic sub-range determination scheme, a single beacon point in a ring collects statistics

from all other beacon points and performs sub-range determination. This raises issues such as scalability of statistics collection and single point of failure in large beacon rings. Distributed, multi-level approaches can be designed to mitigate such problems. One approach would be to designate a few beacon points as *leaders*. Each leader collects statistics from a non-exclusive set of non-leader beacon points. The leader beacon points exchange the statistics they have collected so that each leader has data about all beacon points. Any one of the leaders can compute sub-ranges for the next cycle.

The appropriate duration for the sub-range determination cycle depends upon how dynamic the loads on the various caches of the cloud are; if the load pattern is highly dynamic, the cycle duration needs to be shorter and vice-versa. As our experiments show, for generic scenarios, a cycle duration of a few hundred seconds would be sufficient to obtain good load balancing characteristics.

The origin server and the caches within a cloud can determine the beacon point of any document by a simple and efficient two-step process. In the first step, the beacon ring of the document is determined by the random hash function. In the second step, out of the L beacon points within the j^{th} beacon ring, the beacon point of Dc is determined through the intra-ring hash function as we discussed before. The beacon point whose current sub-range contains $IrH(Dc)$ would be the beacon point of Dc.

Request and Update Processing in Cache Clouds

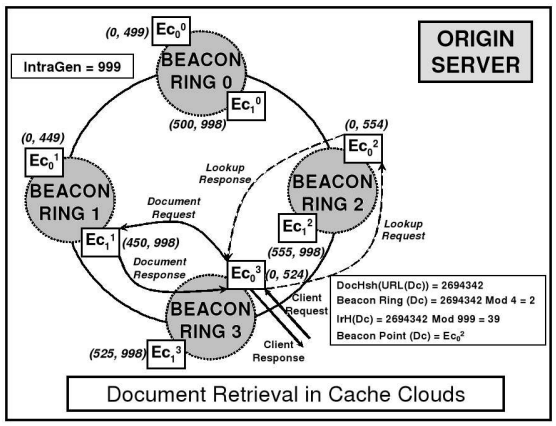

Figure 3: Client Request Processing in Cache Clouds

We now briefly outline the end-to-end scenario of processing client requests and document updates in cooperative EC grid. As mentioned earlier, all caches maintain the current sub-range assignments of all other caches in their cloud. Further, they also store the mapping between the cache IDs and the IP addresses and port numbers of other caches in their cloud. Suppose a client Cl requests a dynamic document Dc from the edge cache Ec_q^p. If the document is not available at Ec_q^p, it first determines Dc's beacon point (say Ec_l^j) through the two-step process described above, and sends a request to it. If the cache Ec_l^j has a copy of Dc, it is sent to Ec_q^p. Otherwise, Ec_l^j sends a list of caches (if any) within the cloud that are currently

storing copies of Dc, in which case Ec_q^p retrieves the copy from one of the current holders, or from the origin server if no other cache in the cloud contains Dc. Figure 3 illustrates the request handling process. Similarly, when the document Dc is modified, the origin server sends an update message to Dc's beacon points in various cache clouds, which in turn communicate it to the appropriate caches in their clouds.

4 Failure Resilience in Beacon Rings

In order to ensure that the document lookup and update operations are resilient to beacon point failures, each document's lookup and update information is replicated at all the caches in the document's beacon ring. We now explain two different strategies for replicating beacon information, namely an *eager replication scheme* and a *periodic replication scheme*. For clarity, we refer to the lookup and update information being maintained at a document's primary beacon point as the master-copy of the document's beacon information, and the copies present at other beacon points as backup-replicas.

Eager Replication of Beacon Information

The eager replication strategy takes a pro-active approach for replicating beacon information. In this strategy, the backup-replicas of the beacon information are always consistent with the master-copy. This means that any changes to the lookup and update information of a document (such as addition or deletion of copies) are immediately reflected at all the backup replicas.

When a beacon point fails, one or more of the other beacon points within the beacon ring assume the lookup and update responsibilities of all the documents that were being handled by the failed beacon point (please see Section 4.1 for further discussions on the scenario when multiple beacon points share lookup/update responsibilities of the failed beacon point). Note that backup-replica assumes the responsibilities of *all* documents that were being handled by the failed beacon point, which includes documents that might have been assigned to it due to failures of other beacon points. As all the replicas of the lookup information are consistent, the backup node has complete and up-to-date information to accurately handle the lookups and updates of all the documents that were originally assigned to the failed beacon point. When a failed beacon point is reactivated, it obtains the up-to-date lookup information from any of the other beacon points that are currently alive and then resumes its normal operation.

Periodic Replication of Beacon Information

In contrast to the eager replication scheme, in periodic replication there are brief durations of time in which the backup copies of the lookup information may be slightly out of sync when compared with the master lookup information available on the primary beacon point. The backup replicas of the lookup information are made consistent with the master copy at the end of predefined time periods called *beacon synchronization cycles*. Within a beacon synchronization cycle any updates to the lookup information of a document occurs only at the document's primary beacon point if it is alive, or at the backup node that is currently handling its lookup and update operations. The main observation that motivates the periodic replication is that the lookup information of a document is a *soft state* that can be recovered even when the primary beacon point fails and no other available replica is completely up-to-date.

Suppose the cache Ec_l^j is the primary beacon point of a document Dc. As in the eager replication case, if Ec_l^j fails, one or more of the backup replicas takes over the lookups and update operations of all the documents (including Dc) that were being handled by the failed cache. For simplicity, we discuss the scenario wherein a single backup replica, say Ec_{l+1}^j, takes over the lookup and update operations of all the documents currently being handled by Ec_l^j, including those documents that originally belonged to a cache that had failed earlier and are being currently handled by Ec_l^j. Better load balancing can be achieved if multiple backup replicas share the load of the failed beacon point (Section 4.1 presents one such simple mechanism).

Notice that the backup replica Ec_{l+1}^j may not have the most up-to-date lookup information of the documents which it takes over from Ec_l^j. Hence, the cache Ec_{l+1}^j marks the lookup information of these documents as *out-of-sync*. When Ec_{l+1}^j gets an update message for Dc whose lookup information is marked out-of-sync, the cache Ec_{l+1}^j sends an update message to *all* the caches in the cloud. However, this is a special update message, wherein the cache Ec_{l+1}^j notifies the caches that the lookup information regarding the document is out-of-sync and asks all the caches that hold the document to re-register with it. Thus, Ec_{l+1}^j recovers the up-to-date lookup information of the document Dc and hence removes the out-of-sync tag from Dc's lookup information.

When Ec_{l+1}^j receives a lookup request for Dc, there are two cases to consider. If the lookup information of Dc has been recovered through a previous update operation on Dc, Ec_{l+1}^j just forwards the

list of caches that currently hold a copy of the document. In case the lookup information of Dc is still marked as out-of-sync, Ec^j_{l+1} sends the lookup information it currently holds to the requesting cache, but it explicitly states that the lookup information may not be up-to-date. In this scenario, there is a small chance that the lookup information is slightly stale (the lookup information might indicate that Dc is available at a cache when it is actually not present and vice-versa). Our experiments show that the probability of a cache receiving stale lookup information is very small. The requesting cache then decides whether to contact one of the caches that is supposed to contain the document as per the lookup information it received or to contact the origin server directly.

