Efficient Server Cooperation Mechanism in Content Delivery Network

Zhiyong Xu
Suffolk University
zxu@mcs.suffolk.edu

Yiming Hu
University of Cincinnati
yhu@ececs.uc.edu

Laxmi Bhuyan
University of California, Riverside
bhuyan@cs.ucr.edu

Abstract—
Content Delivery Network (CDN) plays an important role in today's web services. More and more content providers use CDNs to lower the server overhead, reduce client perceived latency and decrease network traffic. Many research papers have been published in recent years addressing CDN system performance issues. However, most of them are concentrated on server placement policy, content distribution mechanism and request routing algorithm. In this paper, we propose a new CDN architecture to improve system performance by grouping the content servers which are topologically close into server clusters and exploiting the benefits of server cooperation. In our approach, when a server receives a request and can not fulfill this request, it will forward the request to other nearby servers in this cluster. If there s a cache hit, the data can be fetched immediately. Only in case none of the servers can satisfy this request, the data will be fetched from the original server. Furthermore, with the server coordination, system workload can be well balanced on several severs. We conduct extensive simulations and the results show that our solution achieves significant improvement over the conventional CDN architecture.

I. Introduction

With the fascinating evolution in the past decade, Internet achieves great success and becomes a network containing hundreds of millions of users around the world. A key challenge in Internet service is how to deliver the information to the clients fast and efficiently. Numerous research papers addressed this issue. Web Caching [1], [2], [3], [4] and Content Delivery Network (CDN) [5], [6], [7] are two major mechanisms used in today's web services. By caching and replicating the web contents closer to the clients, Web Caching and CDN systems can greatly reduce client perceived latency and decrease network traffic. They use different approaches. Web Caching technique uses a reactive model, the data is cached on proxy servers only when it is required by a client. CDN system takes a more proactive policy: the data is fetched from the original server to the content servers which are topologically closer to the clients before any requests occur.

In Web Caching approach, the proxy servers belong to different ISPs, companies or universities. Server cooperation is very difficult. While in CDN model, a service provider owns all the content servers and they are under the unified administration. Each CDN has a central control facility which maintains all the information, such as server locations and cached files on the content servers within the CDN. Content delivery service achieved great successes in recent years, a large portion of Internet traffic are now generated by content service providers such as Akamai [8], Digital Island [9] and Mirror Image [10]. As more and more CDN service providers appear, the Internet Engineering Task Force (IETF) begin to make protocols and standards for the content delivery network [11] as well as content delivery internetworking [5], [6] which regulate the manipulation and cooperation within a CDN or among different CDN systems. However, content server coordination within a single CDN has not been fully considered.

In this paper, we use server clusters to exploit the benefits of content server cooperation within a single CDN network. In our approach, we group topologically adjacent content servers together to create a server cluster. The servers within a cluster have equal responsibility and provide service to all the clients. The client requests are fulfilled by the server coordination. Our approach is superior to the previous approaches. A server cluster can cache more data than using each server separately, a higher cache hit rate can be achieved. The global network traffic is reduced and the user perceived latency is decreased. Since new value-added web contents such as multimedia contents have high storage requirement, this is a big advantage.

The rest of the paper is organized as follows. First, we discuss the CDN architecture and the existing problems in Section II. New CDN models are described in Section III. We present our simulation environment and analyze experimental results in Section IV and V. The related work is discussed in Section VI and we summarize our contributions and describe the future work in Section VII.

II. Content Delivery Network

A. CDN Overview

Content delivery network offers promising benefits on web services. A CDN distributes the contents from the original web servers to its content servers, clients can fetch the contents from a topologically close content server instead of the remote original servers. The basic CDN components include *content distribution system, request routing system* and *accounting system*. The content distribution system is responsible for moving contents to replica servers, it interacts with the request routing system through feedback to assist replication decision. The request routing routes client requests to the optimal content server instead of the original server and the accounting system is in charge of billing and accounting information. A simple CDN service model is illustrated in Figure 1. More details of CDN architecture can be found in [11], [12].

Since CDN system can significantly reduce user perceived latencies and decrease Internet traffic, more and more content providers begin to use CDN service. The rapid growing requirement stimulates more CDN companies to appear and the scale of CDN system becomes larger and larger. For example,

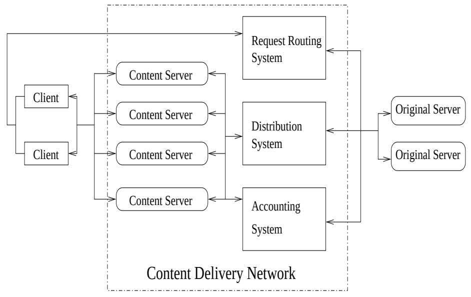


Fig. 1. CDN service model

Akamai has more than 15,000 content servers geographically distributed all around the world. Thus it provides a chance for server cooperation. In current CDN systems, the service model is client-server oriented, before a client request can be fulfilled by a content server, a central facility is responsible for choosing the designated server for that request. Since all the content servers in a CDN are belonging to the same service provider, the server placement policy, content distribution strategy and server selection decision are well controlled. System can easily determine an optimal server.

