DOH: A Content Delivery Peer-to-Peer Network

Jimmy Jernberg[1], Vladimir Vlassov[1], Ali Ghodsi[1],[2], and Seif Haridi[1],[2]

[1] School for Information and Communication Technology (ICT), Royal Institute of Technology (KTH), Stockholm, Sweden
[2] Swedish Institute of Computer Science (SICS), Kista, Sweden

Abstract. Many SMEs and non-profit organizations suffer when their Web servers become unavailable due to flash crowd effects when their web site becomes popular. One of the solutions to the flash-crowd problem is to place the web site on a scalable CDN (Content Delivery Network) that replicates the content and distributes the load in order to improve its response time.

In this paper, we present our approach to building a scalable Web Hosting environment as a CDN on top of a structured peer-to-peer system of collaborative web-servers integrated to share the load and to improve the overall system performance, scalability, availability and robustness. Unlike cluster-based solutions, it can run on heterogeneous hardware, over geographically dispersed areas. To validate and evaluate our approach, we have developed a system prototype called DOH (DKS Organized Hosting) that is a CDN implemented on top of the DKS (Distributed K-nary Search) structured P2P system with DHT (Distributed Hash table) functionality [9]. The prototype is implemented in Java, using the DKS middleware, the Jetty web-server, and a modified JavaFTP server. The proposed design of CDN has been evaluated by simulation and by evaluation experiments on the prototype.

1 Introduction

The major focus of our research presented in this article is to design and evaluate a scalable Content Delivery Network (CDN) built on top of a structured P2P system that provides the Distributed Hash Tables (DHT). Such CDN can be used as a Web-hosting environment that allows improving the overall performance and storage capacity, scalability and availability of hosted Web sites.

As a motivational scenario, assume a small company has a web server on a 10 Mbit broadband line, which usually serves it well. One day a large news portal reviews and recommends the company site to the portal users. Since the site becomes a ”hot object”, it starts generating a huge amount of hits. Subsequently, the company's web server will not be able to cope with the strain, and, eventually, its bandwidth will be totally consumed, making the company's Web-pages unavailable. The situation described above is called the flash crowd effect (also known as the SlashDot effect[1]), when a sudden increase in traffic makes a web site completely unavailable.

One solution for a company to survive a flash crowd is to pay for joining a proprietary CDN like the one owned by Akamai[2] that offers services in distributing the load of heavily trafficked web sites for companies with an extensive web presence. For SMEs and organizations without the need for a CDN on a daily basis, the incurred costs of placing their web-sites on a proprietary CDN might be considered too high.

In our view, one of the cost-efficient approaches to building high-performance and scalable web-sites, CDNs and Web-hosting systems, is to integrate several (open-source) web-servers in a scalable structured P2P system with DHT functionality. A (part of the) URL of a Web-page can be used as a key to determine a web-server on which the page is (to be) stored. The P2P system of web-servers should support content replication in order to improve performance and availability of hosted Web-sites. We believe that this approach should make it possible for SMEs and small organizations to obtain at an affordable price the same hosting services that have been available to big companies for years. CoralCDN[12] is an existing P2P CDN that are already deployed, but while CoralCDN is designed to be an overlay network for handling the flash crowd effect DOH aims for more: to be a low cost, transparent, web-hosting service with a built in ability to handle a flash crowd.

Extensive research has been done in building efficient DHTs on top of structured P2P overlay networks, see e.g.[3], [13], [23], [25], and [26]. A DHT provides a distributed indexing service based on hashing and like an ordinary centralized hash-table, a DHT, whose buckets are distributed among peers, can be used for storing of different kind of information. Note that the DHT should be "open": in the case of a hash collision, when different entries are hashed to the same bucket, a single bucket can contain multiple entries, which should be searched sequentially.

In this paper, we present our approach to building a scalable Web Hosting environment as a CDN on top of a structured P2P system with DHT functionality. In our design, several Web-servers are organized in a structured P2P system in order to share their load and to improve the overall system performance, scalability and storage capacity, as well as availability of hosted web-sites. The underlying P2P overlay network provides an efficient and scalable lookup mechanism needed for DHT, replication and ability to automatically self-organize when nodes join/leave the network.

The DHT is used for fetching and storing web pages. Each of the web-servers is responsible for a region of DHT buckets used to store Web pages, referenced by URLs. Even though the worst-case lookup latency in a structured P2P system with N peers is O(log N), building a Web-hosting environment as a structured P2P system allows improving overall performance and scalability of Web-hosting due to multiple access points, well-balanced load distribution and content replication, increase in overall storage and computational capacity of the P2P CDN. In our design, we use a sophisticated content replication mechanism, called symmetric replication [15], in order to even more improve the system performance, availability and reliability.