When the primary beacon point of Dc (Ec^j_l) is reactivated, it obtains the current lookup information of all the documents that are assigned to it from Ec^j_{l+1} and resumes the lookup and update functionalities. In the very rare event of all the caches in a beacon ring failing simultaneously, the lookups and updates of the documents are handled by sending the appropriate messages to all the caches in the cloud. In both eager and periodic replication schemes, the backup replica that takes over the lookup and update responsibilities of a failed cache sends a message to all caches in the cache cloud. Similarly, when a beacon point recovers from its failure it notifies all other caches in the cloud.

4.1 Discussion

Failure Detection: The cooperative EC grid employs *heartbeat messages* to detect failures of caches. Within a beacon ring logically adjacent caches periodically send heartbeat messages to each other. The recipient of a heartbeat messages acknowledges it, thereby confirming that it is alive. A cache that does not respond to two consecutive heartbeat messages is assumed to have failed, and the sender initiates the beacon point failure handling mechanism. The sender also informs all other caches in the cloud about the failure, and the beacon points update their lookup information so that the failed cache is not included in the document lookup requests until its recovery. Failures might also be noted by non-adjacent caches or the origin server when they do not receive responses to their lookup/update or document retrieval requests, in which case they notify the adjacent beacon points, and the failure is confirmed by them through heartbeat messages.

Multiple Backup Replica Takeover for Better Load Balancing: Until now, we have assumed that a single backup replica would take over the update/lookup operations of all the documents that were being

handled by the failed cache. However, this might lead to a temporary load imbalance (until the next sub-range determination cycle). This problem can be mitigated by having multiple backup replicas share the responsibilities of the failed beacon point. One straightforward to achieve this would be to *split* the IrH value range assigned to the failed cache Ec_l^j among multiple backup replicas. The drawback of this approach is that each cache might be handling non-consecutive sets of IrH values, which complicates the lookup and update mechanisms. A second approach would be to split the IrH range assigned to the cache Ec_l^j between Ec_{l-1}^j and Ec_{l+1}^j. In case of significant load imbalance, any one of these two caches can start the sub-range determination process so that load balance is restored in the beacon ring. In case all but one of the beacon points of a ring have failed for very long durations of time, the single beacon surviving beacon point could be migrated to another beacon ring in order to maintain load balancing.

Comparing Eager and Periodic Replication Strategies: We now briefly discuss the relative pros and cons of the eager and the periodic replication schemes. There are two advantages to the eager replication scheme. First, a cache trying to retrieve a document always receives up-to-date lookup information. Second, there is no need to flood the network with lookup and update messages, except in the very rare event of all the beacon points belonging to a beacon ring failing simultaneously. However, the eager replication scheme suffers from the drawback that when the beacon information of a document is modified, all the beacon points in the document's beacon ring have to be informed. This message overhead could be heavy and may affect the cache clouds' performance. In contrast, the periodic replication scheme just requires a single message to be communicated when the document's lookup information is modified. Therefore, the message overheads are significantly lower. But, with periodic replication there is a small chance that a cache trying to retrieve a document might receive lookup information that is slightly stale. This, in the worst-case, might result in the cache contacting the origin site even though a copy of the document was available within the cache cloud. However, our experiments show that the probability of a cache receiving stale beacon information is very low. Therefore, we believe that periodic replication is a good choice, unless the rate of beacon point failure is very high.

5 Design Rationale

In this section we briefly explain the design rationale of the cooperative EC grid. For purposes of clarity, we compare our design to other possible alternatives such as hierarchical and distributed architectures

with ICP-based object lookup mechanisms and bloom filter-based summary cache protocols [13] which were used in many cooperative proxy-caching schemes, mainly for serving static web documents.

In the cooperative EC grid, the number of messages circulated for performing a document lookup is never greater than two, while both hierarchical and distributed architectures with ICP-based document lookups induce very high (up to $N - 1$) messages per document lookup, where N denotes the total number of caches in the edge cache network) message overheads. Strategies similar to summary cache protocol alleviate the lookup message overhead by maintaining approximate lookup information at each proxy. However, the summary cache may sometimes yield inaccurate lookup information, which can introduce serious inaccuracies in the document update process (discussed in the next paragraph). Further, the lookup message in the cooperative EC grid only has to traverse a single hop, which makes it very efficient. These assertions about the costs of document lookups are valid even when the respective primary beacon point has failed, as long as the beacon point failure has been previously detected and a backup replica has taken over the lookup and update responsibilities. In the rare event when a lookup request first detects the failure of the beacon point for the first time, the requesting cache has to contact one of the backup replicas, which would result in one additional message.

In cooperative EC grid the number of messages sent-out by the origin server per document update is never greater than the number of clouds in the grid, whereas the update load on the server of the ICP-based distributed architecture might be as high as the number of caches in the EC grid. While techniques such as bloom filters can reduce the message load, they cannot ensure that an update message would reach all the caches containing the document. Further, in our scheme the update messages have to traverse exactly 2 hops.

Three aspects of our document lookup/update mechanisms need closer examination. One concern is whether the two-step document retrieval procedure affects the latency experienced by the clients. We contend that effects of our lookup procedure on the client latency are minimal due to two main reasons. First, the caches belonging to a cloud are located *near* to one another, and thus the cost of intra-cloud communications are expected to be very low. Second, the amount of data exchanged during the lookup protocol is on the order of a few bytes. For such short communications, connection establishment costs dominate the data transfer costs. Since the number of caches in each cloud is limited, the caches can

maintain persistent connections to one another thereby eliminating the connection establishment costs for each lookup.

The second issue is the overheads of storing and maintaining the beacon information at various caches. The beacon information of each document just consists of the document ID and a list of identifiers of caches in the cloud that currently contain the document. Thus, the beacon information stored at each cache is very small when compared to the amount of data cached for serving client requests. In order to ensure consistency of the beacon information, a cache that either stores a new copy or removes an existing one needs to notify the document's beacon point, which would cost one message consisting of a few bytes of data. The communication can be further optimized by purging documents periodically and sending collective notification messages. Pure ICP-based systems do not maintain any group-wide information about the documents available in the individual caches and hence, there are no maintenance costs. By contrast, in directory-based schemes like summary cache, changes to directory information have to be reflected in all other cooperating caches costing $N - 1$ messages. Summary cache ameliorates this load by accumulating the changes and updating the directory copies periodically. Our periodic replication-based failure resilience scheme requires that the backup replicas of the beacon information be made consistent with the master copy at the end of each beacon synchronization cycle thereby inducing a message load of L messages per beacon synchronization cycle, where L represents the number of beacon points in the ring. Since beacon rings are typically very small, the message overheads of the failure resilience protocol are minimal.

The third issue is the low geographical scalability of the URL hashing mechanism used in the dynamic hashing scheme. We believe that this does not pose a serious problem to the performance of the cooperative EC grid since the caches among which URLs are hashed belong to a single cache cloud and hence would be in close network proximity.

Thus, we conclude that the cooperative EC grid architecture and our cache cloud design are well suited for efficiently delivering dynamic web content with varying server-update frequencies.

6 Experiments and Results

We have performed a range of experiments to evaluate the cooperative EC grid architecture and the various associated algorithms. Our experimental study has three main goals: **(1)** Studying the benefits and

costs of edge cache cooperation for dynamic content delivery; **(2)** Evaluating the architectural design of the cooperative EC grid; and **(3)** Studying the performance of the two replication schemes for providing failure resilience.

