Distributed Content Delivery using Load-Aware Network Coordinates

Nicholas Ball
Imperial College London
United Kingdom
nsb04@doc.ic.ac.uk

Peter Pietzuch
Imperial College London
United Kingdom
prp@doc.ic.ac.uk

ABSTRACT

To scale to millions of Internet users with good performance, content delivery networks (CDNs) must balance requests between content servers while assigning clients to nearby servers. In this paper, we describe a new CDN design that associates synthetic load-aware coordinates with clients and content servers and uses them to direct content requests to cached content. This approach helps achieve good performance when request workloads and resource availability in the CDN are dynamic. A deployment and evaluation of our system on PlanetLab demonstrates how it achieves low request times with high cache hit ratios when compared to other CDN approaches.

1. INTRODUCTION

The average size of web objects has grown over the years. Today Internet users regularly download rented films (*e.g.,* from iTunes and NetFlix), TV programmes (*e.g.,* using BBC iPlayer), large security updates, virtual machine images and entire operating system distributions. File sizes for this content can range from a few megabytes (security patches) to several gigabytes (rented high-definition films). Content providers use *content delivery networks* (CDNs), such as Akamai [16], Limelight, CoralCDN [3] and CoDeeN [17], to provide files to millions of Internet users through a distributed network of content servers.

To achieve a scalable and reliable service, a CDN should have two desirable properties. First, *load awareness* should partition requests across a group of servers with replicated content, balancing computational load and network congestion. This increases the number of users that can be served requesting the same content. The degree of replication may be chosen dynamically to handle surges of incoming requests (flash crowds) when content suddenly becomes popular (known as the Slashdot effect) [2]. Second, *locality awareness* should exploit proximity relationships in the network, such as geographic distance, round-trip latency or topological distance, when assigning client requests to content servers. Intuitively, by keeping network paths short, a CDN can provide better quality-of-service (QoS) to clients.

It is challenging to design a CDN that makes a trade-off between load- and locality-awareness. Simple CDNs that always redirect clients to geographically-closest content servers lack load-balancing and suffer from overload when local content popularity increases. More advanced CDNs that are based on *distributed hash tables* (DHTs) [15] primarily focus on load-balancing. They use network locality only as a tie-breaker between multiple servers, leading to sub-optimal decisions about network locality. State-of-the-art CDNs such as Akamai [16] are proprietary, with little public knowledge on the types of complex optimisations that they perform.

In this paper, we describe a new type of CDN that uses *load-aware network coordinates* (LANCs) to capture naturally the tension between load and locality awareness. LANCs are synthetic coordinates (calculated using a metric embedding [10] of application-level delay measurements) that incorporate network location and load of content servers. Our CDN uses LANCs to map clients dynamically to the most appropriate server in a decentralised fashion. Popular content is replicated among nearby content servers with low load. By combining locality and load using the unified mechanism of LANCs, we simplify the design of the CDN and discard the need for ad-hoc solutions.

The main contributions of this work are: (1) the introduction of LANCs, showing how they react to CPU load and network congestion; (2) the design and implementation of a CDN that uses LANCs to route client requests to content servers and replicate content; (3) a preliminary evaluation of our LANC-based CDN on PlanetLab, demonstrating its benefits in comparison to other approaches. Our results shows that, with a locality-heavy workload, our approach achieves almost an order of magnitude lower request times compared to direct retrieval of content.

The rest of the paper is structured as follows. In Section 2, we discuss the requirements for CDNs in more detail and, in Section 3, we describe LANCs. We present our CDN design in Section 4, focussing on request mapping and request redirection. In Section 5, we evaluate LANCs on a local network and also present results from a large-scale PlanetLab deployment. Section 6 describes related work and Section 7 concludes with an outlook on future work.

