A New Consistency Algorithm for Dynamic Documents in Content Distribution Networks [1]

Zongming Fei [a]

[a] *Laboratory for Advanced Networking*
Department of Computer Science, University of Kentucky
773 Anderson Hall, Lexington, KY 40506, U.S.A.

Abstract

One of the important problems in Content distribution networks (CDNs) is how to maintain the consistency of content at replicas with the origin server, especially for those documents changing dynamically. In this paper, we propose a new hybrid consistency algorithm that will generate less traffic than the traditional propagation approach, *and* the invalidation approach. The basic scheme is extended to the case in which requests are not evenly distributed over all replicas. We propose a hierarchical framework which allows replicas at different levels to make a decision based on the statistics collected. Extensive simulations are performed to examine how the traffic generated and the freshness ratio at replicas are affected by various parameters. We experiment with a wide range of request rates, update frequencies, and numbers of replicas. The results show that our approach can take advantage of content distribution networks and significantly reduce the traffic generated.

Key words: Web Caching, Data Replication, Propagation, Invalidation

1 Introduction

Over the past few years we have witnessed an unprecedented growth in the number of Internet users. Perhaps even more impressive is the rate at which new "content" is being added to the Web each day. Moreover, there doesn't appear to be any end in sight to the addition of new web content. Making information available to a rapidly growing user population is quickly becoming a challenging problem, which has been attacked from both the client side and the content provider side.

Email address: `fei@cs.uky.edu` (Zongming Fei).

[1] This work was supported in part by the National Science Foundation under Grant CCR-0204304.

- On the client side, the *caching* technique has been employed to store documents accessed by clients and serve the future accesses to the same documents without having to retrieve them from the origin server. A web cache can be maintained at each client or can be shared by a group of clients via a caching proxy. It is also possible to organize caches hierarchically [1].
- On the content provider side, the *replication* technique has been used to set up a cluster of servers and the load of client requests is distributed. More recently, the *Content Distribution Network* (CDN) [2] is used to push the content from the origin server to geographically distributed replicas, which bring content to the edge of the network where clients are attached.

Our focus is on the CDNs and how content at the origin server is delivered to replicas. Two common approaches to this problem are to deliver the data over (1) N unicast channels or (2) over an (application-layer tunneled) multicast tree that connects the replicas [3,4]. Clearly the unicast approach wastes network bandwidth and can cause congestion at bottleneck links, while application-level multicast approaches are more efficient in delivery (although not as efficient as native IP multicast [5]). Currently more deployed CDNs use unicast over multicast, but there are increased interests in designing (application-layer) multicast for the content delivery. We believe the multicast approach will be finally adopted because of its efficiency. In this paper, our discussion will be based on the assumption that there is some multicast delivery network established between the origin server and replicas, either native multicast or application-layer multicast.

The efficiency of delivery schemes depends on the characteristics of documents. Roughly speaking, documents requested by clients can be divided into three categories.

- Static documents. These documents usually do not change at the origin server, or if they do change, do so at a very large time scale.
- Dynamic documents. These are documents that change at a relatively high rate. They can generate large amounts of update traffic if not kept in check.
- Dynamically generated documents. These documents are generated on-the-fly by the server, or replica, based on the information provided by clients.

For dynamically generated documents, the server typically needs to run some "cgi" code to produce the document and thus cannot be implemented with the caching approach. However in CDNs, replicas, like the origin server, can be instantiated with the appropriate code to generate these documents dynamically. The information used to generate the documents may change frequently or infrequently like other standard web documents and thus this information is dealt with just like dynamic or static web content.

The update of static documents is not a problem because it is, in most cases, one time effort and thus does not generate too much traffic. This is not the case for updating dynamic documents. It involves generating a much larger amount of traffic. It is the goal of this paper to design an efficient scheme for updating dynamic documents and maintaining consistency of documents at replicas with the origin server. There are two ways to update dynamic documents:

- Propagation. Whenever an object is changed at the origin server, it is propagated to all replicas.
- Invalidation. Whenever an object is changed, an invalidation message is sent to all replicas. Each replica will fetch the new version of the object individually at a later time, if necessary.

Unfortunately, the propagation approach often generates a significant amount of unnecessary traffic if documents are updated more frequently than they are requested. On the other hand, the invalidation approach may fail to make full use of the delivery network and can degrade to unicast delivery from the origin server to each individual replica. More importantly, the response time will also suffer because the object has to be fetched from the origin server after the request arrives at the replica. This is not true of the propagation approach, where an object is always fresh at replicas.

In this paper, we propose a novel hybrid approach that will generate less traffic than the propagation approach, *and* the invalidation approach. The origin server makes the decision of using either propagation or invalidation method for each document, based on the statistics about the update frequency at the origin server and the request rates collected by replicas. We develop a technique that can reduce the burden of request rate collection at replicas and get rid of the implosion problem when replicas send the statistics to the origin server. The basic scheme is extended to the case in which requests are not evenly distributed over all replicas. We propose a hierarchical framework which allows replicas at different levels to make a decision based on the statistics collected. We illustrate the relationship between decisions made by a replica and its children. Extensive simulations are performed to examine how the traffic generated and the freshness ratio at replicas are affected by various parameters. We experiment with a wide range of request rates, update frequencies, and numbers of replicas. The results show that our approach can take advantage of content distribution networks and significantly reduce the traffic generated, compared with the propagation approach and the invalidation approach.

The remainder of this paper is organized as follows. Related work is discussed in Section 2. Section 3 provides an overview of the architectural model and assumptions. Section 4 proposes a new approach for maintaining consistency of dynamic documents at replicas with the origin server. Section 5 extends the basic scheme to the case where requests to replicas are not evenly distributed. In Section 6 we present extensive simulations to evaluate the performance of various delivery schemes. We conclude the paper in Section 7.

2 Related Work

Efficient delivery of content to clients has attracted much attention over a decade. The interests in it are also exemplified by the commercial development of delivery networks [6–8]. The design and techniques used in these systems are mostly proprietary.