6.1 Experimental Setup

The experimental study was conducted through trace-based simulations of edge cache networks. The simulator can be configured to simulate different caching architectures such as edge cache network with no cooperation, cooperative EC grid, hierarchical architecture, and distributed architecture. Further, it can also simulate various techniques including static hashing with no failure resilience, dynamic hashing with eager replication, and dynamic hashing with periodic replication. The underlying network was simulated using GT-ITM network topology generator according to the hierarchical transit-stub model [40]. Our decision to use GT-ITM was motivated by the fact that GT-ITM is one of the most widely used topology generators for simulating various large-scale distributed systems [9, 18]. Although, there is an on-going debate on the accuracies of various network topology generators, it is commonly believed that the topologies generated by GT-ITM are fairly realistic [21]. We also note that the configuration-settings that we have used for generating the topologies have been adopted by several previous research projects [9].

Each cache in the edge cache network receives requests continuously from a request trace file. In cooperative EC grid the request is processed according to the protocol described in Section 3.1. If the document is available within the cache receiving the request, it is recorded as a local hit. If the document is retrieved from another cache in the cloud, it is a cloud hit, and the request is a miss if the document is retrieved from the origin server. In non-cooperative edge cache networks, the cache suffering a local miss directly contacts the origin server to retrieve the document. The server reads continuously from an update-trace file and utilizes the document update protocol provided by the architecture to communicate the updated version of the document to the appropriate set of caches.

We use two types of datasets for our experiments. Most of our experiments are based upon a dataset which is derived from real request and update traces from a major IBM sporting event website[1](henceforth referred to as *Sydney*). The original request logs contained client requests received at

[1]The Sydney 2000 Olympic Games Website

various geographically distributed caches, and the update logs contained updates generated at the origin server. The logs contained 52527 unique documents with the mean and median of document sizes being 65.3 Kilobytes and 48.3 kilobytes respectively. The access pattern of documents resembles the Zipf distribution with Zipf parameter $\alpha = 0.72$. The access pattern of documents also demonstrates moderately high temporal correlation. The mean and median duration between two consecutive accesses to the same document are 151.6 and 154.7 minutes respectively. The correlation between the document access and update patterns was 0.43. 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. Most of our experiments are based upon real access and update rates. In order to study the effects of update and access rates on the proposed system, we also use a few artificially chosen document update and access rates in a small subset of experiments. The second dataset, called *Zipf-0.9 dataset*, is synthetic and contains $100,000$ unique documents. The mean and median size of documents are 60.0 kilobytes and 43 kilobytes respectively. In this dataset, both document accesses and updates follow the Zipf distribution (power-law distribution), with the Zipf parameter value set to 0.9. The mean and median duration between two consecutive accesses to the same document are 138.2 and 188.3 minutes respectively. The correlation between the access and update patterns of this workload was 0.57.

The main costs of caching dynamic documents are the loads imposed by the server-driven document freshness maintenance schemes on the edge cache network. These costs increase as the number of documents stored at each cache increases. In our simulator, we can control the number of documents stored in each cache through a parameter called *cache limit*. This parameter specifies an upper limit on the number of documents stored at a cache. The cache limit of an edge cache Ec_l^j is defined as the percentage of the total number of unique documents in the trace that the cache Ec_l^j is permitted to store. In other words, if the cache limit of Ec_l^j is set to Cl and if $TotalDocs$ represents the total number of unique documents in the trace, Ec_l^j cannot store more than $\lfloor \frac{Cl*TotalDocs}{100} \rfloor$ documents. The caches in our simulator support two cache replacement policies. The first is the well-known *least recently used (LRU)* policy. The second policy, which we call *access-update sensitive policy (AU)* policy, evicts the document with the highest $\frac{\text{update frequency}}{\text{client access frequency}}$ ratio from the cache. The AU policy aims to optimize the

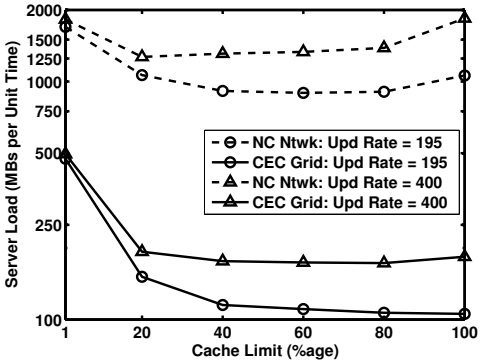

Figure 4: Effects of Cooperation on Server Load (Varying Cache Limit)

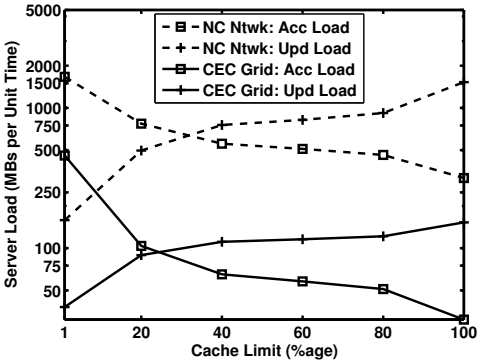

Figure 5: Document Access and Update Loads on Server (varying cache limit)

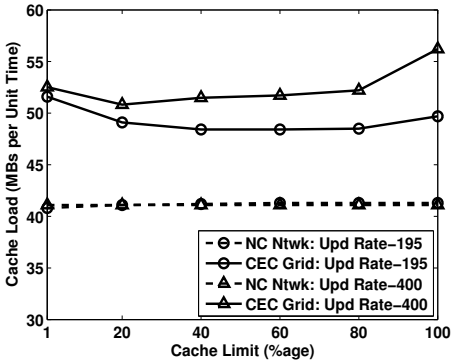

Figure 6: Effects of Cooperation on Cache Load (Varying Cache Limit)

cumulative load on the server due to document consistency maintenance and client accesses by evicting documents that have high update rates but low access rates (thereby minimizing the consistency load on the server), and retaining documents that have relatively high access rates but low update frequencies (thus minimizing the load due to client accesses). In the experiments reported in this paper all the caches employ the AU replacement policy. The caches are in a cold state at the beginning of the simulation and are warmed up by client accesses. When the variations of the vital performance parameters over several successive time units have become minimal, we conclude that the system has reached its steady state. All results were collected several time intervals after the system reaches its steady state. In this section the term *unit time* corresponds to a single minute.

6.2 Costs and Benefits of Cooperation

In the first set of experiments we study the costs and benefits of cache cooperation for edge cache networks which serve highly dynamic web content by comparing the performance of the cooperative EC grid scheme to the edge cache networks wherein the caches interact only with the server and do not cooperate with one another. The edge cache networks which do not support cooperation are henceforth called *non-cooperative edge cache networks (non-cooperative EC network)*. For brevity, in the performance graphs, the cooperative EC grid is represented as *CEC Grid*, and the non-cooperative EC network is denoted as *NC Ntwk*.

The first experiment considers a cooperative EC grid and a non-cooperative EC network with identical configurations in terms of the numbers of caches, their relative positions within the wide area network, and the request and update traces that drive them. Both the cooperative EC grid and the non

cooperative EC network contain 120 caches. For the cooperative EC grid, the caches are organized into 12 cache clouds. Each cloud is comprised of 5 beacon rings with each beacon ring containing 2 caches. We vary the cache limit of the caches and study its impact on the performance of the cooperative EC grid and the non-cooperative EC network. The graph in Figure 4 compares the server loads of the two schemes when the cache limit varies from 1% to 100%. The server load is measured in terms of the number of bytes of data sent out by the server per unit time. The performance is evaluated at two document update rates, namely 195 updates per unit time, which is the real update rate of the trace, and 400 updates per unit time. The term update rate is defined as follows: Suppose *update count* of a document Dc represents the number of times Dc changed during the course of the simulation. Update rate is defined as the ratio of the sum of the update counts of all the documents in the trace to the amount of time elapsed during the simulation. However, the number of updates that reach a particular cache in unit time depends upon the number and the set of documents stored at the cache. For example, when the cache limit was set to 50% the average number of document updates per unit that were seen at the caches was about 104.6.