2. CONTENT DELIVERY NETWORKS

Global CDNs are typically implemented as a distributed network of servers hosting replicated content. Servers may be located at geographically-diverse data centres, providing

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ROADS '08
Copyright 2008 ACM 978-1-60558-266-5/08/0012 ...$5.00.

content to users in a given region. Web clients issue requests to fetch content from the CDN, which are satisfied by content servers according to a particular mapping strategy. A goal of the CDN is to minimise total request time, *i.e.,* the time until a client has successfully retrieved desired content.

Load awareness. For a CDN to be *load-aware*, there must be a mechanism that balances load between content servers. This may be the done by, for example, distributing requests uniformly across all content servers or redirecting requests to the least-loaded server. If content requests go to overloaded servers, clients experience poor performance.

In our model, we distinguish between load balancing for *computational load* and *network congestion*. (1) Computational load at a content server is caused by the processing associated with content requests. This involves request parsing, retrieval of cached content if there is a cache hit and content delivery to the client. (2) Network congestion is caused by the limited capacity of network paths to carry traffic. This creates bottleneck links that determine the throughput at which content can be delivered to clients. Considering the Internet path from client to server, the bottleneck may be: (a) at the client's access link, in which case only the performance of this client is affected and the CDN cannot provide any remedy; (b) at the client's upstream ISP or Internet backbone, affecting a larger number of clients. Here, the choice of a different content server may lead to improved performance; (c) at the content servers' access link, affecting all client requests to this server. In this case, the CDN should redirect requests away from the congested server.

Locality awareness. A CDN is *locality-aware* if network paths are kept short. For example, a CDN can take request origins into account and return content from nearby servers with low load. Proximity may be defined in terms of geographic distance, latency, number of routing hops or overlap between address prefixes. By minimising network path lengths, clients are more likely to experience better QoS [11]. Intuitively, this is because: (1) short paths offer low latencies. This helps TCP obtain high throughput quickly therefore reducing transmission times for small content; (2) short paths are less likely to encounter congestion hot-spots, resulting in improved throughput; (3) short paths tend to be more reliable as they involve fewer network links and routers; (4) short paths decrease overall network saturation, leaving more spare network capacity for other traffic.

3. LOAD-AWARE NETWORK COORDINATES

Overlay networks can use *network coordinates* (NCs) [10] to become locality-aware [13]. In a NC system, each node maintains a synthetic n-dimensional coordinate (of low dimensionality, typically $2 \leq n \leq 5$) based on round-trip latency measurements between nodes. The NCs of nodes are calculated by embedding the graph of latency measurements into a metric space. Euclidean distances between NCs then predicts the actual communication latencies. NCs are updated dynamically to reflect changes in Internet latencies.

A benefit of NCs is that they estimate missing measurements. They allow Internet nodes to reason about their relative proximities without having to collect all measurements. However, triangle inequality violations found in Internet latencies make it impossible to embed measurements without error [9], resulting in less accurate latency prediction. There is also a constant measurement overhead when maintaining NCs in the background.

Vivaldi. Our CDN uses the *Pyxida* library [14] to maintain NCs according to the *Vivaldi* [1] algorithm. Vivaldi is a decentralised algorithm that computes NCs using a spring-relaxation technique. Nodes are modelled as point masses and latencies as springs between nodes. The NCs of nodes change as nodes attract and repel each other.

$$E = \sum_i \sum_j (L_{ij} - \|x_i - x_j\|)^2 \quad (1)$$

Let x_i be the NC assigned to node i and x_j to j. L_{ij} is the actual latency between nodes i and j. Vivaldi characterises the errors in the NCs using the squared-error function in Eq. 1, where $\|x_i - x_j\|$ is the Euclidean distance between the NCs of nodes i and j. Since Eq. 1 corresponds to energy in a physical mass-spring system, Vivaldi minimises prediction errors by finding a low-energy state of the spring network.

NCs are computed by each node in a decentralised fashion: When node i receives a new latency measurement L_{ij} to node j, it compares the true latency L_{ij} to the predicted latency $\|x_i - x_j\|$. It then adjusts its NC x_i to minimise the prediction error according to Eq. 1. Measurements are filtered to remove outliers [7]. The above process converges quickly in practice and leads to stable and accurate NCs [6].