Early research on content delivery concentrates on using native multicast to push the content to clients directly [9,10]. These schemes are most efficient when the content changes at a high frequency [11]. It is not an easy task to determine whether a document should be multicast or not. The lack of native multicast support leads people to design alternative ways for delivery. Prominent among them are application layer multicast schemes such as Yoid [4], the end system multicast [3], and Scattercast [12]. Basically they run an auto-configuration protocol to establish a tunneling delivery structure among participating members. These unicast connections are used for delivery of content from the origin server to participating nodes. Yoid sets up tunnels directly among members, and the end system multicast and Scattercast build a mesh topology first and then run the spanning tree algorithm to establish a delivery tree. While the self-organization topology setup mechanism complements our work, these schemes do not discuss how to update dynamic content efficiently.

Caching is an area closely related to content delivery. Harvest [1] is an early proposal trying to establish a hierarchical caching structure, in which the bandwidth can be more efficiently used and popular web pages can be pushed closer to the clients. Other hierarchical schemes are adaptive web caching [13], access driven cache [14]. The problem with caching schemes is how to maintain the freshness of the content stored in the cache. There are several proposals dealing with this problem. A simple way to guarantee the freshness is that the cache always checks with the origin server. Using *if-modified-since* (IMS), a cache can get the reply from the origin server that the page is fresh, or get a fresh page itself. There are some heuristic schemes to determine the freshness by observing the life time of a page [15]. The longer a page has not been modified, the longer it will not be modified in the future. However, these adaptive schemes cannot guarantee the freshness of a page. The invalidation scheme was proposed to deal with the problem [16]. Unfortunately, requiring a server to contact a huge number of caches is simply not feasible from a scalability standpoint. In CDNs, the invalidation can be used because knowledge about replicas is available and the delivery network can be used to forward invalidations.

Invalidation can be used most efficiently when combined with propagation and other approaches in CDNs. SPREAD architecture [17] analyzed the tradeoff among client validation (by using IMS), invalidation, and propagation (called replication in that paper). It analyzed the effects of different update distributions on the determination of the thresholds that decide which method to use. The discussion was based on the translucent proxying architecture that achieves transparency to clients by intercepting SYN of a TCP connection and using IP tunneling. The paper used a trace-driven simulation to show the benefits of the threshold-based approach with unicast validation, invalidation and propagation in terms of consumed bandwidth and client latency. There was a brief discussion about multicast extension to the architecture, but it did not discuss the problem associated with uneven distribution of client requests. MMO protocol proposed by Li and Cheriton [18] made use of multicast for delivery and invalidation of updates. Whenever an object is modified at the origin server, an invalidation is sent to all replicas by multicast. The new version is immediately delivered to a multicast group, through which all subscribed replicas can get the updated version of the object. A decision is made at each replica whether to join the multicast group or not, based on a threshold W. If there is no request during previous W invalidation periods, the replica does not join; otherwise, it does. The paper showed the performance gain of the protocol without details about how to determine the threshold and how it is related to the inter-update time of an object.

The focus of this paper is on the impact of multicast delivery on the decision making at the origin server and other replica servers whether to use propagation or invalidation. We design a very simple decision rule to select the best strategy on per object basis. Equipped with our efficient scheme for access rate collection, our approach makes propagation and invalidation perform in a way to generate maximal benefit from each of them, and achieves the goal of reducing the traffic for consistency management.

3 Architectural Overview

In this section we briefly outline the architectural model and assumptions. The Internet can be described as a set of interconnected, autonomous domains (see Figure 1). A domain may have zero or more content providers (called *origin servers*), which can be fully or partially replicated in other domains. An origin server can specify whether it is replicable or not, and similarly, a domain may have policies about whether it allows replicas in the domain or not.

In the example in Figure 1, domain D_1 contains an origin server S, which has seven replicas ($R_2, R_3, R_4, R_5, R_8, R_9, R_{11}$) located in different domains. We assume that a delivery infrastructure (or *delivery network*) has been established from the origin server to the replicas. For example, an application layer multicast [3] protocol may have set up unicast tunnels from S to R_2, from R_2 to R_3, R_4, R_5, from R_4 to R_8, R_9, and from R_5 to R_{11}. This delivery network can be used to propagate information from the origin server S to all replicas. Information collected by replicas can be combined along the way when they are sent back to the origin server through this

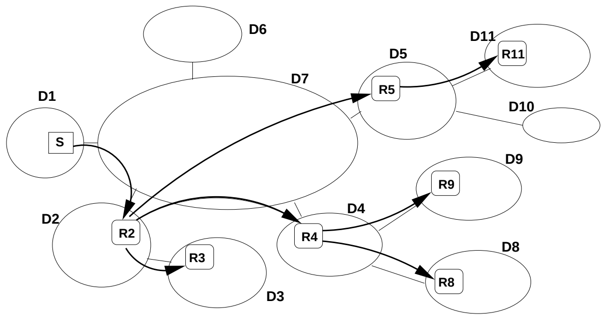

Fig. 1. Architecture for Content Delivery

delivery network.

We view the establishment of replicas as an incremental process. When a new replica is set up, we need to decide how to fill up its content (all or partial). Later when a change is made to a document at the origin server, we need to decide how the update will be made aware to all replicas. Our goal is to guarantee that any object served by any replica to clients is always fresh.

We assume all client requests will be directed to some replica. Which specific replica will be chosen for a request by a client is the server selection problem [19]. Different schemes for server selection may result in different response time of clients and the traffic generated from clients to replicas. Because server selection is not the focus of this paper, we ignore the traffic generated from clients to replicas. The traffic we are interested in and used as one of performance measures is the traffic generated for delivering content from the origin server to replicas. In the following sections, we describe two existing consistency management schemes.

3.1 Propagation

Propagation is an approach often used in CDNs. Whenever an object is changed at the origin server, it is propagated to all replicas through the delivery network by multicast. This can guarantee that an object is always up-to-date at any replica when it is requested from a client. We assume that the propagation will not take too long, so we ignore the temporal state in which a request for a document is processed at a replica when the document has been changed at the origin server and is being propagated towards replicas. In these cases a client will get a copy of document different from the one at the origin server, but we still consider it fresh. Under this assumption, we know that the documents served by replicas are always the newest version.