B. *CDN Problems*

To achieve high performance, a CDN system must be well designed and implemented. Current research concentrates on problems such as server placement strategy, request routing algorithm and the cooperation among different CDN service providers. Few efforts have been done on content server coordination within a single CDN. In a CDN, a content server has no knowledge about the data stored on its neighboring servers. If it can not satisfy a client request, it has to forward the request to the original server even in case the required data is available in its neighboring servers. Moreover, if a server is overloaded, its neighboring servers are not aware of that and can not help to reduce its workload even their workload are very low. Another serious issue is the cache space limitation, since more and more content providers use CDN service, the storage requirement increases dramatically, new services such as high-quality videos, video on demand, real-time video conferences and real time stock information generate even higher storage requirement and exacerbate the problem, thus the cache hit rate on these CDN servers keeps decreasing. We can expect in the near future, this issue will become more and more severe. New solutions are needed.

III. System Design

In this section, we describe the detailed design of our CDN architecture. Our system generates a cooperative structure to provide better CDN service. It can achieve higher cache hit rates and better server load balancing.

A. *Motivation*

The principle in our solution is to utilize the topology information. We group topologically close servers together to provide better service. It is motivated by the following factors: First, the number of content servers in CDNs (Akamai has more than 15,000 servers) is growing. There's opportunities to apply server cooperation strategy. Second, since the content servers are located on the edge of networks and they have high bandwidth connections, the communication latencies between servers which are located in neighboring networks are very small. For example, we choose a machine in University of Cincinnati as the base server and measure its average round trip latency with machines in Ohio State University, Purdue University and Indiana University, the results are 7.12, 9.17 and 9.86 ms respectively. From the client point of view, this difference is small and can be neglected. We can group them together to provide services to the clients in both Cincinnati and Columbus. A client does not have to figure out the requests are satisfied by a server in Cincinnati or Columbus. Third, with the server cooperation, we can achieve better load balancing, in case a server is overloaded, the neighboring servers can take over partial workload. The fourth and most important reason is, the storage requirement in CDN increases sharply. By grouping several servers together, we can achieve larger storage space than using them separately and a higher hit rate can be expected.

Utilizing topological information to boost large-scale distributed system performance is a hot research topic recently. In [13], [14], the authors improved DHT based P2P system routing performance by using this mechanism. In our approach, we take advantage of network topology information together with P2P techniques to improve CDN system performance.

B. *Content Server Cluster*

A server cluster is a group of content servers which are topologically close to each other. It plays the most important role in our system. Unlike P2P systems, grouping topologically close servers into a cluster is not difficult in CDN because all the servers are within the same administration domain. System administrator has the accurate location information about all the servers. For example, we can group servers located in Cincinnati and Columbus in one cluster, servers in San Jose and San Francisco in another cluster. A more accurate grouping mechanism is based on network latency measurement result. The basic idea is to aggregate the servers with the minimal round-trip latency into the same cluster. The server cluster creation procedure is done as follows:

First, we choose one content server which does not belong to any clusters as the start point. This server measures the link latencies to the other servers which are not within any clusters. Then it chooses the server with the smallest latency and adds that server into the current cluster. After the new server is added, the cluster has two members.

Second, the newly added server measures its network latencies with the remaining servers, and it chooses the server which has the smallest average latency to both itself and the first server to be the next one added into the cluster. This procedure continues until the certain number of servers is included.

The number of servers in a cluster is dynamic and can be changed according to network topology. In our current design, to simplify the management, the maximum number of servers in a cluster is set to 10. To avoid grouping two topologically separated servers together, we set a latency threshold, if all the current latency measurement results are higher than the threshold, even in case there's only one server in the cluster, we stop the cluster creation procedure and leave it alone. With this mechanism, we guarantee only the topologically close servers are aggregated. After one cluster is created, we choose another server from the remaining ones and begin a new procedure. The cluster creation procedure continues until all the servers are aggregated.

The cluster creation procedure only needs to be performed once for each server at the very beginning of CDN construction. After the construction finishes, the clusters will keep stable since

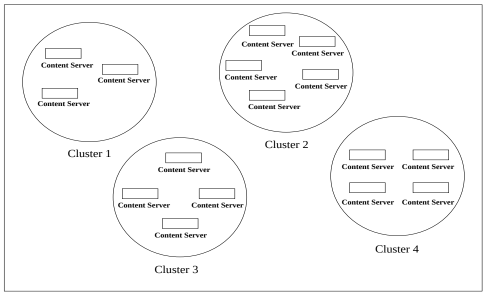


Fig. 2. Overview of a two-layer CDN system with server clusters

servers' locations seldom change. If new servers are added, the same algorithm can be used.