As the graph indicates, the server loads of the cooperative EC grid are an order of magnitude less than the corresponding values for the non-cooperative EC network at both update rates. For example, when the update rate is set to 195 and cache limit is set to 50%, server loads of the cooperative EC grid and the non-cooperative EC network are 115 and 915 MBs respectively. Thus, irrespective of the numbers of documents stored in the cache, cooperation among caches reduces the load on the origin server. The valley shaped curves of the server loads can be explained as follows: Low cache limit results in high miss rates thereby placing high document access loads on the server. On the other hand, when cache limit is very high the server incurs high document consistency maintenance costs, which again increases its load.

To gain a better understanding of this phenomenon, in Figure 5, we plot the loads incurred at the origin server due to document accesses and document updates in both cooperative EC grid and non-cooperative EC network, as the percentages of documents cached varies from 1% to 100%, when the document update rate is set to 400 updates per unit time. As the cache limit increases, the access loads on the servers of both schemes drop, whereas their document update loads increase. For the non-

cooperative EC network the load due to document updates overtakes the access load when the cache limit is around 35%, whereas for the cooperative EC grid, the update load becomes the dominating factor when the cache limit is around 25%.

One of the concerns about promoting cooperation in edge cache networks serving dynamic content is whether cache cooperation causes severe increases in the loads on the individual caches. In order to answer this question, in Figure 6 we measure the average number of bytes sent out per unit time (henceforth referred to as *outgoing byte load*) by the caches of the cooperative EC grid and the non-cooperative EC network when the percentage of cached documents varies from 1% to 100%. In non cooperative EC network, the outgoing byte load of a cache is comprised of the documents that are dispatched from the cache in response to client requests. In contrast, the outgoing byte load in the cooperative EC grid is the aggregate of mainly three kinds of loads, namely load due to client requests, load due to document requests from other caches in the cloud, and load incurred by conveying document updates to caches in the cloud (as a beacon point). The outgoing byte loads on the caches in the cooperative EC grid are about 16% and 23% higher than the loads of the caches in the non-cooperative EC network, when the document update rate was set to 195 and 400 respectively. Thus, the overheads of cooperation on individual caches are within reasonable bounds. The interaction between the request load and the update load again comes into play and causes the valley shaped curve of the cache loads.

In the second experiment we study the effects of document update rates on the benefits and costs of cooperation in edge cache networks. The configuration of the cooperative EC grid and the non-cooperative EC network are the same as in the previous experiment. Figure 7 shows the total server load in terms of the bytes dispatched per unit time for both cooperative EC grid and the non-cooperative EC network when the document update rate varies from 100 to 500. The vertical line at 195 indicates the actual update rate of the trace. The cache limit is set to two values, namely 25% and 50%. Note that in this graph, the Y-axis is on log-scale. The server loads for the non-cooperative EC network is 7 times to 8 times higher than the corresponding values of the cooperative EC grid. Figure 8 shows the average loads (in terms of bytes dispatched per unit time) on the caches of the cooperative EC grid and the non-cooperative EC network as the update rate varies from 100 to 500 updates per unit time, when the cache limit of each cache is set to 25% and 50%. As the results demonstrate, the increase in average

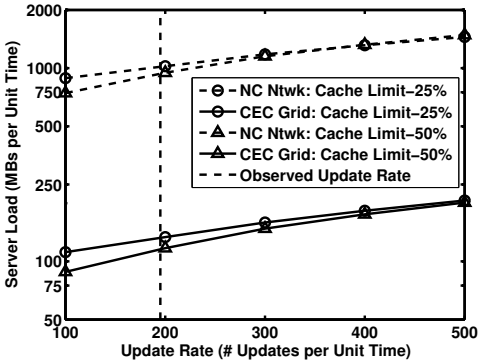

Figure 7: Effects of Cooperation on Server Load (Varying Update Rate)

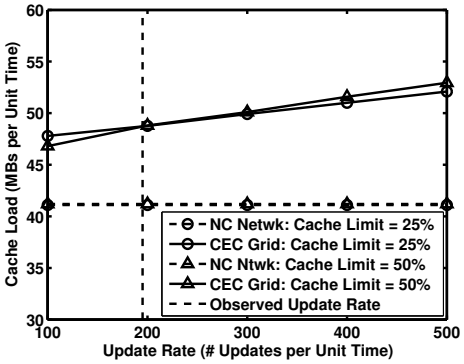

Figure 8: Effects of Cooperation on Cache Load (Varying Update Rate)

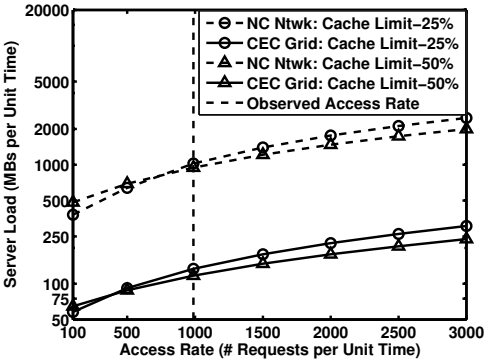

Figure 9: Effects of Cooperation on Server Load (Varying Access Rate)

cache loads as a result of cooperation is very moderate (16% to 25%).

The third experiment studies the impact of document access rates on cooperative EC grid and non cooperative edge cache network. The system configurations are the same as in previous experiments. In Figure 9, we plot the numbers of bytes dispatched per unit time by the servers of cooperative EC grid and non-cooperative edge cache network as the access rates at caches varies from 100 document accesses per unit time to 3000 accesses per unit time, when the cache limit is set to 25% and 50%. The Y-axis of this graph is again on log-scale. The vertical line indicates the actual access rate of the trace. The results reaffirm the observation of the previous experiment that the server load of the non-cooperative EC network is about an order of magnitude higher than that of the cooperative EC grid. We have also measured the loads on individual caches at various access rates. The average numbers of bytes dispatched per unit time by caches of the cooperative EC grid is only slightly higher than those of the non-cooperative edge cache network. For example, when cache limit is set to 25% and access rates of caches are 2500 requests per unit time, the average cache load of the cooperative EC grid is 14% higher than that of the non-cooperative edge cache network.

In the fourth experiment (Figure 10), we compare the scalability properties of the cooperative EC grid and the non-cooperative EC network with respect to the number of caches in the network. We evaluate the cumulative loads on the origin servers of the cooperative EC grid and non cooperative edge cache network when numbers of caches in the system vary from 20 to 200. We assume that the number of clients scales linearly with the number of caches in the system. Two different configurations for the cooperative EC grid are considered; in the first configuration (represented as CEC Grid: Config - 1) the number of clouds in the EC grid is set to $\lfloor \frac{N}{10} \rfloor$, and in the second (represented as CEC Grid: Config

- 2) the number of clouds is set to $\lfloor \frac{N}{20} \rfloor$, where N represents the total number of caches in the edge cache network. We use the real document update rates (195 updates per unit time) for this experiment. The cache limit is set to 25%. The Y-axis of the graph denotes the server loads on the log-scale. The cumulative load on the origin server of the non cooperative EC network grows almost at the same rate as the number of caches, whereas the server loads of the cooperative EC grid increase at a much lower rate. It was also observed that the consistency loads form a large fraction of the cumulative server load. This experiment demonstrates the scalability benefits of effective cache cooperation.

