SCAN: A Dynamic, Scalable, and Efficient Content Distribution Network

Yan Chen, Randy H. Katz and John D. Kubiatowicz

Computer Science Division, University of California at Berkeley

Abstract. We present *SCAN*, the Scalable Content Access Network. SCAN combines dynamic replica placement with a self-organizing application-level multicast tree to meet client QoS and server resource constraints. It utilizes an underlying distributed object routing and location system (DOLR) as an essential component. Simulation results on both flash-crowd-like synthetic workloads and real Web server traces show that SCAN deploys close to an optimal number of replicas, achieves good load balance, and incurs a small delay and bandwidth penalty for update multicast relative to static replica placement on IP multicast. We envision that SCAN could enhance a number of different applications, such as content distribution and peer-to-peer file sharing.

1 Introduction

Exponential growth in processor performance, storage capacity, and network bandwidth is changing our view of computing. Our focus has shifted away from centralized, hand-choreographed systems to global-scale, distributed, self-organizing complexes – composed of thousands or millions of elements. Unfortunately, large pervasive systems are likely to have frequent component failures and be easily partitioned by slow or failed network links. Thus, use of local resources is extremely important – both for performance *and* availability. Further, pervasive streaming applications must tune their communication structure to avoid excess resource usage. To achieve both local access *and* efficient communication, we require flexibility in the placement of data replicas and multicast nodes.

One approach for achieving this flexibility while retaining strong properties of the data is to partition the system into two tiers of replicas[9] – a small, durable *primary* tier and a large, soft-state, *second-tier*. The primary tier could represent a Web server (for Web content delivery), the Byzantine inner ring of a storage system [3, 16], or a streaming media provider. The important aspect of the primary tier is that it must hold the most up-to-date copy of data and be responsible for serializing and committing updates. We will treat the primary tier as a black box, called simply "the data source". The second-tier becomes soft-state and will be the focus of this paper. Examples of second-tiers include content-distribution networks (CDNs), file system caches, or web proxy caches.

Because second-tier replicas (or just "replicas") are soft-state, we can dynamically grow and shrink their numbers to meet constraints of the system. We may,

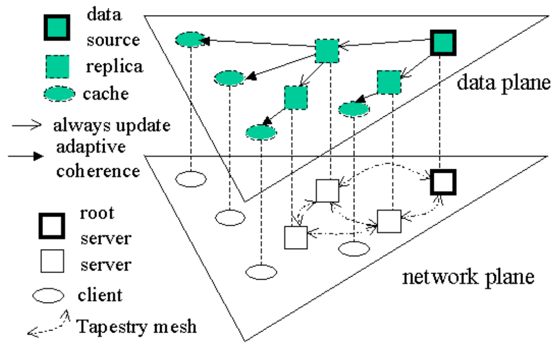

Fig. 1. Architecture of a SCAN system

for instance, wish to achieve a Quality of Service (QoS) guarantee that bounds the maximum network latency between each client and replicas of the data that it is accessing. Since replicas consume resources, we will seek to generate as few replicas as possible to meet this constraint. As a consequence, popular data items may warrant hundreds or thousands of replicas, while unpopular items may require no replicas.

One difficult aspect of unconstrained replication is ensuring that content does not become stale. Slightly relaxed consistency, such as in the Web [10], OceanStore [16], or Coda [14], allows delay between the commitment of updates at the data source and the propagation of updates to replicas. None-the-less, update propagation must still occur in a timely manner. The potentially large number of replicas rules out direct, point-to-point delivery of updates to replicas. In fact, the extremely fluid nature of the second tier suggests a need to self-organize replicas into a multicast tree; we call such a tree a *dissemination tree* (d-tree). Since interior nodes must forward updates to child nodes, we will seek to control the *load* placed on such nodes by restricting the fanout of the tree.

The challenge of second-tier replication is to provide good QoS to clients while retaining *efficient* and *balanced* resource consumption of the underlying infrastructure. To tackle this challenge, we propose a self-organizing soft-state replication system called SCAN: the *Scalable Content Access Network*. Figure 1 illustrates a SCAN system. There are two classes of physical nodes shown in the network-plane of this diagram: *SCAN servers* (squares) and *clients* (circles). We assume that SCAN servers are placed in Internet Data Centers (IDC) of major ISPs with good connectivity to the backbone. Each SCAN server may contain replicas for a variety of data items. One novel aspect of the SCAN system is that it assumes SCAN servers participate in a distributed routing and location (DOLR) system, called Tapestry [11]. Tapestry permits clients to locate nearby replicas without global communication.

There are three types of data illustrated in Figure 1: Data *sources* and *replicas* are the primary topic of this paper and reside on SCAN servers. *Caches* are the images of data that reside on clients and are beyond our scope[1] Our goal is to

[1] Caches may be kept coherent in a variety of ways (for instance [22]).

translate client requests for data into replica management activities. We make the following contributions:

– We provide algorithms that dynamically place a minimal number of replicas while meeting client QoS and server capacity constraints.
– We self-organize these replicas into d-tree with small delay and bandwidth consumption for update dissemination.