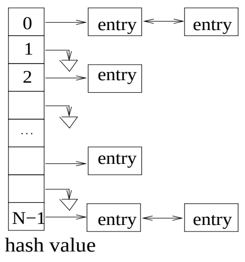
In a cluster, the functionality and responsibility of each server is equal and these servers cooperate together to provide the service. We call the servers in a cluster as peer servers. In our design, we use two-layer architecture. The higher layer includes all the servers. The lower layer is server cluster. A simple example is shown in Figure 2.

C. Cache Structure in Server Cluster

In our system, we keep the original design of server placement, content distribution and request routing mechanisms provided by the current CDN systems. Each client has a designated content server and it sends all its requests to this server. However, if the designated server does not have the requested data, it will check other servers in this cluster to fulfill the request before contacting the original server. Four schemes are proposed: *Fully-Administrated Cluster (FAC), Fully-Peer Cluster (FPC), Partial-Administrated Cluster (PAC)* and *Partial-Peer Cluster (PPC)*. These service models differ from each other mainly in the storage utilization strategy and content management mechanism. The storage space on each server is divided into two types: *individual cache space (ICS)* and *public cache space (PCS)*. ICS stores the frequently accessed data, each time when a request is fulfilled by this server or another server in this cluster, the content in ICS will be modified. It is only used by the clients whose designated server is the current server. For FAC and FPC models, no ICS caches created. PCS cache can be used by all the clients within the same cluster. The data in an ICS can be overlapped with other ICS or PCS in this cluster. The contents in PCS on servers will not be duplicated within the cluster. Each server has an *object location hash table (OLHT)* to maintain the content location information. Each time when a new request comes, system looks up this table for the requested data.

Each cached object has an *objectid* generated by a collision free hash function (such as SHA-1 algorithm), and each server is also given a *serverid* generated with the same algorithm. These ids are used in *FPC* and *PPC* models. In these two models, if the designated server can not fulfill the client's request, the *objectid* of the requested data is used as a search key. Each server also maintains a *Server Location Table (SLT)* for its request forwarding decision. The numerical space of *objectid* and *serverid* is equally divided into several zones according to the number of servers in a cluster, each server is responsible for one zone. When the designated server can not fulfill the request, it will forward the request to the server who is in charge of the corresponding zone that the current *objectid* belongs to. The data structure of OLHT table is illustrated in Figure 3. An entry in OLHT includes an object name (such as 1.jpg), objectid, hash value, its location information and in which table it is stored. A sample SLT table with four servers in a cluster (numerical space is 1024) is also shown in Figure 3. The usage of these tables will be discussed later.

Object Location Hash Table



Entry Structure:

object name	objectid	hash value	location: serverid	Cache Place

Server Location Table

serverid	zone
122	0 − 256
347	256 − 511
399	512 − 767
989	768 − 1023

Fig. 3. OLHT and SLT table structure

D. Cluster Service Models

With the introduction of server clusters, our CDN architecture aims to provide better CDN service under the high request rates and stringent storage requirement.

1) Fully-Administrated Cluster (FAC) Model: In this model, each server only has a PCS cache and an OLHT table. The contents in all PCSs within the cluster are not overlapped, thus we obtain large space in which can store more data than a single server. In its OLHT table, each server maintains the object location information in all the servers' PCS caches in this cluster. In this model, a client request is satisfied with the following steps:

1. A client sends the data request to the CDN request routing mechanism as in normal CDN system.

2. The request routing mechanism chooses the designated content server for this client and forwards the request to that server.

The first two steps only need to be done at the very beginning when the client does not know its designated server. In case the client has already been assigned a content server, it can send the request to the server directly.

3. The designated server checks its OLHT table. Different scenarios could happen:

a). If the requested data is in its own PCS cache, the server sends the data back to the client. This is the same as the original CDN service strategy.

b). In case it does not have the requested data in its PCS cache and its OLHT table shows another server in this cluster has the data, instead of fetching the data from the original server, the designated server forwards the request to the alternate server. The alternate server then fulfills this request by sending the data back to the client directly.

c). In case no server has the data, the designated server fetches the data from the original server and the client must endure the long network latency. The data will be cached on the designated server.

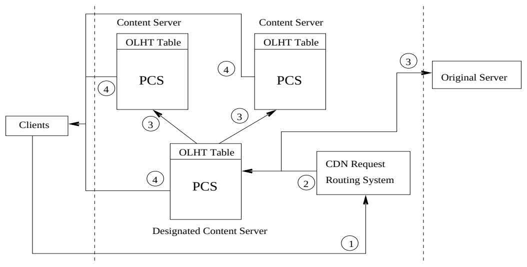


Fig. 4. Fully-Administrated Cluster (FAC) service model

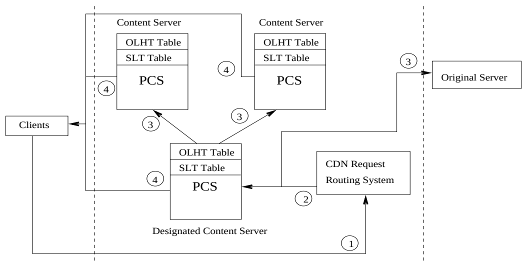


Fig. 5. Fully-Peer Cluster (FPC) service model

4. The client request is satisfied.

In FAC model, a client request can be fulfilled from a server's PCS cache or the original server. Suppose a cluster has *n* servers and each server has *k* GB space, then this cluster has *nk* GB cache space. Thus it can greatly improve the hit rate. Although fetching data from other servers is slightly slower than the designated server, it is still much smaller than fetching data from the original server. A simple illustration of FAC model is shown in Figure 4.