For a newly established replica, all documents at the origin server will be delivered to it by unicast. This will guarantee that each replica has an up-to-date version of all objects at the origin server when it is set up.

3.2 Invalidation

Another approach is invalidation, in which an invalidation message is delivered to all replicas through the delivery network by multicast when an object is changed at the origin server. Each replica will mark the object invalid upon the reception of this invalidation message. When a request for the object is processed at the replica at a later time, the replica will send a request to the origin server, which will use unicast to deliver the object to this replica. We also ignore the temporal state in which an invalidation is being delivered from the origin server to replicas and consider the copy served by the replicas during the time fresh. Though a document in a replica may be invalid for a certain period of time, the copy served by the replica to clients is always the newest version.

In the invalidation approach, a newly established replica can simply assume that all objects are invalid. The replica will be filled up as it gets more requests from clients and tries to fetch a fresh copy from the origin server.

4 A New Approach for Maintaining Consistency

We propose a hybrid approach based on the observations of problems existing in the propagation and invalidation approaches. With propagation, a document is propagated to replicas even if it is not accessed at all. This may generate extra traffic and we find that if a document is not accessed frequently enough, it should not be propagated.

In the invalidation approach, a document may be delivered to replicas through unicast because it is invalidated by the origin server and each replica fetches the document individually. Thus the traffic is more than it could have been if the multicast delivery were used. The observation is that if a document is accessed frequently enough, we should use multicast to deliver it to replicas, rather than let each replica fetch it by unicast.

In addition to the access frequency (or *request rate*), another factor is the *update rate* of documents at the origin server. It determines how frequently a document is propagated or invalidated and thus, the traffic for propagation or unicast fetching. Actually it is the relative values of the request rate and the update rate that determine which method we should use. It is simple in the caching scheme because we can use propagation if the request rate is greater than the update rate, otherwise use invalidation. We cannot generalize this to CDNs by simply comparing the rate of requests and updates, because we have to take into account that the methods of transmitting documents from the origin server to replicas are different, propagation using multicast and invalidation using unicast. Therefore, we need to factor in the relative efficiency of unicast and multicast.

Relations between unicast and multicast transmissions were first studied by Chuang and Sirbu [20] and recently validated by empirical studies [21,22]. Assume that L_m is the total number of multicast links in the distribution tree and L_n is the total number of unicast hops for distribution from the source to all replicas. Then we have

$$\frac{L_m}{L_n} = 1 - \delta \tag{1}$$

where δ represents the percentage gain in multicast efficiency over unicast. We can have an estimate of δ in terms of the number of receivers, N and an associated *efficiency factor*, ε. That is, $\delta = 1 - N^{\varepsilon}$. The value of ε tends to range between -0.34 and -0.30 [22]. In our simulations we set $\varepsilon = -0.3$.

We consider a specific object, assuming its update rate at the origin server is U and the total request rate is R. As discussed above, we focus on the traffic generated for transmission of the document from the origin server to replicas. Assume that the document size is s and the time we measure the traffic is t. If we use propagation, the traffic generated is $T_p = U * L_m * s * t$.

In the invalidation approach, we ignore the traffic generated for sending invalidation messages for now (because they are relatively small) and focus on the unicast fetch by replicas. The request rate by each replica is $\frac{R}{N}$ and the average number of hops from the origin server to each replica is $\frac{L_n}{N}$. Therefore, the traffic generated by invalidation is $T_i = (\frac{R}{N} * \frac{L_n}{N}) * s * t * N = R * L_n * s * t/N$.

Our strategy is based on the relative value of T_p and T_i. Use propagation if $T_p \leq T_i$; use invalidation if $T_p > T_i$. With the power law relation (1) between L_m and L_n, we can get our final decision rule:

For each object,
  if $U * N^{1+\varepsilon} \leq R$, use propagation;
  otherwise use invalidation.

There are four parameters. U is the update rate, R is the total request rate, N is the number of replicas, and ε is the efficiency factor ($-0.34 \leq \varepsilon \leq -0.30$).

The number of replicas is available at the origin server. The update rate for each object can be calculated based on the history. The main problem is to get the total request rate at the origin server, while all requests are made to replicas. A simple solution is that each replica collects statistics about the request rate and periodically pushes the data to the origin server. The implosion problem may occur if the number of replicas is large. Our solution is to make use of the delivery network existing between replicas and the origin server. Rather than letting replicas push the statistics directly to the origin server, we let the origin server periodically send poll messages to those replicas directly connected in the multicast distribution tree. When a replica receives a poll message, it sends a poll message to replicas directly connected to it downstream, if any. If a replica does not have any replica connected to it downstream, it will send a reply with request rate information to its upstream replica (or the origin server). Those replicas having sent out poll messages will calculate the aggregate request rate for each object after it receives all replies to its poll messages. Then these aggregate rates are sent further upstream. The origin server will get the total request rate information after it receives replies from all replicas directly connected to it. This request rate collection process takes place in the time frame of hours. So the traffic generated is not significant compared to the volume of traffic generated for shipping objects from the origin server to replicas. The collection process is similar to the concast in the context of active networking [23,24], but we can deploy the algorithm in our distribution network without requiring the network being active.

One way to reduce the size of request rate information is that we can ignore those objects requested less than or equal to a threshold value. For example, if we set the threshold value to 1, we can ignore the objects requested once or not requested at all. Actually the number of these objects may be large and the size of reply messages can be made much smaller. It is up to the origin server whether to make adjustment to the final result, if such optimization takes place at replicas.

One remaining problem is how to deal with the newly established replica. Our solution is based on the statistics currently available at the origin server. Those documents satisfying $U * N^{1+\varepsilon} \leq R$ will be delivered to the replica by unicast, and all other documents will be marked invalid at the replica.