To validate our approach, we have developed, implemented and evaluated a system prototype called DOH (DKS Organized Hosting) that is a content delivery network implemented on top of the DKS (Distributed K-nary Search) structured P2P system with DHT functionality [9]. DOH provides the same features as a corporate CDN at the same cost as a regular low-end web server. The system prototype is implemented in Java, using DKS[3], the Jetty[14] web server, and a modified JavaFTP[6] server package. We have evaluated our proposed CDN design by simulation and by performing evaluation experiments on the developed prototype.

The remainder of the paper is organized as follows: Section 2 describes the DOH architecture. Section 3 presents our DOH prototype. Section 4 presents results of preliminary performance evaluation. Section 5 discusses some related work. Conclusions and future work are given in Section 6.

2 DOH Design

When designing the DOH content delivery network, two different types of users should be considered: a regular user (called User) browsing the Web; and a content provider (called here Publisher) publishing content of a web-site in DOH. As shown in Figure 1, DOH consists of two types of nodes: Translators and DOH-Nodes (or shortly Nodes).

The DOH-Nodes are connected by the DKS P2P middleware in a structured P2P network. Each DOH-Node contains an FTP server, a web server, and is connected to the DKS overlay network (see Figure 1). It serves HTTP requests submitted by Users; confirms identity, inserts, and removes content provided by Publishers.

Translator nodes handles interaction between the User and the system before an HTTP request is sent to a DOH-Node. A Translator redirects the User's browser to DOH-Nodes based on a load-balancing strategy. Each Translator maintains a cache for storing information about other Translators and DOH-nodes including their load status and RTT times, referred to as the Translator-cache. This information is used for redirection decisions, and when Nodes join.

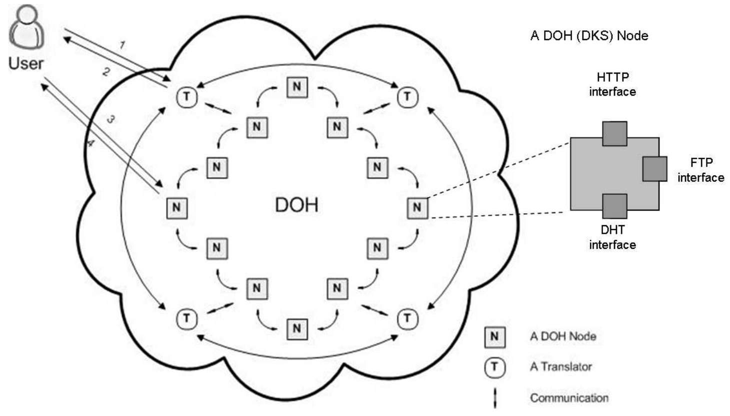


Fig. 1: Architecture of the DKS-based Hosting (DOH) P2P Content Delivery Network

Servicing of an HTTP request arrived to one of the DOH Translator nodes, passes the following steps:

1. Redirection from the old home to a Translator. This step is performed when the DNS entry for the requested web page has been updated;
2. Redirection from a Translator to a DOH-node based on current load of the DOH nodes and network congestion;
3. Retrieval of the requested file (replica) from the Translator-cache of the node or, if the cache misses, from the DHT of the DKS P2P system.
4. Unwrap, assemble, and write the file to the disk (cache) of the requested DOH node;
5. Sending the requested file to the requesting client.

2.1 Translator

A web-hosting system like DOH allows storing (and replicating) content of several web sites in one hosting system. As the content is referenced to by URLs, the system needs to direct a HTTP request to one of DOH-Nodes that serve as access points to the content, i.e. it needs to translate a requested URL to a new URL that redirects a requesting client to one of the DOH-Nodes. To perform this URL-to-URL translation, the DOH system includes Translator nodes that are Users' initial access points to the DOH content delivery network. Translators serve as mediators that redirect web clients to one of the DOH-nodes based on their current load and network congestion. Thus, the URL-to-URL translation performed by Translators aims at load balancing in order to improve availability and performance of the DOH content delivery network.

To perform load balancing, each Translator maintains a cache of information on DOH-nodes: IP-addresses, load, and RTT values; and information on other Translators it knows. This cache is called Translator-cache. When a Translator receives an HTTP request, it checks the Translator-cache to find the currently "best" DOH-node that can service the request. To redirect the client to a DOH-node, the Translator issues a 302-code message that, according to the HTTP standard[11], is used to respond on requests for temporarily moved pages, and adds the IP address of the Node to the new URL when forming the 302-code response to redirect the requesting client. E.g., if the riginal URL is `http://www.url.com/a/b/index.html`, then the translated URL is `http://192.168.2.23/www.url.com/a/b/index.html`, where `192.168.2.23` has been chosen as the currently "best" DOH node to service the request.