Load-awareness. It is easy to see how a CDN could benefit from NCs to achieve locality-awareness. Assuming that clients and servers have known NCs, clients could direct requests to the server with the closest NC. Also, servers could use their neighbours to replicate popular content, keeping content replication local while reducing total request times.

The lower accuracy of NCs for latency prediction compared to direct measurements [18] is less important for a CDN. A small latency reduction by choosing a slightly closer server will only have a marginal impact on the total request time for large content. The goal for locality-awareness in a CDN is to select a content server with good performance, as opposed to finding the single closest one.

However, regular NCs do not provide load-balancing between content servers. Servers in densely-populated areas are likely to become overloaded, whereas servers in sparse regions will have spare capacity. To address this problem, we propose *load-aware NCs* (LANCs) that are calculated using application-level delay measurements, therefore incorporating computational load and network congestion.

Regular NCs are computed from network-level measurements (*e.g.,* ICMP echo requests) to capture pure network latencies between two hosts. In particular, measurements aim to be independent of computational load (*e.g.,* by assigning kernel-space timestamps to packets) and of congestion on network paths (*e.g.,* by using small measurement packets least affected by congestion) [5].

In contrast, LANCs are computed with application-perceived round-trip times (RTTs) between hosts that are measured through dedicated TCP connections with user-space timestamping. The computational load of a host affects user-space timestamps, leading to higher RTTs for overloaded hosts. Congestion on network paths results in the retransmission of TCP packets, again increasing application-perceived RTTs. As a result, overloaded and congested servers are assigned more distant LANCs and can be avoided by clients. (This assumes that clients are, in general, less loaded than servers.)

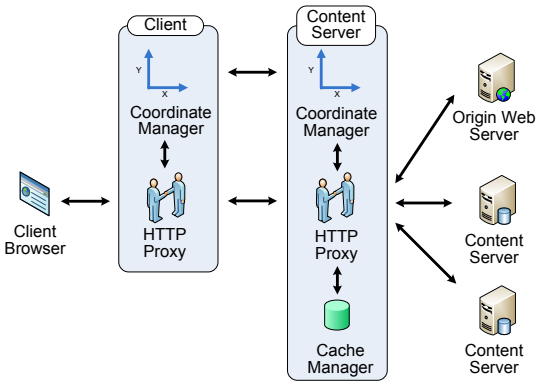

Figure 1: Overview of the architecture of our LANC-based CDN.

By folding load into LANCs, the CDN does not need to manually tune the trade-off between choosing a local content server versus a server with low load. Instead, it can map client requests directly to the "best" content server in terms of expected content delivery performance. The best server has the closest LANC relative to the client and combines low computational load with little congestion on the network path. We believe that this results in a simpler and more natural CDN design.

4. CDN DESIGN

Next we describe the architecture of our LANC-based CDN, how it processes requests, and how requests are redirected between servers. As shown in Figure 1, we distinguish between the *content server*, the *client* and the *origin web server*. Clients generate requests and forward them to content servers. Servers receive requests and handle them by delivering the requested content, potentially fetching an authoritative copy of the content from the origin web server.

Content servers have three components: (1) The *cache manager* provides an interface to store and retrieve content from the local disk. It also defines the cache replacement strategy (*e.g.,* LRU or LFU) when disk space is low and content must be discarded.

(2) The *server HTTP proxy* is the point of entry for HTTP requests by providing a proxy interface. For each request, the proxy decides to (a) retrieve content locally if the cache manager indicates that the requested content exists in the local cache; (b) request content from a nearby server with the help of the server coordinate manager (Section 4.2); or (c) return content from the origin web server to the requester, while caching it locally for future access.

(3) The *server coordinate manager* maintains the LANC. It takes application-level delay measurements to random other servers and clients and updates the LANC accordingly. It also maintains a routing table of neighbours that is used for mapping clients to servers (Section 4.1).