5 Dealing with Uneven Distribution of Requests

The basic scheme assumes that requests are evenly distributed over all replicas [25]. Therefore the decision is only made at the origin server based on the frequency of updates to an object and the total number of requests to it. However, in practical situations, it is possible that requests concentrate on some replicas, while other replicas get no or a very small number of requests. In this case, the total number of requests is not large enough for the origin server to decide to propagate all the updates to replicas. As a matter of fact, it is a right decision from the point of view of the origin server because propagation will generate a large amount of traffic when the overall request rate over all replicas is not very high. A better solution is to propagate the updates only to those replicas with a lot of hits and to send an invalidation message to those replicas with few or no requests. Therefore we extend the basic scheme to allow replicas, in addition to the origin server, to make decision whether the updates should be propagated or invalidation messages should be sent.

In order for a replica to make a decision based on the strategy proposed in the previous section, it has to know three measures, i.e., the request rate for each object to all downstream replicas, the update rate for each object and the number of downstream replicas. Recall that the request rate to an object is collected in a bottom-up fashion and each replica can record the number of requests to an object during the collection process. The number of downstream replicas is relatively static information. It can be collected once and updated infrequently, e.g., once per day. This information can be piggybacked in one of the request rate collection message sent from leaves up to the origin server and recorded by each replica along the path. The update rate of each object can be estimated by the origin server and periodically multicast to all replicas or piggybacked in the propagation of the new version of an object or the invalidation message for the object. Each replica should be able to get the information necessary to make the best decision for the replicas in the subtree rooted at this replica.

Once a replica (assume it is R_{d1}) makes a decision about an object, it will inform its upstream replica (assume it is R_u). It is possible that R_{d1}'s decision for an object is propagation while R_u's decision is invalidation. Obviously if R_u sends an invalidation message to R_{d1}, R_{d1} will not be able to propagate the new version of the object to all downstream replicas. We present a solution to this problem by distinguishing between the decision of a replica and what it tells its parent.

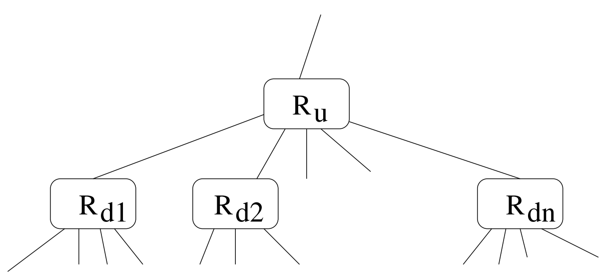

Fig. 2. Relations between a replica and its children

In general, we assume R_u has n children, $R_{d1}, R_{d2}, ..., R_{dn}$, as shown in Figure 2. If R_u's decision for an object agrees with all R_{di}'s ($1 \leq i \leq n$) decisions, there is no problem. If R_u's decision is propagation, it will propagate the new copy to all of its children and each R_{di} can do either propagation or invalidation based on its own decision. If R_u's decision is invalidation while the decision of some of its children is propagation, R_u will inform its parent that its decision is propagation rather than invalidation, so that its parent will propagate the updated version of the object to it. After R_u gets the new version, it will propagate the object to all R_{di}'s who tell R_u the decision is propagation, and it will send an invalidation message to all R_{di}'s who tell R_u the decision is invalidation.

The more accurate strategy comes at the cost of more control messages exchanged between replicas. Therefore we do not require all replicas make a decision for all objects. Those replicas not equipped with the functionality will simply forward the message. Usually we expect that only those replicas above a certain level make the decision. For those R_{di}'s that do not inform R_u about their decisions, R_u will propagate or invalidate based on its own decision.

6 Performance Evaluation

6.1 Simulation Setup

We use GT-ITM [26] to generate 1600 node transit-stub graphs. The origin server is randomly located in one of the stub domains. We experiment with 10, 50, 90 replicas, which are located in different stub domains at a node directly connected to a transit domain. We always use 50 replicas in the following simulations, unless we explicitly specify otherwise. These replicas and the origin server are connected by a multicast delivery network, which is established by first calculating a complete distance graph for replicas and the origin server, and then generating a spanning tree. This tree is used for delivering propagation and invalidation messages.

We use two performance measures. The first one is the *traffic generated*. If an object of size s goes through n hops, the traffic generated is counted as $s*n$. This is used in both multicast and unicast deliveries. Another interesting measure is the *freshness ratio*, which is the percentage of objects that are fresh when a request arrives at the replica. We want to emphasize that if a replica finds out that an object is not fresh, it will fetch a fresh copy from the origin server and deliver this new copy to the client. Therefore, all objects served by replicas are always consistent with that in the origin server, even if the freshness ratio is less than 100%. The freshness ratio only measures the probability that replicas can serve directly without resorting to the origin server. Therefore, the higher the freshness ratio, the shorter the response time.

6.2 Even Distribution of Requests

In this set of simulations, we assume that the requests are evenly distributed among all replica servers. The decision only needs to be made at the origin server. Assume that there are ten thousand objects at the origin server and the total request rate to these objects is either 0.1, 1 or 10 million times per day. We assume the distribution of requests to different objects follows the Zipf distribution [27,28]. If we order the objects from 1 to 10,000 according to their popularities, the probability of requesting objects $1, 2, 3 \cdots$ will be proportional to $\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \cdots$. Each request will be made to one replica selected randomly. We assume that the average size of objects is 10

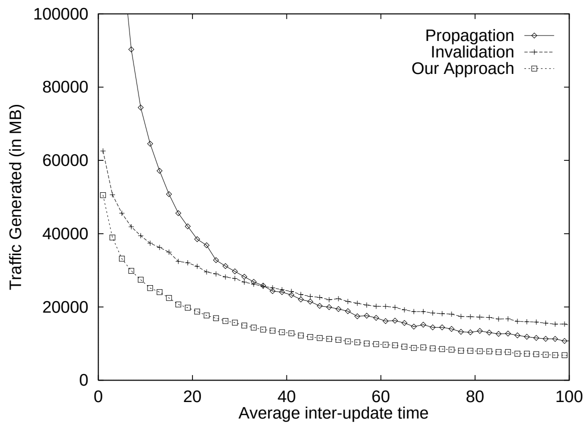

Fig. 3. Traffic generated (request rate = 1M)