Information in the Translator-cache is periodically updated by the Nodes. The data collected in the Translator-cache are used to calculate Nodes' load and network congestion over time. The Translator-cache is also used for bootstrapping, as it contains information on Nodes and Translators that are known to be up and running. There are three levels of the caching structure: (1) level 1 keeps a list of other known Translator-caches, (2) level 2 is the local Translator-cache level, and (3) level 3 is a Translator-cache-entry that stores data on an actual Node. When a new Node joins DOH, it first contacts the Translator-cache (if any) it used last time. If that cache is down, the Node queries a cache list for online caches. The queried Translator-cache may respond with several valid entries, and the new (booting) Node contacts one of these Nodes to join the system. The same mechanism is used when a Publisher wants to find a Node to upload its content on DOH.

2.2 DOH-Node

In DOH, web content and its replicas are stored in a hash table distributed among the DOH-nodes. The content is replicated in DHT according to the symmetric replication mechanism used in DKS[15].

Each DOH-Node includes three subsystems which are the DKS middleware that connect the Node to the P2P DKS overlay network, an FTP server, and a web server. When a client requests a file, the web server searches the file locally, and if the file does not exist locally, or the local replica is considered too old, the web server will perform a lookup operation in the DKS DHT to retrieve the requested file. The FTP server of a DOH-node is used when Publishers upload content.

Granularity of content stored in DHT

To store contents (or content references) of hosted web sites, the DOH CDN uses a Distributed Hash Table (DHT) provided by the DKS P2P middleware[3]. When content of a web site is stored to or fetched from the DHT, either the entire URL or a part of the URL is considered as a key. The hashed key value determines a DOH-node responsible for the DHT-bucket in which the content is (to be) stored.

There are three strategies of placement of web-site content identified by URLs to the DHT, that differ in the granularity of a web-site content stored in a bucket of the DHT: (1) file-wise placement (uses the entire URL as a key, e.g. `http://www.url.com/a/b/index.html`); (2) directory-wise placement (uses the directory part of the URL as a key, e.g. `http://www.url.com/a/b/`); (3) site-wise placement (uses the web-sire part of the URL as a key, e.g. `http://www.url.com/`).

With the file-wise placement, files that belong to the same web-site can be stored in different DHT buckets and distributed among the DOH-nodes.

With the directory-wise placement, all files of the same directory are hashed to the same bucket, i.e. stored on the same DOH-node. Even though the files are hashed directory-wise it does not mean they have to be returned directory-wise on a DHT get request. The DKS API allows a file name to be sent along with the DHT get request as an additional parameter to retrieve specific entry (the requested file).

The web-site-wise placement is a coarse-grained placement and is similar to the directory-wise placement described above. The site-wise placement causes the entire content of a web-site to be stored in the same DHT bucket.

The coarser the placement is, the lower is the level of content distribution. In our DOH prototype, we support all three different levels of granularity. Preliminary evaluation of file-wise and directory-wise distribution shows that the system performance is not very sensitive to the granularity of the content distribution in the DHT but rather to prefetching and caching. One can expect that the file-wise and directory-wise distribution allows improving performance in the case of intensive concurrent accesses to a web-site, as well as improving its availability in the case of node failures. We leave more detailed evolution of the distribution strategies to our future work.

Symmetric Replication and Adaptive Caching

DKS builds on symmetric replication, which is built on-top of the DHT layer. Symmetric replication of DHT content enables parallel lookups, which increase the responsiveness of the system, while keeping the number of messages needed for restoring the replication degree after dynamism low. In addition to the symmetric replication for reliability and higher performance, DOH also implements an adaptive caching of requested content at DOH-nodes. The caching algorithm has been devised based on a combination of the Directory scheme defined in [16] with the entry caching scheme of DNS. We assume that whenever an object (a file) is requested, it is likely to be requested again from the same or another access point. Therefore it makes sense to cache the object on its way to the node that originates the lookup operation. In DKS, the return path of passing the object to the requesting node is recursive; therefore the object can be cached in the nodes along the return path. Consistency of copies is weak and can be kept by using the if-modified-since field built-in to the header of the HTTP protocol. When a cache is full, the Least Recently Used (LRU) algorithm or some other caching policy, could be used for deciding which objects to evict.

3 A System Prototype

We have implemented the DOH prototype in Java, using the DKS P2P middleware with the DHT API[3], the Jetty[14] web server, and a modified JavaFTP[6] server.