The important intuition here is that the presence of the DOLR system enables simultaneous placement of replicas and construction of a dissemination tree without contacting the data source. As a result, each node in a d-tree must maintain state only for its parent and direct children.

The rest of the paper is organized as follows: Section 2 presents related work. Section 3 introduces the Tapestry DOLR while Section 4 describes SCAN algorithms. Results are given in Section 5, while discussion is given in Section 6. We conclude with Section 7.

2 Related Work

Much previous work on replica placement involves *static* placement of replicas – assuming that clients' distribution and access patterns are known in advance[20, 12]. These techniques ignore server capacity constraints and assume explicit knowledge of the global IP network topology. Content Distribution Networks (CDNs) use DNS-based redirection to route clients' requests [1, 7, 25]. Such centralized services do not record locations for each replica. Thus CDNs often place more replicas than necessary, consuming excess storage and update bandwidth.

Inter-domain IP multicast is not widely available; it is also not ideal for Internet distribution [8]. Application-Level Multicast (ALM) builds data distribution trees on top of an efficient overlay network of unicast connections [8, 4, 6, 13, 28]. Most ALM systems utilize a central node to maintain state for all existing children [6, 13, 19, 4] and are not very scalable; some replicate the root to help with this problem [13].

Similar to SCAN, Bayeux [28] is an overlay multicast system build on the Tapestry [11] DOLR; however, unlike SCAN, Bayeux filters all "join" requests through a replicated set of root nodes. Scribe [24] provides a scalable, event-notification multicast system built on top of the Pastry [23] DOLR. It provides mechanisms for dynamic, decentralized construction of multicast groups, but does not include mechanisms for replica placement.

3 Distributed Object Location and Routing

Peer-to-peer researchers have begun to explore distributed object location and routing (DOLR) services [11, 21, 26, 23]. DOLR systems offer a distributed framework in which objects that are named by opaque strings can be located quickly. Since information about objects is distributed throughout the system, messages

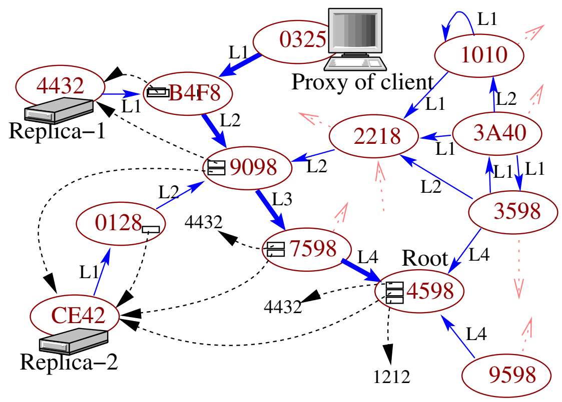

Fig. 2. *The Tapestry Infrastructure:* Nodes route to nodes one digit at a time: e.g. 0325 $\rightarrow B4F8 \rightarrow 9098 \rightarrow 7598 \rightarrow 4598$. Objects are associated with a particular "root" node (e.g. 4598). Servers *publish* replicas by sending messages toward root, leaving back-pointers (dotted arrows). Clients route directly to replicas by sending messages toward root until encountering a pointer (e.g. 0325 $\rightarrow B4F8 \rightarrow 4432$)

are routed to objects by passing them from node to node until they reach their destination. Some DOLR systems have the important property of *routing locality.* This means two things: First, that messages destined for a given object will be routed over a minimal overlay path to their destination; and Second, when two or more objects have the same name, messages destined for that name will be routed to a close object rather than a distant object.

SCAN is built on top of Tapestry [11], a DOLR that exhibits routing locality. Tapestry is an IP overlay network that uses a distributed, fault-tolerant architecture to track the location of objects in the network. In Figure 1, the SCAN servers double as Tapestry nodes. Each client talks to its nearby Tapestry node (*the proxy*) to send object requests. In practice, the proxy node can be located through bootstrap mechanisms.

Figure 2 shows a portion of Tapestry. Each Tapesty node has neighbor links to other Tapestry nodes, forming a routing mesh. Neighbor links provide a route from every node to every other node through incremental, single-digit resolution of the destination. Nodes are added and removed from Tapestry via dynamic membership algorithms [11]. The neighbor-link construction process ensures routing locality.

Each replica is associated with a *Tapestry root node* through a deterministic mapping function[2]. To advertise a replica, the SCAN server storing the object sends a publish message toward the Tapestry location root for that object, depositing *location pointers* each hop. Figure 2 shows two replicas and the Tapestry root for an object. Each object location message is routed toward the object's root until it encounters a pointer, at which point it routes directly to the object.

[2] This root is for location purposes only and has nothing to do with the data source.

The Essential Property: The important property of Tapestry for SCAN is that it provides routing locality, *e.g.* for Figure 2, Client (0325) will find Replica-1 instead of Replica-2. Any DOLR that provides this property will be compatible with the SCAN algorithms in the next section.