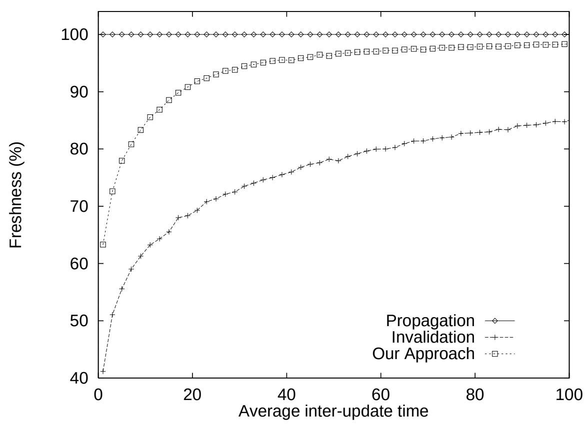

Fig. 4. Freshness ratio (request rate = 1M)

KBytes and the size of an invalidation for each object is 100 Bytes.

Next problem is to determine the *inter-update time* (the inverse of the update rate) of each object. Previous research has shown that there is no direct relationship between the request rate to an object and its inter-update time [28]. We use a random scheme to set the inter-update time for each object. We first choose an average inter-update time of all objects, τ. The inter-update time of an object is randomly chosen from $(0, 2 * \tau)$. We experiment with τ ranging from 1 to 100 hours. Note this is average inter-update time. The real inter-update time of an object can be very close to 0 and thus the update frequency can be very high.

We compare three approaches: (1) the propagation approach, (2) the invalidation approach, and (3) our hybrid approach. We first experiment with the request rate of one million per day and let the average inter-update time change from 1 to 100 hours. The traffic generated by each approach in one day is shown in Figure 3. When the average inter-update time is less than 10, the propagation approach generates extremely high volume of traffic because of the high frequency of blind propagation. When the inter-update time increases from 1 to 100, the traffic generated by all three approachs decreases. The propagation approach decreases faster

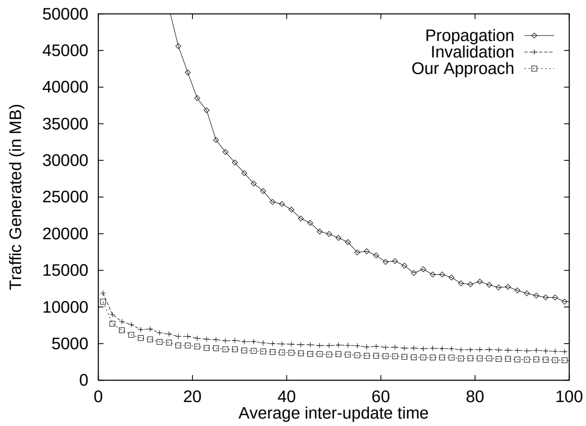

Fig. 5. Traffic generated (request rate = 0.1M)

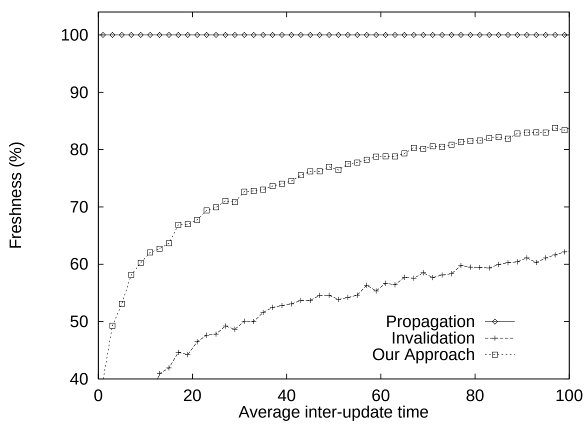

Fig. 6. Freshness ratio (request rate = 0.1M)

than the invalidation approach, because it takes advantage of multicast delivery network while the invalidation approach relies on unicast fetching by replicas from the origin server. When the average inter-update time is larger than 35, the propagation approach generates less traffic than the invalidation approach. Our approach chooses different methods for different kinds of objects and can always generate less traffic than both the propagation approach and the invalidation approach. Figure 4 shows the freshness ratio at replicas. The propagation approach always has 100% freshness ratio. Our approach has a significantly higher freshness ratio than the invalidation approach. The freshness ratio is greater than 90% when the average inter-update time reaches 19, at which point the propagation approach generates more than twice as much traffic as ours and the invalidation approach has a freshness ratio less than 70%.

Figure 5 and Figure 6 give the traffic and the freshness ratio when the request rate is 0.1 million per day. The traffic generated by the invalidation approach and our approach is much less than in the previous 1M request rate case because the number of unicast fetching from the origin server is smaller. They also come closer, and the difference between them and the propagation approach becomes larger, because the propagation traffic basically remains at the same level even when the request rate

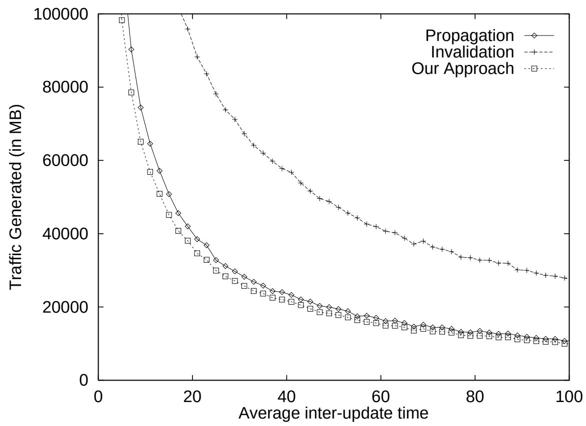

Fig. 7. Traffic generated (request rate = 10M)