An obvious drawback in FAC model is the high maintenance overheads. Each time when data is added or removed from a server's PCS cache, the server must notify all the other servers to modify their OLHT tables accordingly. This is unacceptable in real world. We can define a time interval (*T*), the location information update only occurs once in each interval, to reduce the maintenance overhead. However, in this circumstance, the OLHT tables can not reflect the up-to-date object location information. In some cases, the designated server's OLHT table shows an object is cached in another server's PCS cache which has already been deleted. This inconsistency problem will hurt system performance. Another problem in FAC model is it can not achieve good load balancing, since each object only has one copy in a cluster, all the requests for a hot object goes to the same server which stores that object.

2) Fully-Peer Cluster (FPC) Model: In this model, servers in a cluster do not need to store all the object location information in this cluster. Each server has a SLT table to record all the servers' location information and their corresponding zones on the numerical space. Each server also has a PCS cache and an OLHT table to record the object information in its own PCS cache. However, unlike in FAC model where data can be stored on any server's PCS cache, here, the data can only be stored in the server which the data's objectid is mapped within this server's corresponding zone.

In this model, to serve a client request, the first two steps are the same as FAC model. In the third step, the designated server checks the objectid of the requested data. If the objectid maps to its responsible zone, the server searches the object in its OLHT table using the objectid as the key. If it finds the data in its PCS cache, it sends the data back to the client. If the data does not fall into its zone, the server checks its SLT table and forwards the request to an alternate server which is responsible for that particular zone. If the alternate server finds the data in its PCS cache, it will send the data back to the client. In case of not found, the alternate server will satisfy the request by fetching the data from the remote original server and it will keep a copy of the data in its PCS cache for future usage.

In FPC model, we relieve the high maintenance overhead problem in FAC model since servers do not need to maintain object information in other servers, but it increases the first time miss rate since only a small number of requests can be fulfilled by its designated server and most requests must be satisfied by an alternate server. As we mentioned early, the cache hit rate can be increased and system performance can be improved because the latency difference between fetching data from the designated server or an alternate server is very small. A simple explanation of FPC model is shown in Figure 5.

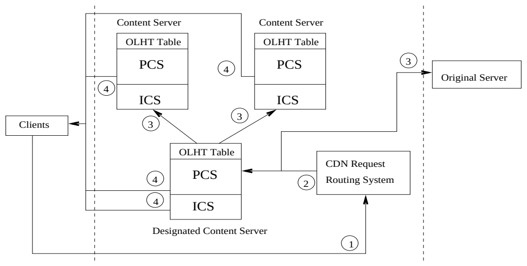


Fig. 6. Partial-Administrated Cluster (PAC) service model

In FPC model, we create a fixed mapping between an object and the server storing it. Since we use the collision free function to generate the objectids, we can expect nearly the same amount of files or even same amount of bytes will be cached on each server. Although we have a big PCS cache and reduce the overhead of the FAC, this model still suffers from the hotspot problem, the requests for a hot data can only be satisfied by the specific content server.

3) Partial-Administrated Cluster (PAC) Model: To solve the hotspot problem in the above service models, we introduce PAC model. In this model, each server has an ICS cache as well as a PCS cache. The content in ICS cache could be overlapped with any other ICS caches or PCS caches in any other servers in the cluster. It is used to store the frequently accessed data. As in FAC model, an OLHT table records all the object location information in PCS caches in this cluster.

When a client request comes, the designated server tries to satisfy the request from its ICS and PCS caches first. In case of data not found, it checks its OLHT table and sends the request to the alternate server which might have the data. In case no server's PCS cache has the data stored, the designated server fetches the data from the remote original server. PAC model is nearly the same as FAC model except it has an ICS cache to keep hot data. Also, in PAC model, if a client request is satisfied by an alternate server, the designated server will keep acopy of that data in its own ICS cache as well. Figure 6 shows PAC service model. Here, the data can be fetched from any server's PCS cache, the designated server's ICS cache and the original server.

Although the PAC model can solve the hotspot problem, each server needs to keep all object location information in its OLHT

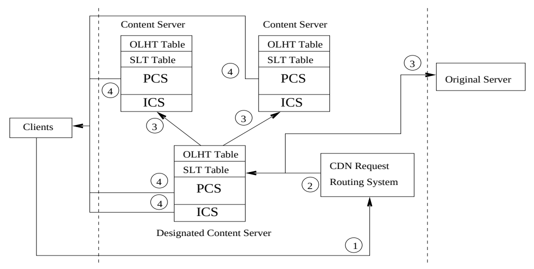


Fig. 7. Partial-Peer Cluster (PPC) service model

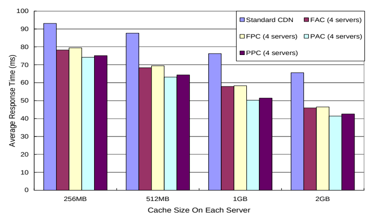


Fig. 8. Average user perceived latency comparison

table as FAC model, it still suffers from the high maintenance overhead problem.