In the final experiment of this set, we estimate the average latencies of the cooperative EC grid and the non-cooperative edge cache network. The response time of a request is in general composed of two types of components, namely the time consumed at various caches and servers to process the request (which we call *processing component*) and the time taken to transfer the document among servers, caches and clients (*communication component*). The communication components are determined by the GT-ITM network simulator package [40]. We make a few simplifying assumptions for computing the processing component. Most of these assumptions are based on prior research results [10, 30] In case a cache contains the document that is being requested (by clients or other caches), the time required for the cache to respond lies within the range $(100, 200)$ milliseconds and it varies linearly with the document size. In case the request reaches the origin server, we distinguish two scenarios. If the request is for static data, the server response time is in the range $(100, 200)$ and varies linearly with the size of the document. Previous studies have shown that the time required for generating dynamic documents is in general orders of magnitude higher than serving static documents [10]. Accordingly, if the request reaching the server is for dynamic content, we assume that the response time at the server is uniformly distributed in the range $(1000, 2000)$ milliseconds. However, note that the server response of 1000 millisecond does not imply that the throughput of the server is 1 per second. Servers handle multiple requests simultaneously, and hence exhibit higher throughputs. Figure 12 compares the average response times of the cooperative EC grid and non cooperative edge cache network when the total numbers of caches in the system is 120 and 500. In each case, the number of clouds is set to $\lfloor \frac{N}{10} \rfloor$. We measure latency values when the cache limits are set to 25% and 50%. The average response latency of the 120 cache non cooperative EC grid are 25% and 22% higher than the corresponding values for the

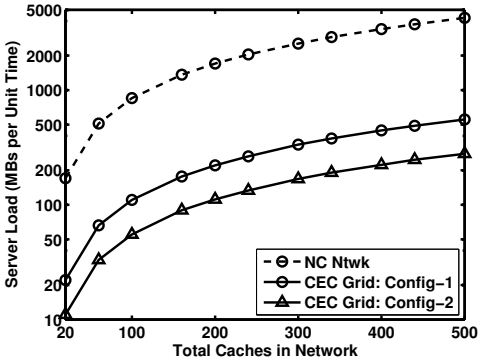

Figure 10: Scalability benefits of edge cache cooperation

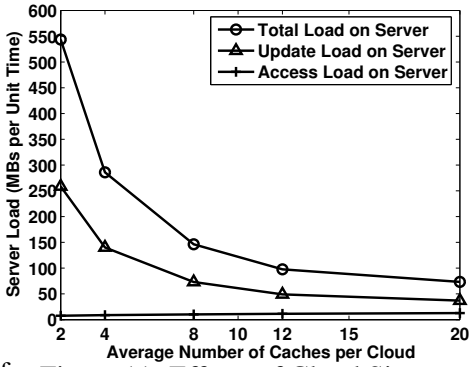

Figure 11: Effects of Cloud Size on Server Load

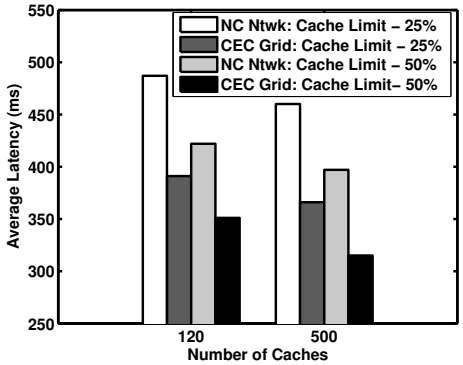

Figure 12: Effects of Cooperation on Latency

cooperative EC grid when the cache limit is set to 25% and 50% respectively. At a cache limit of 50%, both edge cache networks yield significantly lower average latencies because of higher local hit rates. Further, the document lookup phase of processing a miss in the cooperative EC grid constitutes less than 4% of the cumulative average latency of processing a miss, thus demonstrating that our two-phase document retrieval process does not adversely impact the latency of the cooperative EC grid. The GT-ITM network simulator does not support persistent connections. The latency may be further reduced by using persistent connections for communications among caches and the server.

6.3 Evaluating the Design of Cache Clouds

Our next set of experiments study the performance of the techniques that we have proposed for designing individual cache clouds. We first evaluate the load balancing properties of the beacon ring-based dynamic hashing scheme on a cooperative EC grid containing 120 caches. For most experiments in this set, the cooperative EC grid was configured to have 12 clouds each having 10 caches. All the caches are assumed to be of equal capabilities, which implies that perfect load balancing is achieved when all of the beacon points encounter same amount of load. Further, the intra-ring hash generators ($IntraGen$s) are set to 999 for all beacon rings. Since the dynamic hashing scheme essentially works at the level of individual clouds, we report the results on a representative cloud of the cooperative EC grid. We use the coefficient of variation of the loads on the beacon points to quantify load balancing. Coefficient of variation is defined as the ratio of the standard deviation of the load distribution to the mean load. The lower the coefficient of variation, the better is the load balancing.

We compare the load balancing accomplished by the static and the dynamic hashing schemes in a cache cloud with 10 caches. The caches are organized into five beacon rings, with each beacon ring

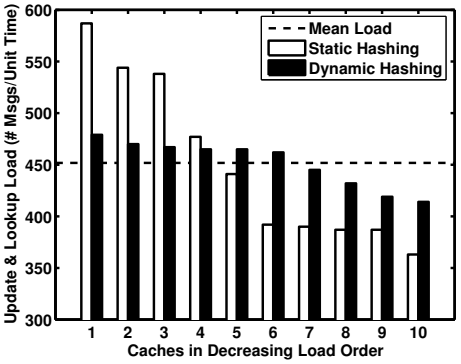

Figure 13: Lookup and Update Load Distribution among Caches (Sydney Data Set)

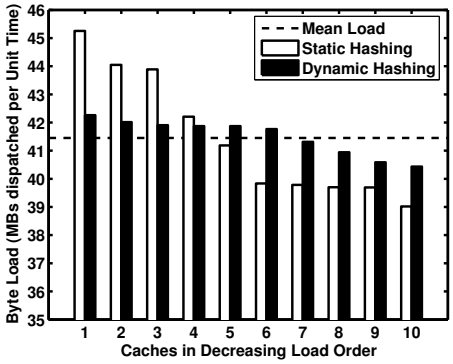

Figure 14: Distribution of Cumulative Load in Cache Cloud (Sydney Data Set)

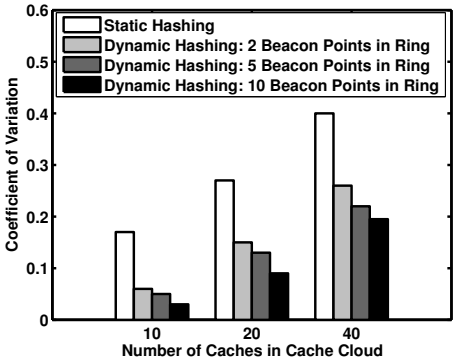

Figure 15: Impact of Beacon Ring Size on Load Balancing

containing 2 beacon points. The cache limit is set to 50%, and the document update rate is 400 updates per unit time. The cycle length for the sub-range determination process was set to 250 time units for all beacon rings. The bar graph in Figure 13 shows the load distribution among the beacon points for the static and the dynamic hashing schemes on the Sydney dataset. On the X-axis are the beacon points in decreasing order of their loads, and on the Y-axis are the loads in terms of the number of updates and lookups being handled by the beacon points per unit time. With static hashing, the load on the most heavily loaded beacon point is 1.35 times the mean load of the cache cloud. In the dynamic hashing scheme this ratio decreases to 1.06, resulting in a 20% improvement over the static hashing scheme. The dynamic hashing scheme also reduces the coefficient of variation by 61%. For the Zipf-0.9 dataset dynamic hashing improves the ratio of the heaviest load to the mean load by 37% and coefficient of variation by 63%. The load distribution graph of the Zipf-0.9 dataset is contained in [27].