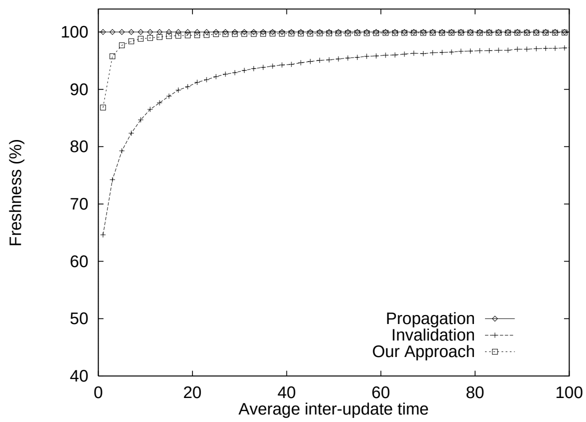

Fig. 8. Freshness ratio (request rate = 10M)

becomes much smaller. The freshness ratio of our approach is always 20% higher than the invalidation approach.

We increase the request rate to 10 millions per day in Figure 7 and Figure 8. The traffic generated by the invalidation approach and our approach is higher than in the 1 million request rate case, especially the traffic generated by the invalidation approach is significantly higher than both the propagation approach and our approach. This tells us that in the case of relatively high request rate, the propagation is more efficient than the invalidation approach because it makes use of the multicast delivery network. Our approach is only slightly better than the propagation approach because only a small number of documents can be exploited to reduce the traffic by using the invalidation method to maintain their consistency. In Figure 8, the freshness ratio of our approach is very close to 100% and much better than that of the invalidation approach.

Next we examine the effect of the number of replicas. We set the request rate to one million. In Figure 9 and Figure 10, we use 10 replicas. All three approaches generate less traffic than the previous 50 replica case in Figure 3, because objects need to be delivered to a smaller number of replicas. The cross point of the propa-

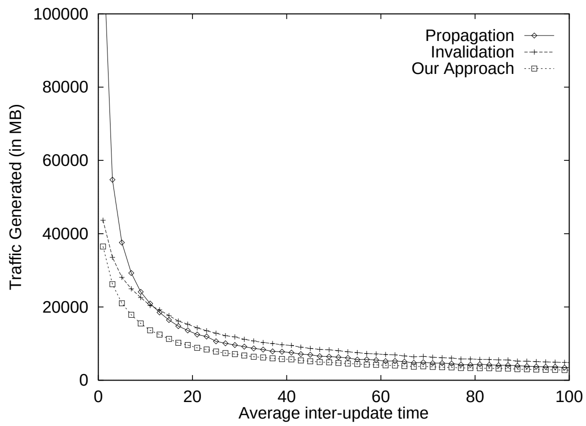

Fig. 9. Traffic generated (10 replicas)

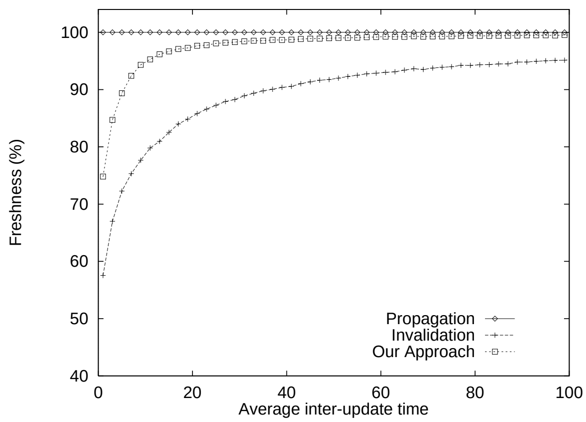

Fig. 10. Freshness ratio (10 replicas)

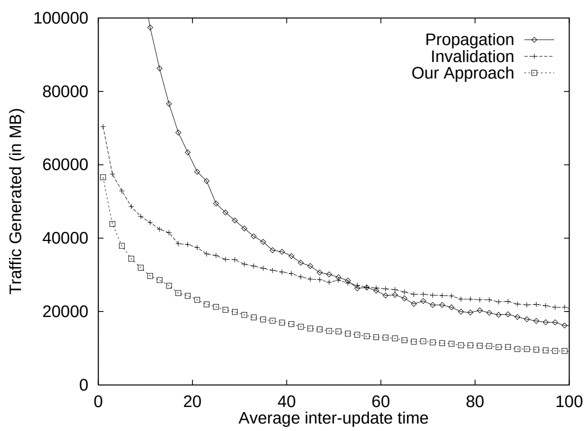

Fig. 11. Traffic generated (90 replicas)

gation approach and the invalidation approach shifts left. It changes from 35 in the 50 replica case to 10. Also the freshness ratio is higher than in the 50 replica case. Figure 11 and Figure 12 show the traffic and the freshness ratio when the number of replicas is 90. As expected, all three approaches generate more traffic than the 50 replica case and the cross point increases to 53. The freshness ratio is lower than

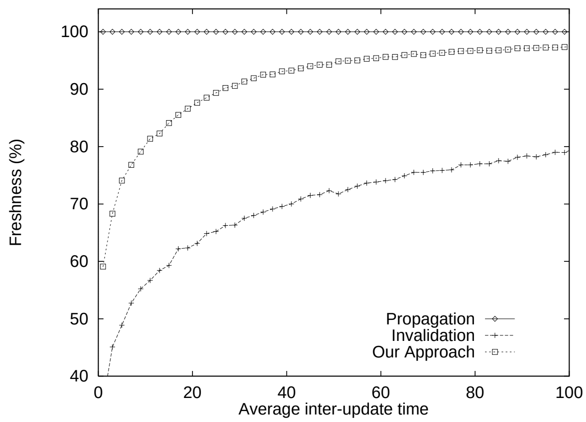

Fig. 12. Freshness ratio (90 replicas)

Table 1
Statistics of the trace

	All Pages	First 10,000 Pages
# of pages	802,968	10,000
Avg size	9,446	5,617
# of servers	25,319	1,567
# of clients	5,778	5,678
# of requests	2,135,398	706,411

that in the 50 replica case.