In the DOH-Node prototype, the web server functionality has been implemented by modifying the Jetty web server, which is licensed under the Apache license[4]. For uploading, downloading and removing content in DOH, we use the modified JavaFTP server package, also licensed under the Apache license. The DOH-Nodes are peers in the P2P DKS network, and the DKS DHT API is used to store and retrieve content of web sites hosted in DOH. In order to integrate the Jetty server in our prototype, we have extended the server so that it creates a special handler that searches the DKS DHT if a requested file is not found locally in the web server's cache.

JavaFTP server is a package that implements the FTP standard[21]. We have modified the server so that it allows a content provider to upload the files in the DHT rather than to the host's local file system. In order for the DOH system prototype to handle large objects, it uses data fragmentation that, we believe, allows better control over memory usage and to avoid running out of memory on a low-end server.

When a file is stored to the DHT the following steps are performed:

1. The key (it can be either the file name or the directory name or the web-site name) is hashed to get a hash table index using SHA-1[10], shortened to 64 bits.
2. If necessary (it depends on the file size), the file is fragmented;
3. Each of the fragments or the entire file is put in the DHT entry defined by the hash-table index. The put operation is based on the DKS lookup operation that finds a DOH node responsible for the target bucket. placed.

When retrieving a file from DHT, a DOH node performs the following steps:

1. The key is hashed to obtain a hash table index;
2. A DHT get operation based on the DKS lookup operation is performed that returns an array of entries (all the files) stored in the target DHT entry;
3. If fragmented, the requested file is assembled by combining the fragments. Copies of the file are stored in the web-caches of the nodes involved in the operation.

3.1 Implementation of a Translator

A Translator is a stand-alone node that serves as a web server for web-clients accessing web-sites hosted in DOH. Translator receives HTTP requests and redirects the clients to DOH-nodes. Translator provides the following functionality: (1) maintains a cache of information on DOH nodes (IP addresses, load and RTT times) and on other Translators (to retrieve information from their caches); (2) provides load balancing so that it redirects HTTP requests to DOH-Nodes based on their load and network congestion; (3) displays Node information to Publishers in a human-readable format.

Each Translator redirects the clients to DOH nodes as described above except of a special case when the requested URL refers to `doh_webcache.xml` indicating that a Publisher asks for an IP address of a Node to upload content. In this case, Translator replies with an XML page that contains information on Nodes from its own Translator-cache. From this file the Publisher can choose a node to connect to.

4 Preliminary Evaluation

In this paper, we present results of preliminary evaluation of the approach, leaving more detailed evaluation to our future work. The DOH prototype has been used to evaluate small-scale configuration mostly in order to verify the design, whereas a specially developed DOH simulator has been used to evaluate impact of different design choices (such as the use of content caches, the granularity of content distribution) on the system performance and reliability of DOH with varies (large) configurations.

In our experiments and simulation, the performance is measured as a service time that is the time from receiving a request to sending a reply. A single stand-alone Jetty web-server (without DHT), has been chosen as a baseline. We assume the synthetic *workload* formed of streams of HTTP requests issued by the number of concurrent independent clients, each of which sends a random sequence of requests to retrieve different randomly selected files from different randomly selected sites with a specified intensity. To generate random sequences of requests, we assume the Zipf distribution, as suggested in [8] for the distribution of incoming page requests in the Web.

4.1 Preliminary Evaluation of the DOH Prototype

We used the prototype mostly in order to validate the DOH design. The evaluation testbed included several Pentium 3, 500 MHz computers with 256 MB RAM running Linux (Red Hat 9.3). We report results of two series of experiments. In the first series, 50 files of the mean size of 10Kb of 6 web-sites were uploaded to a DOH-node. In the second series, the content was "heavier": 18 domains, 47 directories, and 503 files (mean size is still 10Kb). The number of nodes varied from 1 to 6 nodes.

As expected, our experiments have shown that the DOH performance is very sensitive to the use of file caches in DOH-nodes when increasing the number of nodes, i.e. increasing the level of distribution of web-sites in DOH. These results suggest that it is worth to make more efforts to find an caching strategy. Evaluation of the prototype has also shown that the performance of DOH heavily depends on the performance of the underlying DKS network: over 90% of the time used by the system consumed by DKS-related activities when the file size is increased to 4Mb.