4 SCAN Replica Management Algorithms

The presence of an underlying DOLR with routing locality can be exploited to perform simultaneous replica placement and tree construction. Every SCAN server is a member of the DOLR. Hence, new replicas are published into the DOLR. Further, each client directs its requests to its proxy SCAN server; this proxy server interacts with other SCAN servers to deliver content to the client.

Although we use the DOLR to locate replicas during tree building, we otherwise communicate through IP. In particular, we use IP between nodes in a d-tree – parents and children keep track of one another. Further, when a client makes a request that results in placement of a new replica, the client's proxy keeps a cached pointer to this new replica. This permits direct routing of requests from the proxy to the replica. Cached pointers are soft state since we can always use the DOLR to locate replicas.

4.1 Goals for Replica Placement

Replica placement attempts to satisfy both *client latency* and *server load* constraints. *Client latency* refers to the round-trip time required for a client to read information from the SCAN system. We keep this within a pre-specified limit. *Server load* refers to the communication volume handled by a given server. We assume that the load is directly related to the number of clients it handles and number of d-tree children it serves. We keep the load below a specified maximum. Our goal is to meet these constraints while minimizing the number of deployed replicas, keeping the d-tree balanced, and generating as little traffic during update as possible. Our success will be explored in Section 5.

4.2 Dynamic Placement

Our dynamic placement algorithm proceeds in two phases: *replica search* and *replica placement.* The replica search phase attempts to find an existing replica that meets the client latency constraint without being overloaded. If this is successful, we place a link in the client and cache it at the client's proxy server. If not, we proceed to the replica placement phase to place a new replica.

Replica search uses the DOLR to contact a replica "close" to the client proxy; call this the *entry* replica. The locality property of the DOLR ensures that the entry replica is a reasonable candidate to communicate with the client. Further, since the d-tree is connected, the entry replica can contact all other replicas. We can thus imagine three search variants: *Singular* (consider only the entry replica), *Localized* (consider the parent, children, and siblings of the entry replica), and

procedure DynamicReplicaPlacement_Smart(c, o)
c sends *JOIN* request to o through DOLR, reaches entry server s
From s, request forwarded to children (sc), parent (p), and siblings (ss)
Each family member t with $rc_t > 0$ collects $dist_{IP}(c, t)$.
Each t returns rc_t and $dist_{IP}(c, t)$ to s.
At s: Choose family member t with biggest rc_t and $dist_{IP}(t, c) \leq d_c$.
Send result to c.
if *c gets positive result t* **then**
    c chooses t as parent
else
    c sends *PLACEMENT* request to o through DOLR, reaches entry server s
    Request collects $\mathrm{IP}_{s'}$, $dist_{IP}(c, s')$ and $rc_{s'}$ for each server s' on the path.
    At s, choose s' on path with $rc_{s'} > 0$ *and* largest $dist_{IP}(t, c) \leq d_c$
    s puts a replica on s' and becomes its parent, s' becomes parent for c
    s' publishes replica in DOLR
end

Algorithm 1: Smart Replica Placement. Notation: Object o. Client c with latency constraint d_c. Entry Server s. Every server s' has remaining capacity $rc_{s'}$ (additional children it can handle). IP distance, $dist_{IP}$(x,y), collected by sending "pings"

Exhaustive (consider all replicas). For a given variant, we check each of the included replicas and select one that meets our constraints; if none meet the constraint, we proceed to place a new replica.

We restrict replica placement to servers visited by the DOLR routing protocol when sending a message from the client's proxy to the entry replica. We can locate these servers without knowledge of global IP topology. The locality properties of the DOLR suggest that these are good places for replicas. We consider two placement strategies: *Eager* places the replica as close to the client as possible and *Lazy* places the replica as far from the client as possible. If all servers that meet the latency constraint are overloaded, we replace an old replica; if the entry server is overloaded, we disconnect the oldest link among its d-trees.

Dynamic Techniques: We can now combine some of the above options for search and placement to generate dynamic replica management algorithms. Two that we would like to highlight are as follows (see [5] for more information[3]).

- *Naive Placement:* A simple combination utilizes *Singular* search and *Eager* placement. This heuristic generates minimal search and placement traffic.
- *Smart Placement:* A more sophisticated algorithm is shown in Algorithm 5. This algorithm utilizes *Localized* search and *Lazy* placement.

The tradeoff between these approaches is that the latter one consumes more "join" traffic, but constructs a tree with fewer replicas, less update delay, and lower bandwidth consumption. We evaluate this in Section 5.

[3] In fact, in [5] we show how overlay distance can be used to estimate IP distance and reduce ping traffic for part of the algorithm.

Static Comparisons: The dynamic methods above are unlikely to be optimal in terms of the number of replicas deployed, since clients are added sequentially and with limited knowledge of the network topology. For comparison, we will consider two static placement methods in Section 5:

- *IP Static:* Each data source has global IP topology knowledge and is given complete knowledge of all clients. It runs an optimal algorithm [5] to place replicas. It is assumed that the d-tree is formed from IP multicast.
- *Overlay Static:* Again the data source knows the identities of all clients at once. However, it utilizes the overlay distances instead of IP distances to place replicas and build the d-tree.