Clients generate HTTP requests with a regular web browser. To redirect requests to content servers, clients run two components as a separate process (or as part of a browser plug-in): (1) The *client HTTP proxy* provides a local proxy. The local browser is configured to send all HTTP requests through this proxy. The proxy interacts with the client coordinate manager to redirect requests to content servers.

(2) The *client coordinate manager* is similar to the one on the server-side. It makes delay measurements to maintain

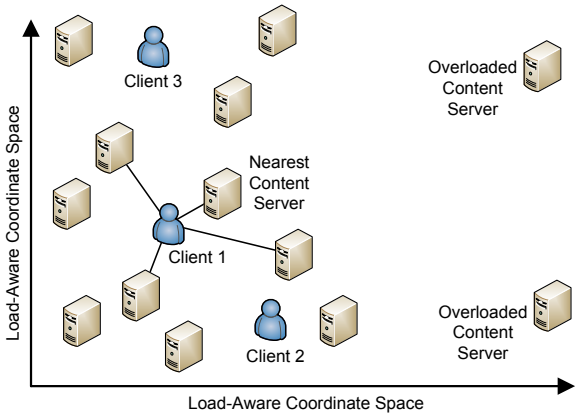

Figure 2: Mapping of requests to content servers using LANCs

a LANC. It also manages a fixed-sized neighbour set with nearby content servers used for mapping requests to servers.

4.1 Request Mapping

Clients map requests to the content server with the closest LANC compared to their own. This guarantees that they use a server with good locality and low load. As shown in Figure 2, each server maintains a *neighbour set* of nearby servers. Requests are redirected to the closest server from that set. Overloaded content servers will move away from other clients by settling for "distant" coordinates in the space.

Each client coordinate manager must dynamically keep track of a small set of nearby servers. For this, we use a geometric routing approach in the coordinate space created by the LANCs of all servers. The algorithm, described in previous work [6], constructs a Θ-graph spanner and uses a greedy approach to route messages to a target coordinate through $O(\log n)$ hops. Informally, each server has a routing table that stores $O(\log n)$ servers at various angles and distances in the LANC space. To route a message to a given target LANC, a node greedily forwards the message to a node from its routing table that is closest to the target. A more detailed description can be found in [6].

When a new client joins the CDN, it contacts an arbitrary server and routes a message with its own LANC as the destination. This message is guaranteed to arrive at the closest existing server to the given coordinate. The client then adds this server to its neighbour set. It may also include other, more distant servers to increase failure resilience. Periodically, clients rejoin the CDN to update the mapping as the LANCs of servers change. Servers follow the regular join protocol described in [6] to construct their routing tables.

Note that multi-hop geometric routing is only done when a client joins the CDN (and also periodically to refresh the mapping) but not for each content request. A content request only requires a look-up of the best content server from the local neighbour set.

4.2 Request Redirection

So far, our LANC-based CDN is only populated with new content when a server fetches content from the origin web server after a cache miss **(local-only redirection)**. However, we could take advantage of the proximity between servers and have them cooperate to retrieve content from each other. This would help exploit *spatial* in addition to

just *temporal* cache locality. For a performance gain, we must ensure that it is faster to retrieve content from another server than to fetch it from the original web server. This is likely to be true for local servers. We propose two simple coordinate-based techniques for cooperation between content servers:

Server-centric redirection. When a *primary* content server receives a request that it cannot satisfy with its local cache, it forwards the request to a set of *secondary* servers in parallel. The secondary servers are neighbours in its geometric routing table within a distance d in the LANC space. The secondary servers then (a) return the requested content to the primary server if available. The primary server then forwards the content to the client while caching it; or (b) respond with cache misses. If all secondary servers had cache misses, the primary server fetches the content from the origin web server. A low value of d ensures that retrieving content from secondary servers gives better performance than fetching it directly from the origin web server. A timeout value t puts an upper bound on the waiting time for responses from secondary servers.

Client-centric redirection. In this scheme, a client sends requests in parallel to nearby servers in the LANC space. Again, the closest server will act as the primary server and fetch the content from the origin web server on a cache miss. At the same time, secondary servers within distance d attempt to retrieve the content from their local caches and return it to the client. The client may receive content multiple times from different servers and therefore needs to abort pending requests after the first successful retrieval.