4) Partial-Peer Cluster (PPC) Model: To relieve both hotspot and high maintenance overhead problems, we introduce the PPC service model. In this model, we use the same strategy as FPC model. The only difference is an ICS cache is created on each server to cache the hot data. As shown in Figure 7, in step 4, the client request can be satisfied by any server's ICS or PCS caches. It uses ICS cache to solve the hotspot problem and achieve load balancing. It also uses P2P architecture to avoid high maintenance overhead. It has better performance than all the other three models.

E. Data Replacement Strategy

Since clients in a server cluster are served by all the content servers in this cluster, the data replacement policies in ICS and PCS are important for the system to achieve good performance. In current design, we use LRU algorithm for all the models. In FAC model, if a requested object is not in PCS caches and must be fetched from the original server, the designated server will add the data in its PCS cache, in case no enough space left, it will check if any other server's PCS cache has enough space and store the data in that server's PCS cache. If it cannot find such a server, it will evict some data from its own PCS cache according to the LRU usage. In PAC model, the replacement algorithm is the same except it has one more operation, the designated server will place the data in its ICS cache as well, in case no space left in ICS cache, the server also uses the LRU replacement algorithm to select a victim data. After the contents in ICS and PCS caches changed, the corresponding tables are modified to reflect the modification.

In FPC model, the data can only be cached on the server who is in charge of the corresponding zone, it also uses LRU algorithm to select victim data in case of cache full. The difference between FPC and PPC models is, in PPC model, the data fetched from the original server is also stored in the designated server's ICS cache using the same LRU replacement algorithm.

F. Server Cluster Maintenance

In our system, when a new content server is added, a serverid is generated first and the related data structures are created according to the different service models. Then it is added to a server cluster. For FAC and PAC models, no data migration is needed. For FPC and PPC models, data migration is necessary. Assuming there are *n* servers in the cluster, the name space is divided into n equal size zones. After the new server joins, the number of servers is *n+1* and the name space is re-divided into n+1 zones. The content within the cluster need to be redistributed according to the objectids. In case a server is removed, for PAC and PPC models, the same operation is necessary. There's no operation need to be done for FAC and PAC models.

FPC and PPC models have more work to do in case of server join/leave. However, in CDN system, server join/leave operations seldom happen. The server clusters are relatively stable after it is created. No much maintenance work needs to be done.

IV. Experimental Methodology

We conduct trace-driven simulations to evaluate the performance of the service models.

A. Workload Trace

Most CDN networks are commercial systems, thus it is difficult to get the real world CDN traces. We take the web proxy logs obtained from the National Laboratory for Applied Network Research (NLANR) as workload traces as the workload. The trace data we use is collected from *UC* server between Feb 21st and Feb 24th, 2003. The total size of the individual files in the trace is 21.8GB.

B. Simulation Environment

We compare proposed models with the standard CDN configuration. To simplify the simulation, unless specified, the number of servers in proposed models is 4. Each client has a designated CDN server. Client requests are first submitted to this server before it can be satisfied by any other servers according to the different strategies. Only in case none of the servers has the data, the request will go to the original server. For the standard CDN configuration, client requests will be sent to the original servers if the designated server can not fulfill the requests. The cache replacement policy used is LRU. We define the overhead of satisfying a request from the designated server is 15ms. If the data is served by another server in the cluster, we add another 5ms communication latency, and set the overhead to 20ms. In case of the data must be fetched from the original server, the overhead is set to 100ms. To evaluate the effects of different cache sizes, we vary the cache space on each server from 256MB to 2GB. It is reasonable because in the real world, the cache space on each server is far smaller than the total amount of data which the original servers provide. The size of ICS cache is set to 1/3 of the PCS cache. For all the service models, we assume each server knows the accurate data location information.

V. Performance Evaluation

A. Average User Perceived Latency

The most important metric to measure CDN system performance is the average user perceived latency. In the first experiment, we compare the performance of proposed models with