The first of these is a "guaranteed-not-to-exceed" optimal placement. We expect that it will consume the least total number of replicas and lowest multicast traffic. The second algorithm explores the best that we could expect to achieve gathering all topology information from the DOLR system.

4.3 Soft State Tree Management

Soft-state infrastructures have the potential to be extremely robust, precisely because they can be easily reconfigured to adapt to circumstances. For SCAN we target two types of adaptation: fault recovery and performance tuning.

To achieve fault resilience, the data source sends periodic *heartbeat* messages through the d-tree. Members know the frequency of these heartbeats and can react when they have not seen one for a sufficiently long time. In such a situation, the replica initiates a *rejoin* process – similar to the replica search phase above – to find a new parent. Further, each member periodically sends a *refresh* message to its parent. If the parent does not get the refresh message within a certain threshold, it invalidates the child's entry. With such soft-state group management, any SCAN server may crash without significantly affecting overall CDN performance.

Performance tuning consists of pruning and re-balancing the d-tree. Replicas at the leaves are pruned when they have seen insufficient client traffic. To balance the d-tree, each member periodically rejoins the tree to find a new parent.

5 Evaluation

In this section, we evaluate the performance of the SCAN dynamic replica management algorithms. What we will show is that:

- For realistic workloads, SCAN places close to an optimal number of replicas, while providing good load balance, low delay, and reasonable update bandwidth consumption relative to static replica placement on IP multicast.
- SCAN outperforms the existing DNS-redirection based CDNs on both replication and update bandwidth consumption.
- The performance of SCAN is relatively insensitive to the SCAN server deployment, client/server ratio, and server density.
- The capacity constraint is quite effective at balancing load.

5.1 Experimental Setup

We utilize an event-driven simulation of SCAN. This includes a packet-level network simulator (with a static version of the Tapestry DOLR) and a replica management framework. The soft-state replica layer is driven from simulated clients running workloads.

Our goal is to evaluate the replica schemes of Section 4.2. These strategies are dynamic naive placement (*od_naive*), dynamic smart placement (*od_smart*), overlay static placement (*overlay_s*), and static placement on IP network (*IP_s*). We compare the efficacy of these four schemes via three classes of metrics:

- *Quality of Replica Placement*: Includes number of deployed replicas and degree of load distribution, measured by the ratio of the standard deviation vs. the mean of the number of client children for each replica server.
- *Multicast Performance*: We measure the relative delay penalty (RDP) and the bandwidth consumption which is computed by summing the number of bytes multiplied by the transmission time over every link in the network. For example, the bandwidth consumption for 1K bytes transmitted in two links (one has 10 ms, the other 20 ms latency) is 1KB $\times$ (10+20)ms = 0.03(KB.sec).
- *Tree Construction Traffic:* We count both the number of application-level messages sent and the bandwidth consumption for deploying replicas and constructing d-tree.

In addition, we quantify the effectiveness of capacity constraints by computing the *maximal load* with or without constraints. The maximal load is defined as the maximal number of client cache children on any SCAN server. Sensitivity analysis are carried out for various client/server ratios and server densities.

Network Setup. We use the GT-ITM transit-stub model to generate five 5000-node topologies [27]. The results are averaged over the experiments on the five topologies. A packet-level, priority-queue based event manager is implemented to simulate the network latency. The simulator models the propagation delay of physical links, but does not model bandwidth limitations, queuing delays, or packet losses.

We utilize two strategies for placing SCAN servers. One selects all SCAN servers at random (labeled *random SCAN*). The other preferentially chooses transit and gateway nodes (labeled *backbone SCAN*). This latter approach mimics the strategy of placing SCAN servers strategically in the network.

To compare with a DNS-redirection-based Web content distribution network (CDN), we simulate typical behavior of such a system. We assume that every client request is redirected to the closest CDN server, which will cache a copy of the requested information for the client. This means that popular objects may be cached in every CDN server. We assume that content servers are allowed to send updates to replicas via IP multicast.

Table 1. Statistics of Web site access logs used for simulation

Web site	Period	# Requests total - simulated	# Clients	# Client groups total - simulated	# Objects simulated
MSNBC	10-11 am, 8/2/99	1604944 - 1377620	139890	16369 - 4000	4186
NASA	All day, 7/1/95	64398 - 64398	5177	1842 - 1842	3258

Workloads. To evaluate the replication schemes, we use both a synthetic workload and access logs collected from real Web servers. These workloads are a first step toward exploring more general uses of SCAN.

Our synthetic workload is a simplified approximation of *flash crowds*. Flash crowds are unpredictable, event-driven traffic surges that swamp servers and disrupt site services. For our simulation, all the clients (not servers) make requests to a given hot object in random order.