Client-centric redirection is likely to give better performance when fetching content from secondary servers. However, it forces the primary server to retrieve content from the origin server, even when a secondary server has had a cache hit. This unnecessarily increases load on origin web servers. To avoid this (at the cost of one additional RTT), the client could send the request to the primary server only *after* receiving negative responses from the secondary ones.

Fundamentally, server-centric redirection spreads popular content more effectively in the CDN because primary servers retrieve it from secondary servers without involving origin web servers. Therefore our evaluation in Section 5.2 focuses on server-centric redirection.

5. EVALUATION

Our evaluation goals were (a) to investigate the behaviour of LANCs under CPU load and network congestion in a controlled environment; and (b) to observe the performance of a large-scale deployment of our LANC-based CDN with a non-trivial workload on PlanetLab.

5.1 LAN Experiment

The first experiment examines how LANCs are influenced by computational load on hosts. We set up our CDN on a LAN with 6 nodes (1 content server and 5 clients). The content server ran on a resource-constraint laptop connected via a wireless link and acted as the bootstrap node. To achieve high sensitivity to load changes, all nodes performed aggressive RTT measurements every 10 seconds with 100 KB payloads. The nodes were left until their LANCs stabilised. We then generated synthetic load on the server with a CPU-bound process for 10 minutes.

Figure 3 shows the application-level RTT samples from

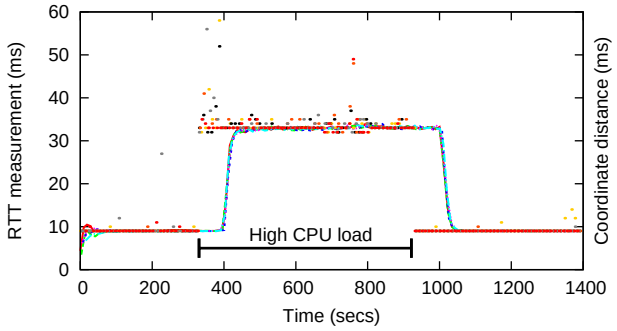

Figure 3: Increase in RTT and coordinate distances under high content server load

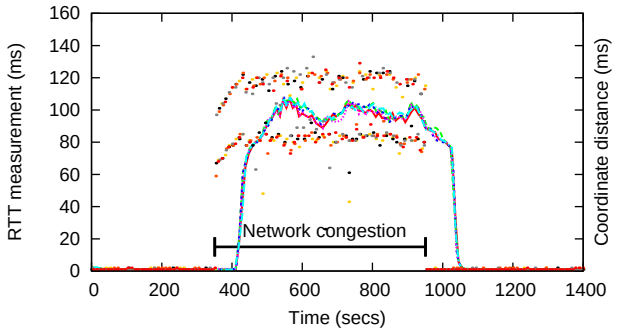

Figure 4: Increase in RTT samples and coordinate distances under network congestion

the five clients to the server (shown as dots; left axis) and the corresponding coordinate distances (shown as lines; right axis) between the clients and the server. The CPU load on the content server causes an increase in RTT, which is then (with a small lag) also reflect by the LANC distances. The change in LANC distances is due to a change of the server coordinate. Clients observe that the RTTs among them have not changed, whereas the server measures higher RTTs to all clients and adapts its coordinate accordingly.

In the second experiment, we ran 1 (well-provisioned) content server and 5 clients in a LAN. After a stabilisation period, we added 3 elastic TCP flows between the node running the content server and 3 other (non-client) nodes. As shown in Figure 4, the resulting network congestion increases the RTT samples (dots; left axis) between the clients and server and makes them vary between 80 and 120 ms. This is picked by the LANC coordinate of the server, leading to increased distances (lines; right axis). After the additional flows terminate, the coordinate returns to its original value.