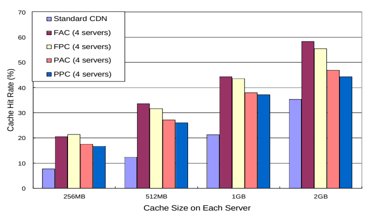


Fig. 9. Cache hit rate comparison

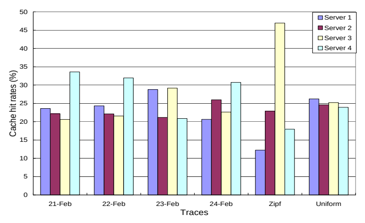


Fig. 10. Sever Load on Each Individual Server, PAC Model

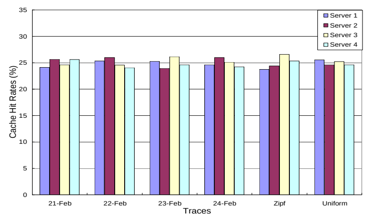


Fig. 11. Sever Load on Each Individual Server, PPC Model

the standard CDN configuration using this metric. Figure 8 shows the comparison results. Clearly, in our service models, with the server coordination, client requests have higher chance to be satisfied by the designated server or an alternative server than in standard CDN configuration, therefore all the four service models can achieve a lower average user perceived latency than the standard CDN system. With a cache size 256MB on each server, the average latency reduction is between 14.67% and 20.30%. This is because of the huge amount of overall data (21.8GB). As the cache size on each server increases, our models can achieve higher performance improvement because more space can be used to cache data within a server cluster. With a 2GB cache on each server, the average latency reduction increase to 29.14% and 36.91%.

Within four proposed service models, PAC has the lowest average latency because of the two features: First, it has an ICS cache to cache the hottest data, it efficiently improve cache hit rates on the designated server; Second, it has the unified space management for PCS caches on all the servers, this can efficiently avoid data duplication. Thus the system has the maximum size of cache to store data. PPC model also achieves good performance, it is only a little worse than PAC model. FAC and FPC models have worse performance that PAC and PPC models because they do not have ICS caches. FAC model has slightly better performance than FPC model because in FAC model, data can be cached anywhere within the cluster, while in FPC model, an object can only be stored on the server when its objectid fallen into the corresponding zone. Thus, FAC has higher PCS cache hit rate. We will analyze it in Section V-B. Although PAC and PPC have the best performance, PAC model introduces significant maintenance overhead. Thus, PPC model is the optimal configuration.

B. *Cache Hit Rate*

We examine the cache hit rates in the second simulation. The results are shown in Figure 9. We can generate two conclusions: First, the server coordination can efficiently increase the overall cache hit rate. The standard CDN system has the lowest cache hit rate. All the four service models have much better performance than it. Second, if measured by the metric: cache hit rate, FAC model has the best performance. FPC model also has very good performance which is slightly lower than FAC model. Although both PAC and PPC models can achieve lower average user perceived latencies, the cache hit rates in them are much lower than in FAC and FPC models. A dispatch appears if we compare with the results shown in Figure 8.

To figure out the reason, we divide the cache hits in PAC and PPC models into two categories: ICS cache hits and PCS cache hits. Table I shows the ICS and PCS cache hit rates in both PAC and PPC models. Clearly, the latency and hit rate dispatch comes from the usage of ICS cache. The latency of an ICS cache hit is much smaller than in PCS cache in most cases. Although FAC and FPC models have higher PCS cache hit rates, they have larger latency than PAC and PPC models. Thus, the introduction of ICS cache is crucial, it can efficiently improve system overall performance.

C. *Load Balancing*

To achieve good scalability, workload should be well distributed on all the servers. We evaluate load balancing property for the proposed models. In this experiment, we use a four-server configuration and assume each one contains one fourth of all the cached contents. If each server can cache all the contents, then a request definitely can be satisfied by its designated server, no server load balance can be achieved, thus we did not compare with the standard CDN design. The content is distributed evenly all across the four servers, in this experiment, as a request comes, each server has the equal opportunity to be selected as the designated server (we assume each designated server has nearly equal client population). The total requests number is truncated to 500000.

Figure 10 shows the measured results in PAC model. PAC model has optimal server load balance property only in case the client requests are uniformly distributed. For the real world web trace and Zipf synthetic traces, some servers have much heavy workloads than other servers. This is because of the cache organization in PAC model, the location of a hot object is fixed and the requests can only be satisfied by the particular server. FAC model has a little better performance if we can carefully assign locations for each data and distribute all across servers. However, the hot data changes from time to time, it is difficult to fix it.

We also evaluate PPC model and the results are shown in Figure 11. PPC model has good load balance property for all the traces. Even for Zipf distribution trace which a large amount of requests concentrated on hot objects, PPC model still can achieve good load balance. Clearly, this is because of the usage of ICS cache on each server to cache these hot objects. FPC model has similar results and we did not shown here due to the space limitation.