We now study the effects of the dynamic hashing mechanism on the cumulative loads of the caches due to the two distinct roles they play in the cooperative EC grid, namely responding to client requests and handling the lookups and updates of a set of documents. Figure 14 shows the distribution of the cumulative loads among the caches for the static and the dynamic hashing schemes for the Sydney dataset. The cumulative cache loads are measured in terms of the total numbers of bytes dispatched by the caches per unit time. As the graph shows, the distribution of the cumulative loads among the caches of the dynamic hashing scheme is better balanced than the caches in the static hashing scheme. The results for the Zipf-0.9 dataset indicate similar patterns, albeit on a larger scale.

In the next experiment we study the effect of the size of the beacon rings on load balancing. We evaluate the dynamic hashing scheme on cache clouds consisting of 10, 20 and 40 caches. For each

cache cloud we consider three configurations in which each beacon ring contains 2, 5, and 10 beacon points. Figure 15 indicates the results of the experiment on the Sydney dataset. The dynamic hashing scheme with 2 beacon points per ring provides significantly better load balancing in comparison to static hashing. When the size of the beacon rings is further increased we observe an incremental improvement in the load balancing achieved by the dynamic hashing scheme. The reason for the observed behavior is explained in Section 3.1.

The third experiment (Figure 16) of this set studies the impact of the dataset characteristics on the static and the dynamic hashing schemes. For this experiment we consider several datasets all of which follow the Zipf distribution with Zipf parameters ranging from 0.0 to 0.99, and measure the coefficients of variations at various Zipf values. The skewness of the load increases with increasing value of the Zipf parameter. As the skewness in the load increases the coefficient of variation values also increase for both schemes. However, the increase is more rapid for the static hashing scheme. At a Zipf parameter value of 0.9, the coefficient of variation for the static hashing scheme is 0.65, whereas it is 0.44 for the dynamic hashing scheme.

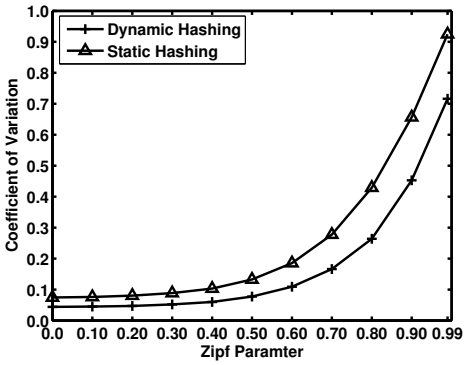

Figure 16: Impact of Zipf parameter on load balancing

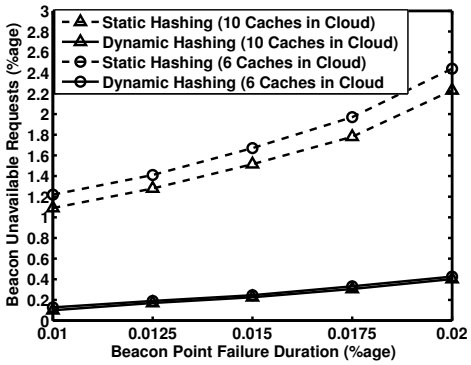

Figure 17: Improvement in Beacon Information Availability

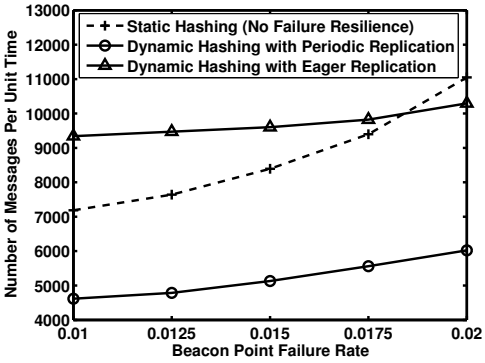

Figure 18: Network Load with Various Failure Resilience Schemes (10 Caches Per Cloud)

6.4 Evaluating Failure Resilience Mechanisms

In this set of experiments we evaluate the performance of the two approaches for providing resilience to beacon point failures, namely the eager and the periodic replication mechanisms. The experiments are performed on a cooperative EC grid of 120 caches. We report the results on representative clouds of the cooperative EC grid.

The individual beacon points of the cache cloud can fail at arbitrary points of time. The failure

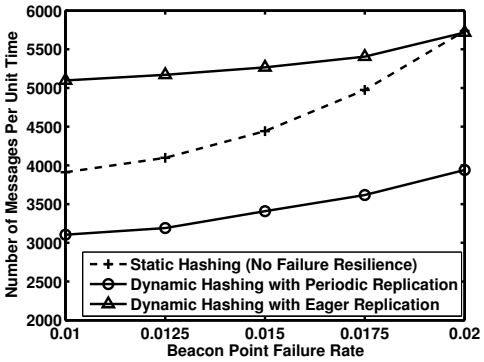

Figure 19: Network Load with Various Failure Resilience Schemes (6 Caches Per Cloud)

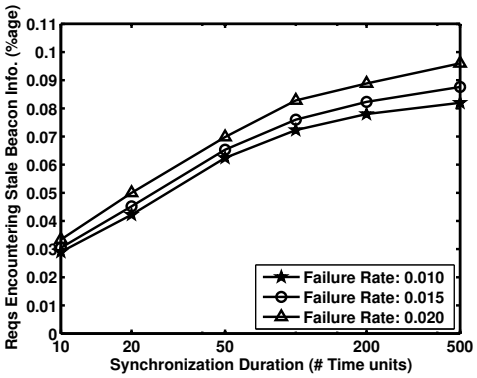

Figure 20: Staleness of Beacon Information in Periodic Replication (10 Caches in Cloud)