Our trace-driven simulation includes a large and popular commercial news site, MSNBC [17], as well as traces from NASA Kennedy Space Center [18]. Table 1 shows the detailed trace information. We use the access logs in the following way. We group the Web clients based on BGP prefixes [15] using the BGP tables from a BBNPlanet (Genuity) router [2]. For the NASA traces, since most entries in the traces contain host names, we group the clients based on their domains, which we define as the last two parts of the host names (e.g., a1.b1.com and a2.b1.com belong to the same domain). Given the maximal topology we can simulate is 5000 (limited by machine memory), we simulate all the clients groups for NASA and 4000 top client groups (cover 86.1% of requests) for MSNBC. Since the clients are unlikely to be on transit nodes nor on server nodes, we map them randomly to the rest of nodes in the topology.

5.2 Results for Synthetic Workload

We start by examining the synthetic, flash crowd workload. 500 nodes are chosen to be SCAN servers with either "random" or "backbone" approach. Remaining nodes are clients and access some hot object in a random order. We randomly choose one non-transit SCAN server to be the data source and set as 50KB the size of the hot object. Further, we assume the latency constraint is 50ms and the load capacity is 200 clients/server.

Comparison Between Strategies. Figure 3 shows the number of replicas placed and the load distribution on these servers. *Od_smart* approach uses only about 30% to 60% of the servers used by *od_naive*, is even better than *overlay_s*, and is very close to the optimal case: *IP_s*. Also note that *od_smart* has better load distribution than *od_naive* and *overlay_s*, close to *IP_s* for both *random* and *backbone SCAN*.

Relative Delay Penalty (RDP) is the ratio of the overlay delay between the root and any member in d-tree vs. the unicast delay between them [6]. In Figure 4, *od_smart* has better RDP than *od_naive*, and 85% of *od_smart* RDPs between any member server and the root pairs are within 4. Figure 5 contrasts the bandwidth consumption of various replica placement techniques with the optimal IP

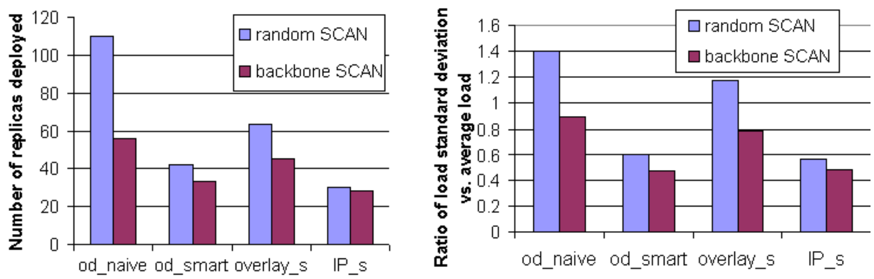

Fig. 3. Number of replicas deployed (left) and load distribution on selected servers (right) (500 SCAN servers)

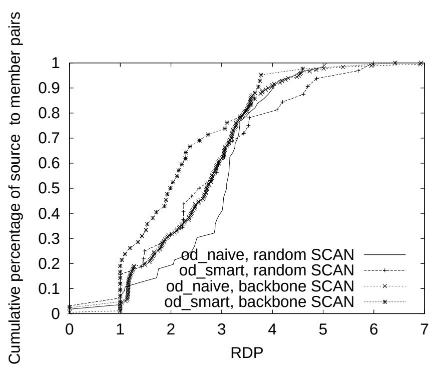

Fig. 4. Cumulative distribution of RDP (500 SCAN servers)

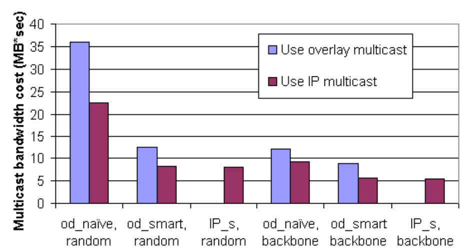

Fig. 5. Bandwidth consumption of 1MB update multicast (500 SCAN servers)

static placement. The results are very encouraging: the bandwidth consumption of *od_smart* is quite close to *IP_s* and is much less than that of *od_naive*.

The performance above is achieved at the cost of d-tree construction (Figure 6). However, for both *random* and *backbone SCAN*, *od_smart* approach produces less than three times of the messages of *od_naive* and less than six times of that for optimal case: *IP_s*. Meanwhile, *od_naive* uses almost the same amount of bandwidth as *IP_s* while *od_smart* uses about three to five times that of *IP_s*.

In short, the smart dynamic algorithm has performance that is close to the ideal case (static placement with IP multicast). It places close to an optimal number of replicas, provides better load distribution, and less delay and multicast bandwidth consumption than the naive approach – at the price of three to five times as much tree construction traffic. Since d-tree construction is a much less frequent than data access and update this is a good tradeoff.