5.2 PlanetLab Deployment

We deployed our LANC-based CDN on PlanetLab (PL) to investigate how it exploits temporal and spatial locality between client requests. We ran content servers on 108 PL nodes distributed world-wide and 16 clients on hosts at our university. We then conducted six experiments with different configurations and measured the performance of satisfying content requests:

(a) **LANC+SC (server-centric).** This configuration uses the LANC-based request mapping with server-centric redirection of requests. On a cache miss, a content server forwards a request to all its known neighbours and waits

	a) LANC+SC	b) LANC+LO	c) Nearest	d) Random	e) Direct	f) CoralCDN
Total number of requests	12,922	8524	1204	1089	3331	4074
Successful requests	97.8%	97.8%	96.3%	96.5%	80.6%	75.8%
Total transfer volume (GB)	40.47	26.69	3.72	3.37	8.60	9.90
Cache hit ratio	97.3%	89.2%	90.2%	4.6%	N/A	Unknown
Avg. request time (secs)	3.20	4.83	35.84	50.33	20.40	18.41
Median request time (secs)	2.24	1.76	34.84	15.97	3.45	4.01
90th perc. request time (secs)	4.57	4.97	41.81	117.58	28.26	19.55
Avg. transfer rate (KB/s)	1,533	1,935	106	305	1,778	876
Median transfer rate (KB/s)	1,500	1,912	96	210	973	741
90th perc. transfer rate (KB/s)	2,448	2,726	142	756	4,448	1,739

Table 1: Summary of CDN performance on PlanetLab with six configurations

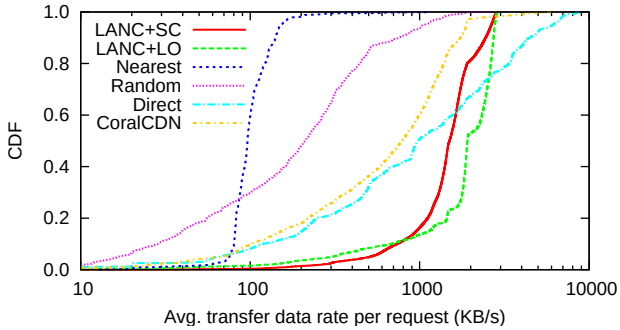

Figure 5: CDF plot of the distribution of transfer data rates for six configurations. (Faster is better)

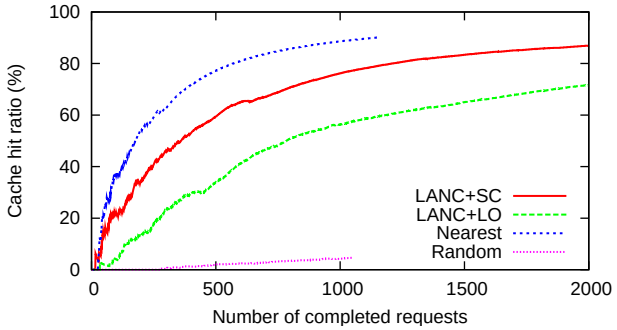

Figure 6: Cache hit ratio as a function of completed requests (for first 2000 requests only)

for responses with a 2 second timeout.

(b) **LANC+LO (local-only).** This uses the LANC-based request mapping but only satisfies requests from the local server cache.

(c) **Nearest.** This directs all requests to the single, nearest content server (located at one of the Imperial PL nodes).

(d) **Random.** This directs requests to random servers.

(e) **Direct.** For comparison, this retrieves content directly from the origin web servers without caching.

(f) **CoralCDN.** This configuration directs all requests to the local CoralCDN node running on PL.

Each experiment was set up in the same way. The first content server acted as the bootstrap node for all other nodes. After starting the servers, we added the client nodes and let coordinates stabilise for one hour. RTT measurements between nodes were taken with 4 KB payloads exchanged through TCP connections every minute. All content servers started with cold caches.