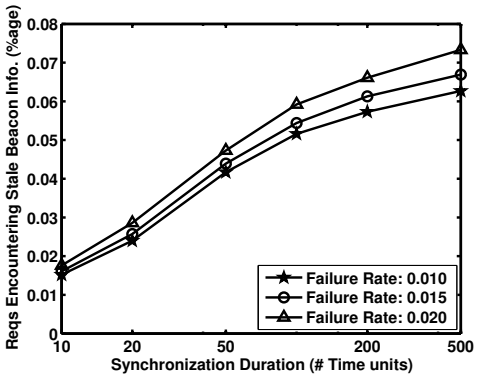

Figure 21: Staleness of Beacon Information in Periodic Replication (6 Caches in Cloud)

pattern of an arbitrary beacon point is modeled as a *Weibull* process with increasing failure rates. Weibull processes are commonly used in reliability engineering to model failure patterns of systems and system components. The Weibull process is characterized by the probability distribution function $f(t) = \kappa\lambda^{\kappa}t^{\kappa-1}e^{-(\lambda t)^{\kappa}}$, where λ is called the scale parameter and κ is called the shape parameter. In the Weibull function, if κ is set to values greater than 1.00, then we obtain a class of distributions that exhibit increasing failure rates (IFR). In IFR distributions, the probability of a component failing during an arbitrary time period (T_0+T_d) given that the component has not failed until the time instant T_0, grows with increasing values of T_0. In the current context, IFR lifetime distributions imply that a beacon point that has not failed in the recent past has a higher probability of failure than a beacon point that has failed and has been restored recently. Our assumption about IFR of caches is based upon previous research results which indicate that most system failures are due to software crashes, and software crashes generally exhibit IFR distributions typically because of data corruption and error accumulation over time [8]. In our experiments the beacon points are active for long stretches of time, whereas the durations of failures are very short. At the end of the failure duration the beacon point begins the recovery process. We use two metrics to quantify the performance of the scheme, namely, beacon information availability and the induced message load. All the simulations were run for 5000 time units. We experimented with κ being set to 2.0, 3.0, 4.0 and 5.0. The results of the various trials showed similar trends. We report the results when κ was set 2.0. λ was set to appropriate values to obtain various failure rates reported in the experiments.

Figure 17 compares the dynamic hashing scheme (with failure resilience support through eager or

periodic replication) and the static hashing scheme (with no failure resilience support) with respect to their beacon information availabilities. The X-axis of the graph indicates the failure rates of the individual beacon points, and the Y-axis shows the percentage of lookup and update requests for which the beacon information was not available. The graph indicates that the dynamic hashing scheme (with failure resilience support) provides an order of magnitude improvement in beacon information availability over the static hashing scheme. A total of 378 beacon point failures were recorded, which included multiple failures of individual beacon points. The maximum number of beacon points in the cloud that were down at the same time was 3, and the maximum number of beacon points of the same beacon ring that were down at the same time was 2, although both of these were rare and lasted for very short durations of time. The observed results can be explained as follows: In static hashing, when a beacon point fails, the lookups and updates of all the documents mapped to it encounter unavailable beacon information, whereas in the dynamic hashing scheme a lookup or an update request for a document encounters the unavailable beacon information condition only when all the beacon points belonging to the document's beacon ring have failed.

Next, we study the performance of the various failure resilience schemes with respect to the message loads induced by them within the cache cloud. In Figure 18 and Figure 19 we plot the number of lookup/update messages circulated per unit time with each of the three schemes in cache clouds consisting of 10 and 6 caches respectively. In each case the clouds are configured such that the each beacon ring contains 2 beacon points. The X-axes of the graphs indicate the failure rate of beacon points, and the Y-axes show the numbers of messages circulated in unit time. The synchronization cycle durations of all beacon rings are set to 250 time units, which implies that the backup replica of each document's lookup information is made consistent with its master copy once in every 250 time units. We have experimented with other synchronization cycle durations to obtain similar results.

A few interesting observations emerge from these two graphs. First, as we mentioned in Section 4, eager replication scheme is costly in terms of the message cost. In fact, when the failure rates of the beacon points are low, the no failure resilience scheme is better than the eager replication scheme. However, as the failure rate increases, the number of messages in the no failure resilience scheme grows at a fast rate, and it overtakes the message cost of the eager replication scheme when the failure rate

is around 0.018. This phenomenon is due to the fact that as the failure rate increases, the frequency of cloud-wide flooding in the no resilience scheme rises sharply. The periodic replication scheme performs better than both the no failure resilience approach and the eager replication scheme at all values of failure rates.

Although the periodic replication scheme provides significant performance benefits, it has the problem that a cache trying to retrieve a document might receive stale lookup information. In the next two experiments, we show that the probability of caches receiving stale lookup information is very low, and hence this problem has little impact on the overall performance of the cache cloud. In Figure 20 and Figure 21, we plot the percentage of lookup requests receiving stale beacon information at various values of the synchronization cycle duration for a cache cloud with 10 caches and 6 caches respectively. The duration of the synchronization cycle is likely to have considerable impact on the percentage of lookups receiving stale beacon information because it determines how frequently the backup copies of the lookup information are made consistent with the master copy.

We observe that the percentages of lookups receiving stale information are very small at both cache cloud sizes. For a cache cloud with 10 caches, when the failure rate is 0.010, the percentage of stale lookups ranges between 0.03% and 0.08% as the duration of the synchronization duration varies from 10 time units to 500 time units. Hence, we see that even when the duration of the synchronization cycle is significantly long, the percentage of stale lookups is very low. It should also be noted that a stale lookup does not affect the correctness of the document retrieval protocol (please see Section 4). Considering the low message overhead and the very small probability of stale lookups, we conclude that periodic replication scheme is a good choice for providing resilience to beacon point failures.

7 Related Work

Edge computing has received considerable attention from the research community in recent years. Further, it has been adopted by companies such as Akamai [1], and by products like IBM's Websphere Edge Server [3] and Oracle's 10g Application server [4]. ACDN [25] is an edge computing platform that supports automatic redeployment of applications as necessitated by changing request patterns. Yuan et al. [39] present a comparative study of 4 different offloading strategies, showing that a simple strategy of offloading the functionality of composing web pages from fragments can be very effective in

terms of latency and server load reduction. We note that our design of the cooperative EC grid supports this offloading strategy. Gao et al. [15] propose an edge caching scheme that alleviates consistency maintenance costs by according different levels of consistency to different objects depending upon the application semantics.

The above edge caching schemes regard individual edge caches as completely independent entities, and they do not provide support for cooperation among their edge caches. Akamai's edge cache networks have incorporated limited cooperation among its caches [1], wherein the cooperation is restricted to serving cache misses. Moreover, these edge cache networks only provide Time-to-Live-based (TTL) weaker consistency guarantees. In contrast, the cooperative EC grid utilizes cache cooperation in multiple ways to enhance dynamic content delivery, and it provides stronger document consistency guarantees through low-cost, server-driven protocols.

Previously, cooperation among caches has been studied in the context of client-side proxy caches [11, 13, 22, 32]. Internet Cache Protocol (ICP) [16] was designed specifically for communication among the web caches. Researchers have also developed various alternatives for optimizing communication costs and other performance parameters, prominent among which are directory-based schemes [13, 32], cache routing schemes [33, 34] and multicast-based schemes [22]. In directory-based schemes, each cache maintains possibly inexact and incomplete listings of the contents of all other cooperating caches, where as multicast-based schemes utilize IP level multicasting for inter-cache communication. Routing schemes hash each document to a cache in the cooperative group which is responsible for retrieving and storing the document. Our dynamic hashing scheme can be viewed as a combination of directory and hashing techniques, wherein documents are mapped to beacon points via the dynamic hashing mechanism and each beacon point maintains up-to-date lookup information on the documents that are mapped to it. These schemes are not very effective for dynamic content delivery since they do not provide support for stronger consistency mechanisms and do not consider the costs of document updates in their design.