Due to the limited number and/or distribution of servers, there may exist some clients who cannot be covered when facing the QoS and capacity requirements. In this case, our algorithm can provide hints as where to place more servers. Note that experiments show that the naive scheme has many more un-

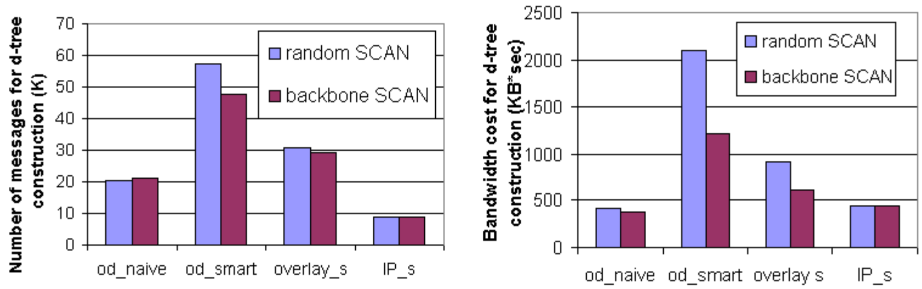

Fig. 6. Number of application-level messages (left) and total bandwidth consumed (right) for d-tree construction (500 SCAN servers)

covered clients than the smart one, due to the nature of its unbalanced load. Thus, we remove it from consideration for the rest of synthetic workload study.

Comparison with a CDN. As an additional comparison, we contrast the overlay smart approach with a DNS-redirection-based CDN. Compared with a traditional CDN, the overlay smart approach uses a fraction of the number of replicas (6-8%) and less than 10% of bandwidth for disseminating updates.

Effectiveness of Distributed Load Balancing. We study how the capacity constraint helps load balancing with three client populations: 100, 1000 and 4500. The former two are randomly selected from 4500 clients. Figure 7 shows that lack of capacity constraints (labeled *w/o LB*) leads to hot spot or congestion: some servers will take on about 2-13 times their maximum load. Performance with load balancing is labeled as *w/ LB* for contrast.

Performance Sensitivity to Client/Server Ratio. We further evaluate SCAN with the three client populations Figure 8 shows the number of replicas deployed. When the number of clients is small, *w/ LB* and *w/o LB* do not differ much because no server exceeds the constraint. The number of replicas required for *od_smart* is consistently less than that of *overlay_s* and within the bound of 1.5 for *IP_s*. As before, we also simulate other metrics, such as load distribution, delay and bandwidth penalty for update multicast under various client/server ratios. The trends are similar, that is, "od_smart" is always better than "overlay_s", and very close to "IP_s". We omit the graphs to save the space.

Performance Sensitivity to Server Density. Next, we increase the density of SCAN servers. We randomly choose 2500 out of the 5000 nodes to be SCAN servers and measure the resulting performance. Obviously, this configuration can support better QoS for clients and require less capacity for servers. Hence, we set the latency constraint to be 30 ms and capacity constraint 50 clients/server. The number of clients vary from 100 to 2500.

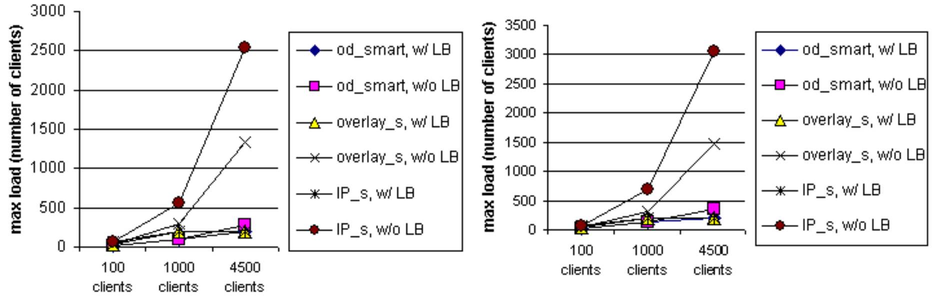

Fig. 7. Maximal load measured with and without load balancing constraints (LB) for various numbers of clients (left: 500 random servers, right: 500 backbone servers)

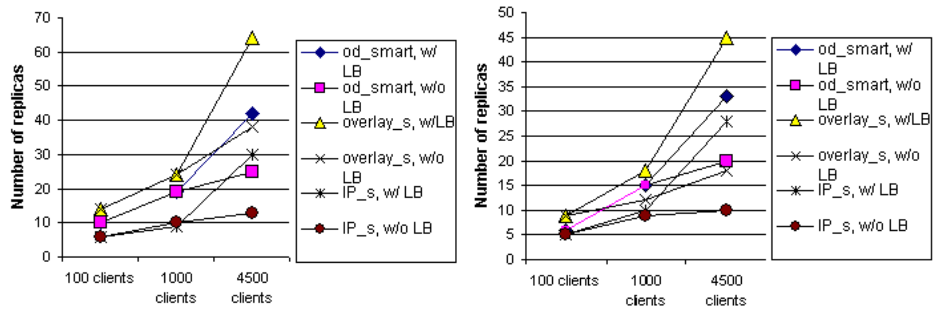

Fig. 8. Number of replicas deployed with and without load balancing constraints (LB) for various numbers of clients (left: 500 random servers, right: 500 backbone servers)

With very dense SCAN servers, our *od_smart* still uses less replicas than *overlay_s*, although they are quite close. *IP_s* only needs about half of the replicas, as in Figure 9. In addition, we notice that the load balancing is still effective. That is, overloaded machines or congestion cannot be avoided simply by adding more servers while neglecting careful design.