We have also preliminary evaluated three different strategies of storing files in DHT: file-, directory-, and site-wise - in order to see whether the system performance is sensitive to the strategy used, and which of the strategies is the best with respect to performance. Remind that the three placement strategies use different parts of a URL as a key to determine a DOH-node responsible for the content pointed to by the URL. Unfortunately, the evaluation results for small-scale system configurations show that there is no clear best candidate to use in all cases studied. If published web pages are changing rapidly or if the load of the network is small, then the file-wise approach yields the best results. If there are seldom changes in the stored sites or the load is high, then the directory-wise or even the site-wise approaches are better to use. The system can perform even better if it can support a combination of at least two of the placement strategies, and DKS indeed supports this kind of flexibility.

A full-scale evaluation of the prototype should be done on configurations larger than the setups we could afford for now. In our future work we intend to evaluate large more realistic configurations of the prototype (with large number of nodes and clients). Our future plans also include further performance optimization of the prototype.

4.2 Performance Evaluation using the DOH Simulator

As we continue improving and optimizing performance of the DOH prototype and, in particular, the DKS P2P middleware, we have developed an accurate simulator of DOH in order to evaluate the impact of different design choices (such as the use of caches in nodes, different strategies of storing files in DHT: file-, directory-, and site-wise, prefetching, different replication schemes and the number of replicas) and system changes (nodes leave the network, new nodes join the network) on the overall system performance and reliability. The simulator is based on timing estimates obtained from experiments on the DOH prototype and a stand-alone Jetty web-server.

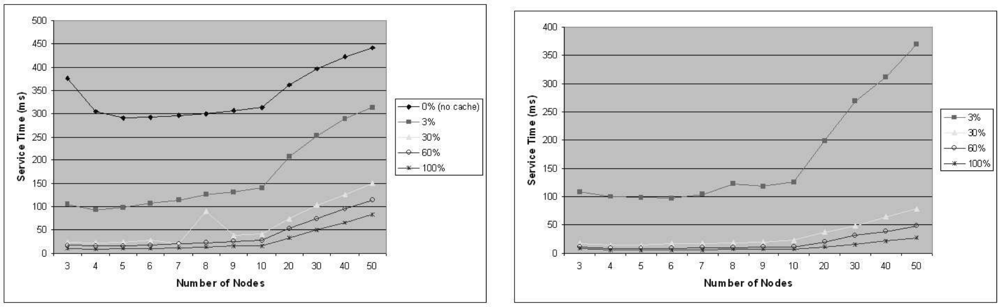
In our simulation, we assume the following workload: a random sequence of requests with the predefined rate (1000-5000 requests/sec) is issued by several clients to retrieve different randomly selected files from different randomly selected sites; the content stored in DOH includes 18 sites, i.e. about 100 directories with about 10 files in each directory; the average file size is assumed to be 30 Kb (concurring with the average file size in the Web, as shown in [5]); the ratio of TTL for files in node caches varies from 0% (no cache) to 100% (always in the cache) of the simulation time. The cache TTL defines how long a file stays in a cache before it's removed from the cache.

The average service time has been computed based on timing estimates obtained from experiments on the DOH prototype with smaller configurations and the Jetty web-server. We assume that there are three major factors that affect the service time: (1) the current load of the server, (2) the size of the requested file, and (3) whether the file is cached or not. The service time was computed as follows:

$$T_s = 2 + \mathit{Load} \times 0.85 + Miss \times (15 + 2 \times fileSize + H \times 100 \times \log_2 N)$$

Here T_s is the service time in ms; *Load* is the number of parallel requests served; $Miss \in 0, 1$ indicates whether the cache misses ($Miss = 1$) or hits ($Miss = 0$); $fileSize$ is the size of the requested file in Kbytes; $H \in 0, 1$ indicates whether the file is stored locally in one of the local DHT buckets ($H = 0$) or remotely ($H = 1$) and a number of hops is required to find and fetch the file from DHT- the probability that $H = 1$ is f/N, where f is the number of replicas per file in DKS; N is the number of nodes. Numeric constants (in ms) in the formula are average times obtained from experiments on the prototype and the stand-alone Jetty web-server.