6.3 Uneven Distribution of Requests

To deal with the issue of uneven distribution of client requests, we proposed a hierarchical method in the previous section. We use a trace-driven simulation to evaluate its performance. The trace is from UC Berkeley Home IP Web Traces [29]. It lasts about 4 days and consists of 2,468,890 requests from 5780 unique clients. Preprocessing of this trace removes some requests and urls. We use requests to first 10,000 objects to drive the simulation. The statistics of the processed trace is summarized in Table 1.

In Figure 13, we show the number of accesses to each page. The most popular page (page 0) was accessed 56,517 times, while pages with ID number bigger than 234,000 were accessed only once. Figure 14 shows the distribution of the size of these pages. We put them in 1KB buckets. There are 174,788 pages that are smaller than 1KB. As the size increases, the number of pages decreases. There are 341 pages that are bigger than 1MB.

Each record in the trace consists of the time of the request, the client making the

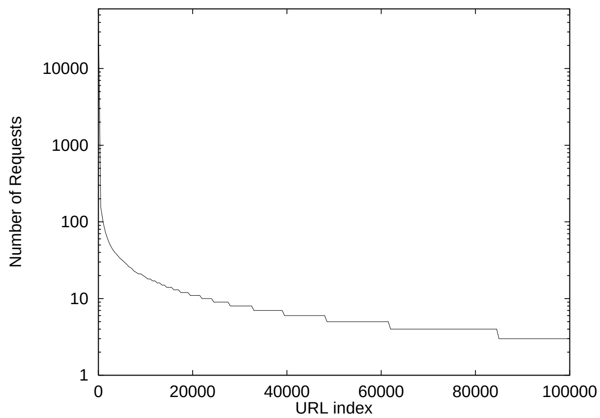

Fig. 13. The number of accesses to each page

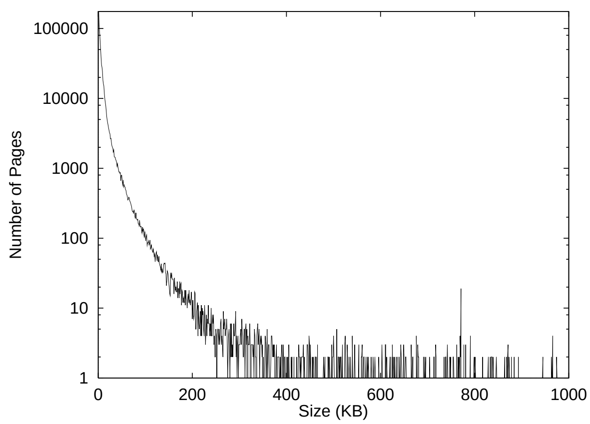

Fig. 14. The distribution of the page sizes

request, the url and the server the url is located. These requests are distributed to clients in the network. We assume that each client will make the request to its nearest replica server. As we understand the load on replica servers are unevenly distributed. We can get the size of each object from the trace, but information about the life time of each object is not complete in the trace. We use the similar model for the life time of each object as used in the even distribution case.

We compare four methods: (1) propagation, (2) invalidation, (3) root-only, which makes the decision to propagate or invalidate only at the origin server, and (4) hierarchical, which makes the decision at other nodes of the delivery tree as well. Figure 15 shows the traffic generated when we use a 1600 node transit-stub topology generated by GT-ITM. We can find that in this case the root-only method is better than both the propagation and invalidation methods. The hierarchical method further improves the root-only method, generating only about half of the traffic when the inter-update time is large.

In Figure 16, we generate a different topology and distribute requests to client nodes. It is interesting to notice that the root-only method generates slightly more traffic than the invalidation method. Our analysis pointed out that the distribution

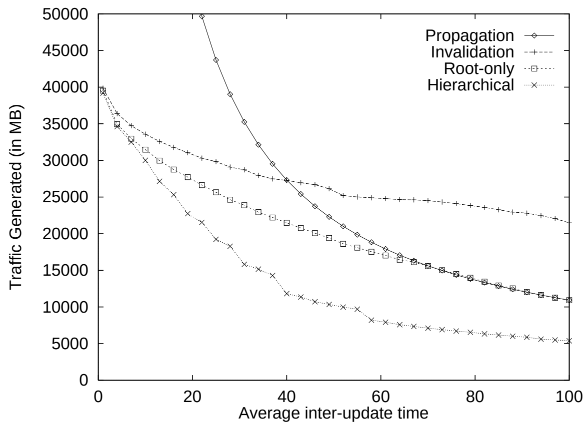

Fig. 15. Traffic generated

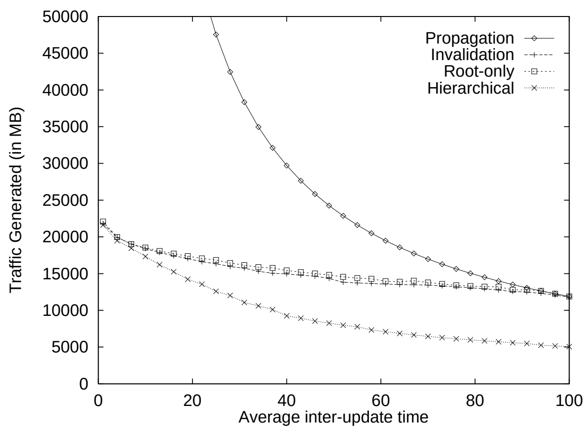

Fig. 16. Traffic generated

of requests are more skewed in this case. The assumption that the load is evenly distributed among replica servers is no longer true. For an object that the root-only method decides that the propagation is a better method, the invalidation method may actually perform better. Because in this case the traffic is concentrated on a small number of replica servers, the invalidation method requires only these replica servers to use unicast to get a new version of the page. The total traffic generated is less than the multicast delivery of the object to all replicas. This explains why the root-only method performs worse. The hierarchical method allows replica servers at lower levels of the tree to make a decision. This provides an opportunity to multicast the new version of the object to only those parts of the tree with a lot of requests. This can generate less traffic than the invalidation method, which needs each replica in those areas to fetch the new version of the object individually. In all topologies we generated, the hierarchical method can always perform better than either of the three methods.