Due to space limitations, we cannot show all simulation results here. In summary, *od_smart* performs well with various SCAN server deployments, various client/server ratios, and various server densities. The capacity constraint based distributed load balancing is effective.

5.3 Results for Web Traces Workload

Next, we explore the behavior of SCAN for Web traces with documents of widely varying popularity. Figure 10.a characterizes the request distribution for the two traces used (note that the x-axis is logarithmic.). This figure reveals that the request number for different URLs is quite unevenly distributed for both traces.

For each URL in the traces, we compute the number of replicas generated with *od_naive*, *od_smart*, and IP_s. Then we normalize the replica numbers of *od_naive* and *od_smart* by dividing them with the replica number of IP_s. We plot the CDF of such ratios for both NASA and MSNBC in Figure 10.b. The

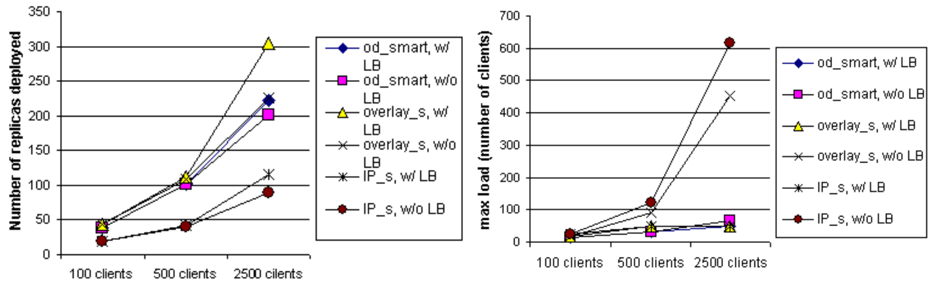

Fig. 9. Number of replicas deployed (left) and maximal load (right) on 2500 random SCAN servers with and without the load balancing constraint (LB)

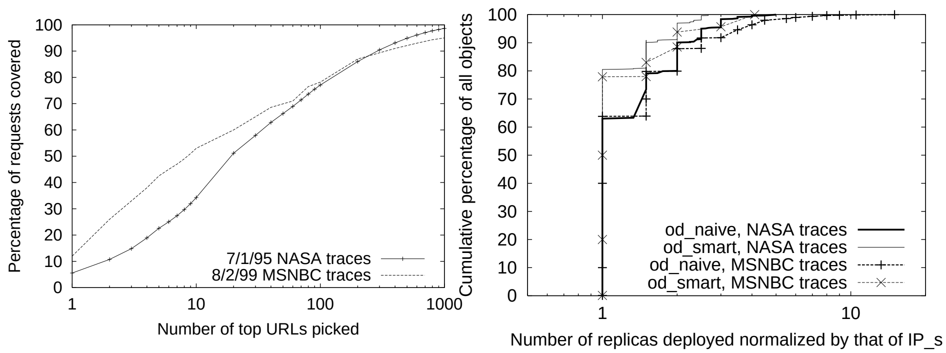

Fig. 10. Simulation with NASA and MSNBC traces on 100 backbone SCAN servers. (a) Percentage of requests covered by different number of top URLs (left); (b) the CDF of replica number deployed with *od_naive* and *od_smart* normalized by the number of replicas using IP_s (right)

lower percentage part of the CDF curves are overlapped and close to 1. The reasons are most of the URLs have very few requests, and we only simulate a limited period, thus the number of replicas deployed by the three methods are very small and similar. However, *od_smart* and *od_naive* differ significantly for popular objects, exhibited in the higher percentage part. *Od_smart* is very close to IP_s, for all objects, the ratio is less than 2.7 for NASA and 4.1 for MSNBC, while the ratio for *od_naive* can go as high as 5.0 and 15.0, respectively.

In addition, we contrast the bandwidth consumption for disseminating updates. Given an update of unit size, for each URL, we compute the bandwidth consumed by using (1) overlay multicast on *od_naive* tree, (2) overlay multicast on *od_smart* tree, and (3) IP multicast on IP_s tree. Again, we have metric (1) and (2) normalized by (3), and plot the CDF of the ratios. The curves are quite similar to Figure 10.b. So we omit them in the interest of space.

In conclusion, although *od_smart* and *od_naive* perform similarly for infrequent or cold objects, *od_smart* outperforms dramatically over *od_naive* for hot objects which dominate overall requests.