	256MB	512MB	1GB	2GB
PAC model, ICS cache	4.35%	6.47%	9.01%	12.04%
PAC model, PCS cache	13.19%	20.66%	28.97%	34.90%
PPC model, ICS cache	4.35%	6.47%	9.01%	12.04%
PPC model, PCS cache	12.46%	19.47%	28.12%	32.17%

TABLE I
ICS AND PCS CACHE HIT RATES IN PAC AND PPC MODELS

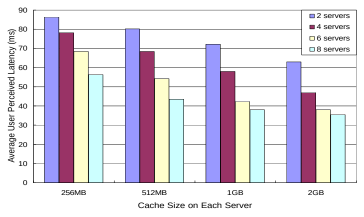


Fig. 12. Server number effects, FAC model

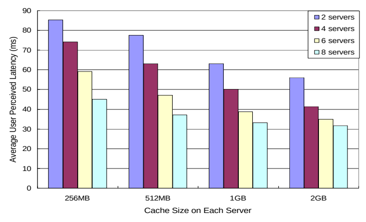


Fig. 14. Server number effects, PAC model

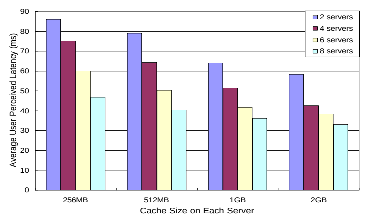


Fig. 15. Server number effects, PPC model

D. *Server Number Effects*

To evaluate the effects of different number of servers in a cluster, in this simulation, we compare the system performance with different number of servers in a cluster and the results are shown in Figure 12, 13, 14 and 15. Obviously, the average latency decreases as the number of servers in a cluster increases since system has larger storage space to cache data. For a 2-server configuration, system has the lowest performance. However, it is still outperforms the standard CDN system. The average latency reduction is between 5.36% and 17.23% with different cache size. As the number of servers within a cluster increases, the average latency drops dramatically. With a 8-server configuration, the average reduction is between 37.34% and 57.64%. Again, PAC and PPC models have the best performance. From this experiment, we can conclude that with more servers introduced, system can achieve higher performance. On the other hand, the maintenance overhead for FAC and PAC models also increases significantly. For FPC and PPC models, this does not happen.

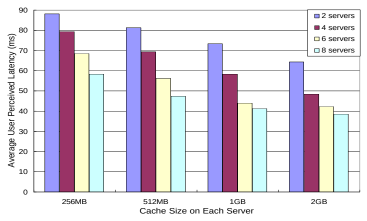


Fig. 13. Server number effects, FPC model

VI. RELATED WORKS

Many research papers discussed the server placement strategy in distributed networks including CDN [15], [16], [17]. They used graph theories to find out an optimal result. However, in these systems, CDN is viewed as a static system. These approaches can achieve good performance under certain conditions such as relatively stable clients and semi-accurate network distance measurement results. However, none of these systems considered the potential advantage of using content server coordination.

Since a lot of CDN networks have already been deployed on Internet, utilizing resources among different CDNs could achieve more benefits. Thus, CDN network cooperation becomes a hot research topic recently. IETF are now working on standard protocols for the Content Delivery Internetworking (CDI) [6], [5]. CDI can be viewed as the next generation edge and overlay service, it creates the interface for CDNs to utilize other's resources. We have different objects. Unlike our proposal, CDI aims to provide cooperation among content servers in different CDNs. Our proposal is concentrated on exploiting the potential benefits of server coordination within a single CDN. Compared to CDI, our proposal is much simpler, it does not need to deal with difficult problems such as create interfaces among different CDNs. Also, the security problem is much easier.

Web Caching [18] reduces network traffic and decreases user perceived latency with a different strategy from CDN. In current web caching design, hierarchical architecture [19], [2] is a very important technique to address scalability problem. However, in most cases, web proxy servers are belonging to different ISPs, the administration and coordination among proxy servers are much more difficult than in CDN environment.

DHT based P2P systems [20], [21], Tapestry [22], Chord [23] and CAN [24] are scalable solutions for wide area distributed services. However, most DHT P2P algorithms are still under research and are not widely deployed. One reason for this is the decentralized nature of P2P systems. In such systems, the routing, caching and security problem is much more difficult than client-server systems. The members of P2P systems are unstable, they may join/leave suddenly and system performance is degraded in dynamic systems. In CDN systems, all the content servers are stable and under the same administration, these problems are much easier to solve than P2P systems.

New P2P algorithms such as [13], [14] also use topological information to boost P2P system routing performance. By grouping topologically adjacent nodes together and making the routing hops via the low latency links, P2P routing performance can be improved. However, these schemes still suffer from problems such as dynamic join/leave nodes. Also, to discover the topological information of a newly joined member, these systems encounter considerable network measurement overhead.

In [25], Kangasharju applies P2P technology to improve CDN system performance. As [26], their strategy is to utilize the aggregation of storage space offered by the clients within an AS. Clearly, this approach also suffers from the unstable client population problem. While in our approach, the cooperation is among reliable content servers which can avoid this problem.

VII. Conclusions and Future Work

In this paper, we propose a new CDN architecture by creating server clusters in a single content delivery network. Our architecture can achieve better server load balance and fit for the future large storage requirement for CDN services. With the coordination among content servers, client requests are satisfied by different servers within a cluster. FPC and PPC models can greatly reduce the user perceived latencies in FAC and PAC models. Also, FPC and PPC models have much less maintenance overhead than FAC and PAC models. Among all the four service models, PPC is the best choice.