7 Concluding Remarks

Content distribution network can be deployed to improve client performance by pushing the content towards the edges of the network and closer to clients. One of the key problems in CDNs is how to manage the consistency of content at replicas with the origin server, especially for those documents changing dynamically. This problem will become more prominent when more web sites use CDNs. In this paper, we proposed a novel approach in which the origin server determines the best way for each document. We explored the impact of multicast delivery network on our decision making process. By using the power law relation between multicast and unicast distributions, we can more accurately determine the traffic generated and make a right decision. The process of collecting request rates is made scalable to a very large number of replicas by aggregating information at the intermediate replicas. We also discussed how to extend the basic scheme to the case where requests to replicas are not evenly distributed. We performed extensive simulations to explore the effects of the request rate, the average inter-update time and the number of replicas. The simulations demonstrate that our approach can generate significantly less traffic than both the propagation approach and the invalidation approach.

8 Acknowledgement

This work was supported in part by the National Science Foundation under Grant CCR-0204304. The author would like to thank Dr. Jim Griffioen for his discussions of the subject and anonymous reviewers for their comments on the paper.

References

[1] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, K. J. Worrel, A hierarchical Internet object cache, in: Usenix'96, 1996.

[2] M. Day, B. Cain, G. Tomlinson, P. Rzewski, A model for CDN peering, Internet Draft Http://www.IETF.org/internet-drafts/draft-day-cdnp-model-09.txt.

[3] Y. Chu, S. Rao, H. Zhang, A case for end system multicast, in: Proceedings of ACM Sigmetrics, 2000, Santa Clara, CA.

[4] P. Francis, Yoid: Extending the Internet multicast architecture, http://www.icir.org/yoid/docs/yoidArch.ps.

[5] S. Deering, D. Cheriton, Multicast routing in datagram internetworks and extended LANs, ACM Transactions on Computer Systems 8 (1990) 85–110.

[6] Akamai, http://www.akamai.com.

[7] D. Island, http://www.digitalisland.net.

[8] Inktomi Corporation, http://www.inktomi.com.

[9] J. Nonnenmacher, E. Biersack, Asynchronous multicast push: AMP, in: Proceedings of ICCC'97, 1997, pp. 419–430, Cannes, France.

[10] Rodriguez, E. Biersack, Continuous multicast push of web documents over the Internet, IEEE Network Magazine 12 (2) (1998) 18–31.

[11] P. Rodriguez, E. W. Biersack, K. W. Ross, Improving the www: Caching or multicast?, in: Proceedings of the 3rd International Web Caching Workshop, 1998, Manchester, England.

[12] Y. Chawathe, S. McCanne, E. A. Brewer, An architecture for Internet content distribution as an infrastructure service, available at http://www.cs.berkeley.edu/ yatin/papers/scattercast.ps (Feb. 2000).

[13] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, V. Jacobson, Adaptive web caching: Towards a new caching architecture, Computer Network and ISDN Systems .

[14] J. Yang, W. Wang, R. Muntz, J. Wang, Access driven web caching, UCLA Technical Report 990007.

[15] J. Gwertzman, M. Seltzer, World-wide web cache consistency, in: Proceedings of the USENIX Conference, 1996, Copper Mountain Resort, CO.

[16] H. Yu, L. Breslau, S. Shenker, A scalable web cache consistency architecture, in: Proceedings of ACM Sigcomm'99, 1999.

[17] P. Rodriguez, S. Sibal, SPREAD: Scalable platform for reliable and efficient automated distribution, in: Proceedings of the 9th International World Wide Web Conference, 2000, Amsterdam.

[18] D. Li, D. R. Cheriton, Scalable web caching of frequently updated objects using reliable multicast, in: Proceedings of 2nd USENIX Symposium on Internet Technologies and Systems (USITS'99), 1999, Boulder, Colorado.

[19] E. Zegura, M. Ammar, Z. Fei, S. Bhattacharjee, Application-layer anycasting: A server selection architecture and use in a replicated web service, IEEE/ACM Transactions on Networking 8 (2000) 455–466.

[20] J. Chuang, M. Sirbu, Pricing multicast communication: A cost based approach, in: Proceedings of INET'98, 1998, Geneva, Switzerland.

[21] G. Phillips, S. Shenker, H. Tangmunarunkit, Scaling of multicast trees: Comments on the Chuang-Sirbu scaling law, in: Proceedings of ACM SIGCOMM'99, 1999, Cambridge, Massachusetts.

[22] R. C. Chalmers, K. C. Almeroth, Modeling the branching characteristics and efficiency gains in global multicast trees, in: Proceedings of INFOCOM'2001, 2001.

[23] K. Calvert, J. Griffioen, A. Sehgal, S. Wen, Concast: Design and implementation of a new network service, in: Proceedings of 1999 International Conference on Network Protocols, 1999, Toronto, Ontario.

[24] K. Calvert, J. Griffioen, A. Sehgal, S. Wen, Implementing a concast service, in: Proceedings of the 37th Annual Allerton Conference on Communication, Control, and Computing, 1999.

[25] Z. Fei, A novel approach to managing consistency in content distribution networks, in: Proceedings of Sixth International Workshop on Web Caching and Content Distribution (WCW'01), 2001, pp. 71–86, Boston, MA.

[26] K. Calvert, M. Doar, E. W. Zegura, Modeling Internet topology, IEEE Communications Magazine .

[27] G. Zipf (Ed.), Human Behavior and the Principle of Least Effort, Reading, MA: Addison-Wesley, 1949.

[28] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web caching and Zipf-like distributions: Evidence, and implications, in: Proceedings of INFOCOM'99, 1999, pp. 126 –134.

[29] UC Berkeley, UC Berkeley home IP web traces Http://ita.ee.lbl.gov/html/contrib/UCB.home-IP-HTTP.html.