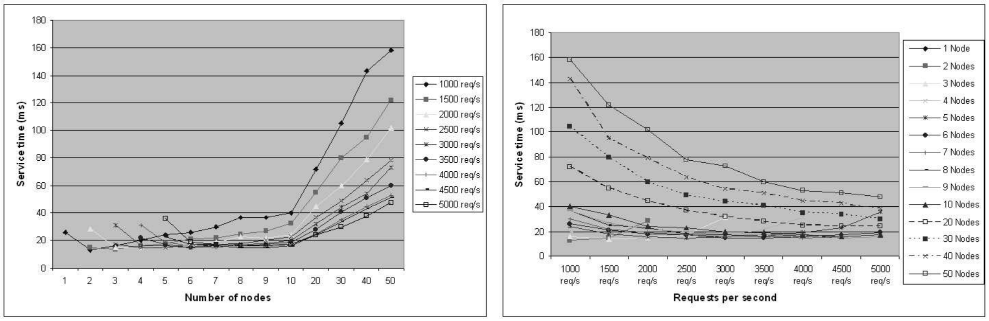
We have evaluated the effect of different design choices on the performance of DOH. Figure 2 shows plots of the service time as the function of the number of nodes for different TTL of cached content and different strategies of placement of content to the distributed hash table of DOH. As expected, it has been observed that the service time is sensitive to the use of caches: the service time is shorter if cached content stays longer in the cache (i.e. the higher cache hit ratio). The service time degrades as the number of nodes increases because of the increase in the DKS lookup latency. However the service time degrades slower when TTL of cached content is high. This result suggests that it is worth to make more efforts to find (more) efficient caching strategies. Plots in Figure 2 also show that DOH with the directory-wise placement (Figure 2 (b)) serves faster than DOH with the file-wise placement (Figure 2 (a)) because the directory-wise placement is used in combination with prefetching: when a file is fetched from DHT the entire directory is prefetched to the cache of the requesting node.



(a) file-wise placement of content in DHT (b) directory-wise placement of content in DHT

Fig. 2: Effect of the use of caches on performance of DOH with different number of nodes, different TTL ratio and different DHT placement strategies. Request rate is 2500 req/sec.

We have compared performance of DOH with different number of nodes and a stand-alone web server (indicated in plots as cases where the number of nodes is 1). Figure 3 shows plots of the service time for different request rates in DOH with the directory-wise content placement and different number of nodes. The TTL of cached content is assumed to be 30 sec. Even though the DOH performance degrades as the number of nodes increases, the service time of DOH with the large number of nodes scales better than the service time of a single web-server for high request rates.



(a) (b)

Fig. 3: Performance of DOH with the directory-wise DHT placement strategy. The cache TTL ratio is 30% of 100 sec of the simulation time (i.e. a file stays in the cache 30 sec)

As expected, in the case of low workload, the DOH network with the small number of nodes performs slower than a stand-alone Jetty server because of an extra overhead introduced by the DKS middleware (implemented in Java), that causes increase of the average service time in DOH as the number of nodes increases. However, as the request rate increases, DOH shows better performance scalability than a stand-alone

Jetty server: the DOH's service time does not increase as fast as the service time of the stand-alone Jetty. For example, at the request rate 600 req/sec (100 parallel requests), the average response time for Jetty is 87ms compared to 110ms for DOH. However at service rate of 1170 req/sec (200 parallel requests), the average response time for Jetty is 171ms compared to 135ms for DOH.

It has been observed that for each request rate there is the certain number of nodes, at which the system shows minimum response time, i.e. there is no improve in service time when increasing the number of DOH nodes beyond a certain value. We believe that this effect depends on the distribution of content in DHT and on how the content is cached in DHT nodes. We intend to study this in our future work.

The simulator suggests that DOH will perform well with service times smaller then 300ms, if the request rate is smaller than approximately 1200 requests per second per node.

Thus, our preliminary evaluation has shown that DOH would be able to handle a flash crowd; however the price the users of the system would have to pay is that the page retrieval under normal workloads would be sligthly slower than in the case of a stand-alone web server.

5 Some Related Work

Many P2P systems, like those proposed in [22], [25], [17], [23], and [26], have been used for creating DHTs. However, the reason for choosing DKS for building our P2P CDN prototype is manifold. There are two main arguments for choosing DKS. First, DKS provides local atomic joins and leaves. By serializing all joins and leaves, DKS guarantees that the DHT will never be in an inconsistent state. Second, DKS uses symmetric replication that allows for concurrent lookups. With symmetric replication, several replicas are placed evenly (symmetrically) in the DHT rather than on a set of consecutive nodes on the structured P2P ring. Symmetric replication allows improving lookup time as well as reliability of the DHT. It also allows the client to get more than one result when doing a lookup. This feature can be used in a voting protocol of the client which needs to be sure that the retrieved object not has been tampered with. To our best knowledge, no other P2P overlay network provides these features of DKS described above.