In the future, more work can be done to evaluate the efficiency of the proposed approach. We will use more realistic network models and design new algorithms for ICS and PCS cache organization and data replacement strategies. We will also try to transplant the idea on web caching systems. Although applying this idea on proxy servers may face more difficult problems than CDN systems since proxy servers are not belonging to the same company, we believe grouping topologically-closed proxy severs to form proxy clusters can effectively improve web caching system performance.

VIII. acknowledgement

This research has been supported by NSF Grant CNS-0509207 and a research grant from Intel Corporation.

References

[1] P. Cao and S. Irani, “Cost-Aware WWW Proxy Caching Algorithms,” in *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, (Monterey, CA), 1997.

[2] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, “Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol,” *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000.

[3] B. D. Davison, “A Survey of Proxy Cache Evaluation Techniques,” in *Proceedings of the 4th International Web Caching Workshop*, (San Diego, CA), March 1999.

[4] J. Dilley and M. F. Arlitt, “Improving Proxy Cache Performance: Analysis of Three Replacement Policies,” *IEEE Internet Computing*, vol. 3, no. 6, pp. 44–50, 1999.

[5] F. Douglis, I. Chaudhri, and P. Rzewski, “Known Mechanisms for Content Internetworking.” IETF Draft (draft-douglis-cdi-known-mech-00.txt), June 2001.

[6] M. Day, B. Cain, G. Tomlinson, and P. Rzewski, “A Model for Content Internetworking (CDI).” Internet Draft (draft-ietf-cdi-model-01.txt), Feb. 2002.

[7] S. Gadde, J. S. Chase, and M. Rabinovich, “Web Caching and Content Distribution: A View From the Interior,” *Computer Communications*, vol. 24, no. 2, pp. 222–231, 2001.

[8] Akamai, “http://www.akamai.com.”

[9] Digital Island, “http://www.digitalisland.net.”

[10] Mirror Image, “http://www.mirror-image.com.”

[11] M. Green, B. Cain, G. Tomlinson, and E. S. Thomas, “CDN Peering Architectural Overview.” IETF Draft (draft-green-cdnp-gen-architecture-00-01.txt), Nov. 2000.

[12] G. Peng, “CDN: Content Distribution Network,” Technical Report TR-125, Computer Science Department, State University of New York, Stony Brook, NY, Jan 2003.

[13] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, “Topologically-Aware Overlay Construction and Server Selection,” in *Proceedings of IEEE INFOCOM’02*, (New York, NY), Jun. 2002.

[14] Z. Xu, R. Min, and Y. Hu, “HIERAS: A DHT-Based Hierarchical Peer-to-Peer Routing Algorithm,” in *the Proceedings of the 2003 International Conference on Parallel Processing (ICPP’03)*, (Kaohsiung, Taiwan, ROC), pp. 187–194, October 2003.

[15] P. Radoslavov, R. Govindan, and D. Estrin, “Topology-Informed Internet Replica Placement,” in *Proceedings of WCW’01: Web Caching and Content Distribution Workshop*, (Boston, MA), June 2001.

[16] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, “On the placement of internet instrumentation,” in *Proceedings of IEEE INFOCOM*, (Anchorage, AL), pp. 295–304, April 2000.

[17] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, “On the Placement of Web Server Replicas,” in *Proceedings of IEEE INFOCOM*, (Anchorage, AL), pp. 1587–1596, April 2001.

[18] G. Barish and K. Obraczka, “World Wide Web Caching: Trends and Techniques,” in *IEEE Communications Magazine Internet Technology Series*, May 2000.

[19] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, “A Hierarchical Internet Object Cache,” in *Proceedings of USENIX Annual Technical Conference*, (Toronto, CA), pp. 153–164, June 1996.

[20] A. I. T. Rowstron and P. Druschel, “Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems,” in *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, (Heidelberg, Germany), pp. 329–350, Nov. 2001.

[21] A. Rowstron and P. Druschel, “Storage Management and Caching in PAST, A Large-Scale, Persistent Peer-to-peer Storage Utility,” in *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, (Banff, Alberta, Canada), pp. 188–201, Oct. 2001.

[22] B. Zhao, J. Kubiatowicz, and A. Joseph, “Tapestry: An infrastructure for fault-tolerant widearea location and routing.” Technical Report UCB/CSD-01-1141, U.C.Berkeley, CA, 2001.

[23] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for Internet applications.” Technical Report TR-819, MIT., Mar. 2001.

[24] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A Scalable Content Addressable Network.” Technical Report, TR-00-010, U.C.Berkeley, CA, 2000.

[25] J. Kangasharju, J. Roberts, and K. Ross, “Object Replication Strategies in Content Distribution Networks,” in *Proceedings of WCW’01: Web Caching and Content Distribution Workshop*, (Boston, MA), June 2001.

[26] L. Xiao, X. Zhang, and Z. Xu, “On Reliable and Scalable Peer-to-Peer Web Document Sharing,” in *Proceedings of International Parallel and Distributed Processing Symposium, (IPDPS’02)*, (Fort Lauderdale, FL), Apr 2002.