6 Discussion

How does the distortion of topology through Tapestry affect replica placement? Notice that the overlay distance through Tapestry, on average, is about 2-3 times more than the IP distance. Our simulations in Sec. 5, shed some light on the resulting penalty: *Overlay_s* applies exactly the same algorithm as IP_s for replica placement, but uses the static Tapestry-level topology instead of IP-level topology. Simulation results show that *overlay_s* places 1.5 - 2 times more replicas than IP_s. For similar reasons, *od_smart* outperforms *overlay_s*. The reason is that *od_smart* uses "ping" messages to get the real IP distance between clients and servers. This observation also explains why *od_smart* gets similar performance to IP_s. One could imagine scaling overlay latency by an expected "stretch" factor to estimate real IP distance – thereby reducing ping probe traffic.

We start this paper with a general discussion of second-tier, soft-state replication. How does the SCAN system fulfill that role? Our initial results are encouraging precisely because they demonstrate that a dynamic, self-organizing system can utilize resources *conservatively* in the *interior* of the network to meet client QoS requirements. This is in marked contrast to many peer-to-peer replication systems that cache information only at clients, or CDN systems that cache data widely. The underlying DOLR (Tapestry) provides the essential mechanism for distributing information about resources and topology. The exploitation of such an infrastructure is a powerful organizing principle for future systems.

7 Conclusions

The importance of adaptive replica placement and update dissemination is growing as distribution systems become pervasive and global. In this paper, we present SCAN, a scalable, soft-state replica management framework built on top of a distributed object location and routing framework (DOLR) with locality. SCAN generates replicas on demand and self-organizes them into an application-level multicast tree, while respecting client QoS and server capacity constraints. An event-driven simulation of SCAN shows that SCAN places close to an optimal number of replicas, while providing good load distribution, low delay, and small multicast bandwidth consumption compared with static replica placement on IP multicast. Further, SCAN outperforms existing DNS-redirection based CDNs in terms of replication and update cost. SCAN shows great promise as an essential component of global-scale peer-to-peer infrastructures.

8 Acknowledgments

We graciously acknowledge sponsorship and grants from DARPA (grant N66061-99-2-8913), NSF career award #ANI-9985250, California Micro Grant #01-042, Ericsson, Nokia, Siemens, Sprint, NTTDoCoMo and HRL laboratories. We thank Lili Qiu for providing anonymized MSNBC traces, thank Albert Wang for help with implementation, and thank Johnny Lam and Lakshminarayanan Subramanian for reviewing the draft of the paper.

References

1. Akamai Technologies Inc. http://www.akamai.com.
2. BBNPlanet. telnet://ner-routes.bbnplanet.net.
3. M. Castro and B. Liskov. Proactive recovery in a byzantine-fault-tolerant system. In *Proc. of USENIX Symp. on OSDI*, 2000.
4. Y. Chawathe, S. McCanne, and E. Brewer. RMX: Reliable multicast for heterogeneous networks. In *Proceedings of IEEE INFOCOM*, 2000.
5. Y. Chen et al. Dynamic replica placement for scalable content delivery. In *Proc. of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Mar. 2002.
6. Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS*, June 2000.
7. Digital Island Inc. http://www.digitalisland.com.
8. P. Francis. Yoid: Extending the Internet multicast architecture. Technical report, ICIR, http://www.icir.org/yoid/docs/yoidArch.ps, April, 2000.
9. J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proc. of ACM SIGMOD Conf.*, pages 173–182, 1996.
10. James Gwertzman and Margo Seltzer. World-Wide Web Cache Consistency, 1996.
11. K. Hildrum, J. Kubiatowicz, S. Rao, and B. Zhao. Distributed data location in a dynamic network. In *Proc. of ACM SPAA*, 2002.
12. S. Jamin, C. Jin, A. Kurc, D. Raz, and Y. Shavitt. Constrained mirror placement on the Internet. In *Proceedings of IEEE Infocom*, 2001.
13. J. Jannotti et al. Overcast: Reliable multicasting with an overlay network. In *Proceedings of OSDI*, 2000.
14. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.
15. B. Krishnamurthy and J. Wang. On network-aware clustering of Web clients. In *Proc. of SIGCOMM*, 2000.
16. John Kubiatowicz et al. Oceanstore: An architecture for global-scale persistent storage. In *Proceeedings of 9th ASPLOS*, 2000.
17. MSNBC. http://www.msnbc.com.
18. NASA server traces. http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html.
19. D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *Proceedings of 3rd USITS*, 2001.
20. L. Qiu, V. N. Padmanabhan, and G. Voelker. On the placement of Web server replicas. In *Proceedings of IEEE Infocom*, 2001.
21. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, 2001.
22. P. Rodriguez and S. Sibal. SPREAD: Scaleable platform for reliable and efficient automated distribution. In *Proceedings of WWW*, 2000.
23. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of Middleware 2001*.
24. A. Rowstron, A-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *Proceedings of NGC*, 2001.
25. Speedera Inc. http://www.speedera.com.
26. I. Stoica et al. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM*, 2001.
27. E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an Internetwork. In *Proceedings of IEEE INFOCOM*, 1996.
28. S. Q. Zhuang et al. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of ACM NOSSDAV*, 2001.