Systems like Globule[19], SCAN[7] and CoralCDN[12], all propose P2P content delivery networks similar to the one presented in this paper. For example, the authors of Globule[19] make the observation that local web space is cheap, and therefore it could be traded for non-local space, creating replicas on different other servers (called slaves [19]) over the world. However, in Globule, negotiation for the replication space, its configuration and management are not handled automatically but rather by a human administrator, whereas DOH is autonomous and has the ability to self-organize when a node joins/leaves the system. SCAN, which is a P2P CDN proposed in [7], uses Tapestry[26] as an underlying P2P network. One of the main goals of SCAN is to keep the number of replicas at a minimum to reduce overhead. This may cause sites to be unavailable whenever the master copy is unavailable for some time. CoralCDN[12] uses the Coral[13] implementation of a Distributed Sloppy Hash Table to keep references to the master copy (or valid cached copies) on different nodes. In CoralCDN, as in SCAN, if the master copy of a site becomes unavailable for Coral, the site will soon become unreachable. In contrast to SCAN and CoralCDN, DOH presented in this paper uses symmetric replication provided by DKS[15] to improve availability of hosted web-sites. Furthermore, in CoralCDN the URLs needs to be "coralized"[3] to be a part of the overlay network, i.e. CoralCDN is not even initally transparent to the end-users, which is one of the major design goals of DOH. PAST[24] is a storage utility built on top of Pastry[23] and shares many of the key ideas with the DOH system presented in this paper. However, as stated in [24] "PAST does not provide facilities for searching, directory lookup, or key distribution", which are features already implemented in our DOH prototype. Open Content Network (OCN) [18] exemplifies an effort to build a P2P CDN. OCN extends the HTTP protocol to create a Content-Addressable Web and, unlike DOH, uses a browser plug-in for its clients to recognize it. The authors claim that OCN is a P2P CDN, however it is more traditional P2P file-sharing system than a typical CDN because clients are also involved in content distribution. OCN is designed for handling large files and uses the clients extensively to distribute files, as most of existing P2P file-sharing systems do. DotSlash[27] is described by the authors as being a rescue system for web servers during hotspots. The authors of DotSlash do share the same motivation for developing a P2P CDN as in this paper: to help web servers survive a flash crowd. DotSlash does not store content globally (i.e. does not distribute and/or replicate a web-site among nodes as it is done in DOH) but all servers will store their own content. When a flash crowd occurs, an overlay network with rescue servers will be created, and the "hot objects" will be cached at these servers during the flash crowd. This network will be abandoned when workloads are back to normal. In contrast to DotSlash, DOH allows to distribute content of a web-site and its replicas among nodes making the web site more available for intensive concurrent requests.

6 Conclusions and Future Work

In this paper we have presented an approach to building a content delivery network as a structured P2P system of web-servers. This approach allows improving availability and scalability of a web site due to the load distribution, multiple access points, and replication. Such content delivery P2P network of collaborative web-servers can be used as a Web-hosting environment to host several web-sites. This approach can also be considered as one inexpensive solution to the problem of a flash crowd[1].

To validate, and preliminary evaluate our approach we have developed a system prototype called DOH (DKS Hosting system) based on the DKS P2P middleware that integrate the Jetty web-servers in a scalable content delivery network (web-hosting system). Each node in the DOH network is a DKS-node, i.e. is a part of the DKS overlay network, has a web server (to retrieve files) and an FTP server (to download/upload files). The network also includes Translators which are client contact points to the DOH network. Translators are used to distributed requests among DOH nodes in order to achieve load balancing. A Translator redirects web clients to DOH nodes based on the nodes' load and network congestion. Each Translator maintains a cache of information about nodes (including their load and RTT times) that are known to be in the system. The Translator-cache is also used to help a content provider to find nodes to login to, when uploading content using FTP.

[3] See [12]

DOH stores files in a Distributed Hash Table (DHT) provided by DKS so each node is able to retrieve the requested files from the DHT and cache them locally for future requests. Thus when a sudden traffic surge occurs, there will not only be one server (DHT-node) serving all the requests but a network of cooperating web servers helping each other by dividing the load.

We have evaluated the prototype for small-scale DOH configurations. To preliminary evaluate medium- and large-scale setup we have developed a DOH simulator based on timing estimates obtained from the performance experiments on the prototype. Evaluation results show that DOH performs better than a stand-alone webserver in the case of the high request rate (a large number of simultaneous requests) and the response time in DOH scales better than the response time in a single webserver, so the approach is valid for solving the intended problem of a flash crowd. However, as expected, with a low request rate, the single web-server outperforms DOH. Experiments also show that performance of DOH is very sensitive to the use of caches. We can also expect that explicit replication supported in DKS will improve performance of the prototype.

Two scenarios of performance has thus been identified: during low and high request rate, and both needs to be addressed in our future work which also includes more detailed performance evaluation, including assessment of impact (if any) of replication on service time; improving the caching mechanism in order to improve system performance; extending the system design to support dynamic contents (web-based applications). There are many issues to be considered when extending the system for dynamic contents, e.g. how to deploy and replicate applications, how to handle transactions, states, and failures; how to store the state to achieve failover. We leave answering all these questions to our future work.