We generated a synthetic request workload on the 16 clients. Each client continuously requested a Gentoo Linux distribution file with a size of 3.28 MB from a list of 100 web servers distributed world-wide [4]. (Since the URLs were different, all CDNs assumed these to be different files, with no caching between URLs.) This request load provided spatial locality (all requests came from clients at our university) and temporal locality (clients eventually requested the same URL repeatedly). The high load on PL nodes exercised our load-balancing mechanism. We deliberately chose a relatively small file size to stay below per-slice transmission limits on PL. Each configuration ran for one hour.

The results are summarised in Table 1. Figure 5 shows the distribution of average transfer data rates per request across all clients on a logarithmic scale.

LANC+SC manages to satisfy the most requests (12,922) in one hour. The low value for the 90th percentile of request times (4.57 seconds) is due to its good choice of content servers (with nearby LANCs), which results in high transfer rates, and its aggressive fetching of cached content from other servers, which means a high cache hit ratio of 97.3%. During the experiment, we observed that most clients requested content from a small set of servers running on PL nodes with low load at Imperial, in France and in Germany.

LANC+LO satisfies fewer requests compared to LANC+SC. As seen in Figure 5, its performance suffers from a small fraction of requests with low transfer rates. We observed that these were mostly cache misses that forced the retrieval of content from slow web servers. At the same time, the 90th percentile of its transfer rate is slightly higher than for LANC+SC because servers do not have the overhead of relaying content when fetching it from neighbours. Since LANC+LO only considers a single content server for satisfying requests, it has a lower cache hit ratio (89.2%).

Nearest gives poor performance because the single content server becomes a bottleneck and can only satisfy requests with a consistently low transfer rate. It has a high cache hit ratio (90.2%) because all requests are directed to the same server (but not the highest because clients do not manage to submit all requests twice within one hour).

Random balances requests across many servers but fails to exploit spatial or temporal locality, leading to a low cache hit ratio (4.6%). Selected servers often take a long time to retrieve content due to high load on PL nodes and/or low available bandwidth.

Direct illustrates the benefit of caching with a CDN compared to retrieving content directly from hosting web servers.

Finally, we used **CoralCDN** on PL to retrieve content. Although this is not a fair comparison because, as a public service, CoralCDN handles a higher work-load than just the requests we directed at it, we believe that it illustrates the potential of our approach. As expected, CoralCDN showed

a substantially higher average request time (18.41 seconds) than our LANC-based CDN. As shown in Figure 5, its average transfer rates were lower than fetching content directly (while presumably significantly reducing load on the origin web servers). For now, we can only speculate whether this is due to CoralCDN's high workload or less optimal choice of content servers. In future work, we plan to repeat this experiment in a controlled setting.

Figure 6 illustrates the difference between LANC+SC and LANC+LO in more detail. It shows the change in cache hit ratio as a function of completed requests. The cache hit ratio of LANC+SC grows faster because it considers nearby servers for requested content. This means that LANC+SC reduces the load on the origin web servers by retrieving more content from the CDN. With our workload, eventually all requests can be satisfied by the CDN and the cache hit ratio asymptotically approaches unity.

6. RELATED WORK

CoralCDN [3] is a peer-to-peer CDN built to help web servers handle huge demands of flash crowds. With cooperating cache nodes, CoralCDN minimises the load on the original web server and avoids the creation of hot spots. It builds a load-balanced, latency-optimised hierarchical indexing infrastructure based on a weakly-consistent DHT and achieves locality-awareness by making nodes members of multiple DHTs called clusters. Clusters are specified by a maximum RTT and can reduce request latencies by prioritising nearby nodes. Our work shares many of the design goals of CoralCDN in terms of locality- and load-awareness. However our approach of using a unified scheme based on LANCs is different, aiming for good request mappings in environments with dynamic load and network congestion.

CoDeeN [17] is a network of open proxy servers running on PL. The system leaves it up to the user to choose a suitable proxy server for browser requests. If the local proxy cannot satisfy a request, it selects another proxy based on locality, load and reliability. Our LANC-based CDN automates the initial mapping step at the cost of running a client component on user machines.