Ninan et al. [23] describe *cooperative leases* – a lease-based mechanism for maintaining document consistency among a set of caches. Shah et al. [31] present a dynamic data disseminating system among cooperative repositories in which a dissemination tree is constructed for each data item based on the

coherency requirements of the repositories. The server circulates the updates to the data item through this tree. The focus of both these works is on the problem of consistency maintenance of documents among a set of caches. In contrast, our cooperative EC grid architecture systematically addresses various aspects of cooperation such as collaborative miss handling, cooperative consistency management, efficient and failure-resilient document lookups and updates, and cost-sensitive document placements. Researchers have proposed schemes such as adaptive push-pull and different variants of the generic *lease*-based framework for minimizing the overheads of the basic server-driven consistency mechanism for cached dynamic content [7, 20, 35, 36, 37, 38]. Low cost cache cooperation techniques that we have proposed can further improve the efficiency and scalability of these consistency maintenance schemes.

Consistent hashing [19] is related to the dynamic hashing scheme proposed in this paper. While consistent hashing can provide reasonable load balancing properties, it cannot easily adapt to sudden changes in lookup and update patterns. In contrast, adaptability to changing update and lookup patterns is an important feature of the dynamic hashing scheme. Additionally, there is a considerable body of literature addressing various problems in the general area of dynamic content delivery [2, 12, 24, 26].

8 Conclusions

Although caching on the edge of the Internet has been a popular technique to deliver dynamic web content, most edge caching systems do not provide adequate support for cooperation among the individual edge caches. This paper demonstrates that cache cooperation can provide significant benefits to edge cache networks. We present *cooperative EC grid* - a large-scale edge cache network specifically designed to support low-cost cooperation among its caches. The architecture of cooperative EC grid is based on the concept of cache clouds, which are group of edge caches that cooperate for the purposes of maintaining document freshness, serving client-requests, managing cumulative resources, and providing failure resilience. Our design of cache clouds incorporates several novel strategies for improving the efficiency and effectiveness of cooperation, such as beacon ring-based dynamic hashing schemes for documents lookups and updates, and two schemes for replicating lookup information in order to provide resilience to beacon point failures. This paper reports series of experiments showing that the proposed techniques are very effective in enhancing the performance of edge cache networks.

Acknowledgements
This work is partially sponsored by NSF CSR, NSF CyberTrust, NSF ITR, an IBM Faculty Award, an IBM SUR grant, an Air Force AFOST grant, and a grant from the UGA research foundation. We also would like to thank the reviewers and the subject editor for their helpful comments.

References

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

[2] Edge Side Includes - Standard Specification. http://www.esi.org.

[3] IBM WebSphere Edge Server. http://www-3.ibm.com/software/webservers/edgeserver/.

[4] Oracle Corporation: Application Server 10g Release 2 White Paper. http://www.oracle.com/technology/products/ias/pdf/1012_nf_paper.pdf.

[5] A. Acharya, A. Shaikh, R. Tewari, and D. Verma. MPLS-based Rrquest Routing. In *Proceedings of the International Workshop on Web Caching and Content Distribution (WCW)*, 2001.

[6] A. Anagnostopoulos, A. Kirsch, and E. Upfal. Stability and Efficiency of a Random Local Load Balancing Protocol. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 2003.

[7] M. Bhide, P. Deolasse, A. Katker, A. Panchbudhe, K. Ramamritham, and P. Shenoy. Adaptive Push-Pull: Disseminating Dynamic Web Data. *IEEE Transactions on Computers*, 51(6), June 2002.

[8] V. Castelli, R. E. Harper, P.Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, and W. P. Zeggert. Proactive Management of Software Aging. *IBM Journal of Research and Development*, 45(2), 2001.

[9] M. Castro, P. Druschel, A.-M. Kermarec, and A. Rowstron. Scribe: A Large Scale and Decentralized Application-level Multicast Infrastructure. *IEEE Journal on Seleceted Areas in Communications*, 20(8), 2002.

[10] 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.

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

[12] 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*, 2002.

[13] 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*, 1998.

[14] P. Ganesan, M. Bawa, and H. Garcia-Molina. Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems. In *Proccedings of International Conference on Very Large Databases (VLDB)*, 2004.

[15] L. Gao, M. Dahlin, A. Nayate, J. Zheng, and A. Iyengar. Improving Availability and Performance with Application-Specific Data Replication. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 17(1), January 2005.

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

[17] A. Iyengar, E. Nahum, A. Shaikh, and R. Tewari. Web Caching, Consistency, and Content Distribution. In M. P. Singh, editor, *The Practical Handbook of Internet Computing*. Chapmann and Hall/CRC Press, 2005.

[18] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of the* 4^{th} *Symposium on Operating System Design and Implementation (OSDI)*, 2000.

[19] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi. Web Caching with Consistent Hashing. In *Proceedings of the* 8^{th} *International World Wide Web Conference*, 1999.

[20] W.-S. Li, W.-P. Hsiung, D. V. Kalshnikov, R. Sion, O. Po, D. Agrawal, and K. S. Candan. Issues and Evaluations of Caching Solutions for Web Application Acceleration. In *Proceedings of the* 28^{th} *International Conference on Very Large Data Bases (VLDB-2002)*, 2002.

[21] N. Malouch, Z. Liu, D. Rubenstein, and S. Sahu. A Graph Theoretic Approach to Bounding Delay in Proxy-Assisted, End-System Multicast. In *Proceedings of the Tenth IEEE International Workshop on Quality of Service (IWQoS)*, 2002.

[22] 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.

[23] 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.

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

[25] 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*, 2003.

[26] L. Ramaswamy, A. Iyengar, L. Liu, and F. Douglis. Automatic Detection of Fragments in Dynamic Web Pages and its Impact on Caching. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 17(6), June 2005.

[27] 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)*, 2005.

[28] L. Ramaswamy, L. Liu, and J. Zhang. Efficient Formation of Edge Cache Groups for Dynamic Content Delivery. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, 2006.

[29] S. Rangarajan, S. Mukherjee, and P. Rodriguez. User Specific Request Redirection in a Content Delivery Network. In *Proceedings of the International Workshop on Web Content Caching and Distribution (IWCW)*, 2003.

[30] M.-C. Rosu and D. Rosu. Kernel Support for Faster Web Proxies. In *Proceedings of the USENIX Annual Technical Conference*, 2003.

[31] S. Shah, K. Ramamritham, and P. Shenoy. Resilient and Coherence Preserving Dissemination of Dynamic Data Using Cooperating Peers. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(7), July 2004.

[32] 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*, 1999.

[33] D. Thaler and C. Ravihsankar. Using Name-Based Mappings to Increase Hit Rates. *IEEE/ACM Transactions on Networking*, 6(1), February 1998.

[34] V. Valloppillil and K. W. Ross. Cache Array Routing Protocol v1.0., 1997. Internet Draft.

[35] J. Yin, L. Alvisi, M. Dahlin, and A. Iyengar. Engineering Web Cache Consistency. *ACM Transactions on Internet Technology*, 2(3), August 2002.

[36] J. Yin, L. Alvisi, M. Dahlin, and C. Lin. Volume Leases for Consistency in Large-Scale Systems. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 11(4), June 1999.

[37] H. Yu, L. Breslau, and S. Shenker. A Scalable Web Cache Consistency Architecture. In *Proceedings of the ACM SIGCOMM '99 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 1999.

[38] H. Yu and A. Vahdat. Design and Evaluation of a Conit-based Continuous Consistency Model for Replicated Services. *ACM Transactions on Computer Systems*, 20(3), August 2002.

[39] 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.

[40] E. W. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proceedings of IEEE-INFOCOM*, 1996.