References

1. Adler, S.: The Slashdot Effect: An Analysis of Three Internet Publications [Online] http://ssadler.phy.bnl.gov/adler/SDE/SlashDotEffect.html (2005)
2. Akamai Technologies, Inc. [Online] http://www.akamai.com/
3. Alima, L.O.,El-Ansary, S., Brand, P., Haridi, S.: DKS(N, k, f): A Family of Low Communication, Scalable and Fault-Tolerant Infrastructures for P2P Applications, In *The 3rd Int workshop CCGRID2003*, Tokyo, Japan, (2003)
4. Apache License, Version 2.0 [Online] http://www.apache.org/licenses/LICENSE-2.0
5. Arlitt, M. F., Williamson, C. L.: Internet web servers: Workload characterization and performance implications. In *IEEE/ACM Trans on Networking*, **5(5)** (1997) 631-645
6. R. Bhattacharyya [Online] http://www.mycgiserver.com/r̃anab/ftp/ (2005)
7. Chen, Y., Katz, R., Kubiatowicz, J.: SCAN: A dynamic, scalable, and efficient content distribution network. *Proc of Int Conf on Pervasive Computing*, Zurich (2002)
8. Crovella, M. E., Taqqu, M. S., Bestavros, A.: Heavy-tailed probability distributions in the World Wide Web. In *A Pract. Guide To Heavy Tails*, Chapman & Hall (1998) 3-26
9. Distributed K-ary System (DKS), [Online] http://dks.sics.se/
10. Eastlake, D., Jones, P.: US Secure Hash Algorithm 1 (SHA1), RFC 3174, 2001. [Online] http://www.ietf.org/rfc/rfc3174.txt
11. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext Transfer Protocol - HTTP 1.1, RFC2616, 1999. [Online] http://www.ietf.org/rfc/rfc2616.txt

12. Freedman, M. J., Freudenthal, E., and Mazi‘eres, D: Democratizing Content Publication with Coral. In *Proc of the 1st Symp on Networked Systems Design and Implementation (NSDI 2004)*, San Francisco, USA (2004)
13. Freedman M., Mazi‘eres, D.: Sloppy hashing and self-organizing clusters. In *2nd Int Peer To Peer Systems Workshop*, Berkeley, USA (2003)
14. Jetty Java HTTP Servlet Server [Online] http://jetty.mortbay.org/jetty/index.html
15. Ghodsi, A., Alima, L.O., Haridi, S.: Symmetric Replication for Structured Peer-to-Peer Systems. In *The 3rd Int Workshop on Databases, Information Systems and Peer-to-Peer Computing*, Trondheim, Norway (2005)
16. Iyer, S., Rowstron, A., Druschel, P.: Squirrel: A decentralized, peer-to-peer web cache. In *Proc of the 21st Ann ACM Symp on Principles of Distributed Computing*, ACM (2002)
17. Maymounkov, P., Mazi‘eres, D.: Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS02*, Cambridge, MA (2002)
18. Open Content Network, [Online]http://open-content.net/ (2005)
19. Pierre, G., van Steen, M.: Design and implementation of a user-centered content delivery network. In *Proc. 3rd Workshop on Internet Applications*, San Jose, USA (2003)
20. Pierre, G., van Steen, M., Tanenbaum, A. S.: Dynamically selecting optimal distribution strategies for Web documents. *IEEE Trans on Computers*, **51(6)** (2002) 637–651
21. Postel, J., Reynolds, J.K.: File Transfer Protocol, RFC959, Oct 1985. [Online] http://www.ietf.org/rfc/rfc959.txt
22. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A scalable content-addressable network, In *Proc of the 2001 Conf on Applications, Technologies, Architectures, and Protocols for Computer Communications*, San Diego, USA (2001) 161-172
23. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Int Conf on Distr Systems Platforms (Middleware)* (2001) 329-350
24. Rowstron, A., Druschel, P.: Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility, In *Proc of the 18-th ACM Symposium on Operating Systems Principles* (2001)
25. Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, In *Conf on Applications, Technologies, Architectures, and Protocols for Comp. Communications* (2001) 149-160
26. Zhao, B. Y., Kubiatowicz, J. D., Joseph, A. D.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. TR UCB/CSD-01-1141, UC Berkeley (2001)
27. Zhao, W., Schulzrinne, H.: DotSlash: A selfconfiguring and scalable rescue system for handling web hotspots effectively. In *Int Workshop on Web Caching and Content Distribution (WCW)*, Beijing, China (2004)