Globule [12] is a collaborative CDN that exploit client resources for caching using a browser plug-in. Because Globule runs on client hosts, it must make different assumptions about churn and security. It uses landmark-based NCs for locality-awareness. Replicated content is placed according to the locations of clients in a coordinate space. Since it leverages many client machines for caching, balancing computational load is less important.

Meridian [18] builds an overlay network for locality-aware node selection with on-demand probing to estimate node distances. To find the nearest node to a client, Meridian uses a set of ICMP echo requests to move logarithmically closer to the target. Similar to our geometric routing tables, each nodes maintains a fixed number of links to other nodes with exponentially increasing distances. Meridian aims to return the closest existing node to a client. We argue that such accuracy is not necessary when selecting content servers, as other factors such as load will be equally important.

7. CONCLUSIONS

We have described a novel CDN that uses a coordinate space with application-level latency measurements between clients and content servers for the mapping and redirection of requests. We demonstrated how our LANC-based CDN reacts to computational load and network congestion. A large-scale deployment on PL with a locality-heavy workload highlights the benefits of our approach.

As future work, we plan to study the stability and load-awareness of LANCs under dynamic workloads (*e.g.,* flash crowds) and varying resource availability. We also plan a more extensive comparison against other CDNs. Finally, we want to investigate how proxy NCs [8] can relieve the burden from maintaining LANCs from client nodes.

8. ACKNOWLEDGEMENTS

We would like to thank Richard Clegg, Jonathan Ledlie, Cristian Lumezanu, Miguel Rio and the anonymous reviewers for their feedback on this paper.

9. REFERENCES

[1] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proc. of SIGCOMM*, Aug. 2004.

[2] J. Elson and J. Howell. Handling Flash Crowds from Your Garage. In *Proc. of USENIX*, 2008.

[3] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing Content Publication with Coral. In *Proc. of NSDI*, 2004.

[4] Gentoo. Gentoo Mirror List. `www.gentoo.org/main/en/mirrors2.xml`, Aug. 2008.

[5] J. Ledlie, P. Gardner, and M. Seltzer. Network Coordinates in the Wild. In *Proc. of NSDI*, 2007.

[6] J. Ledlie, M. Mitzenmacher, M. Seltzer, and P. Pietzuch. Wired Geometric Routing. In *Proc. of IPTPS*, Bellevue, WA, USA, Feb. 2007.

[7] J. Ledlie, P. Pietzuch, and M. Seltzer. Stable and Accurate Network Coordinates. In *ICDCS*, July 2006.

[8] J. Ledlie, M. Seltzer, and P. Pietzuch. Proxy Network Coordinates. TR 2008/4, Imperial College, Feb. 2008.

[9] E. K. Lua, T. Griffin, et al. On the Accuracy of Embeddings for Internet Coord. Sys. In *IMC*, 2005.

[10] T. E. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proc. of INFOCOM'02*, New York, NY, June 2002.

[11] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Distributed Resource Discovery on PlanetLab with SWORD. In *WORLDS*, Dec. 2004.

[12] G. Pierre and M. van Steen. Globule: a Collaborative CDN. *IEEE Comms. Magazine*, 44(8), Aug. 2006.

[13] P. Pietzuch, J. Ledlie, M. Mitzenmacher, and M. Seltzer. Network-Aware Overlays with Network Coordinates. In *Proc. of IWDDS*, July 2006.

[14] Pyxida Project. `pyxida.sourceforge.net`, 2006.

[15] I. Stoica, R. Morris, D. Karger, et al. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. SIGCOMM*, Aug. 2001.

[16] A.-J. Su, D. Choffnes, A. Kuzmanovic, et al. Drafting Behind Akamai. In *SIGCOMM*, 2006.

[17] L. Wang, K. Park, R. Pang, V. S. Pai, and L. Peterson. Reliability and Security in the CoDeeN Content Distribution Network. In *USENIX*, 2004.

[18] B. Wong et al. Meridian: A Lightweight Network Loc. Service without Virtual Coords. In *SIGCOMM*, 2005.