MICHAEL J. FREEDMAN

automating server selection with OASIS

Michael J. Freedman is a doctoral student at NYU, currently visiting Stanford University, and received his M.Eng. and S.B. degrees from MIT. His research interests are security, distributed systems, and cryptography. He is the author of the Coral Content Distribution Network (http://www.coralcdn.org/) and OASIS (http://oasis.coralcdn.org).

mfreed@cs.nyu.edu

OASIS PROVIDES A PUBLICLY AVAILABLE, locality-aware server-selection infrastructure. Replicated servers adopting OASIS each run a small application that communicates with the OASIS infrastructure, sharing information about their current load levels and their measured network response times to selected IP addresses. OASIS in turn can redirect unmodified clients to nearby and/or lightly loaded live replica servers. In this article, I explain how OASIS works, provide some performance analysis, and describe how services can start using OASIS.

OASIS (Overlay Anycast Service InfraStructure) has been publicly deployed since November 2005 on PlanetLab [10], a distributed testbed running at over 300 academic and industry sites. It has already been adopted by a number of services [1, 2, 4, 7, 8, 11, 12]. OASIS supports a variety of protocols—currently DNS, HTTP, and RPC—that redirect unmodified clients for server selection and that expose its geolocation and distance-estimation functionality to OASIS-aware hosts.

Why This Matters

High-volume Web sites are typically replicated at multiple locations for performance and availability. Content distribution networks amplify a Web site's capacity by serving clients through a large network of Web proxies. File-sharing, instant messaging, and VoIP systems use rendezvous servers to bridge hosts behind NATs.

In all of these examples, system designers must tackle the problem of *server selection*: After deploying servers at various locations throughout the Internet, they must now direct clients to ones that are appropriately nearby or unloaded. Or, posed more concretely, when accessing a replicated Web site, from which mirror should a user download?

This server-selection problem is not merely academic: The performance and cost of such systems depend highly on clients' choice of servers. File download times can vary greatly based on the locality and load of the chosen replica. A service provider's costs may depend on the load spikes that the selection mechanism produces, as many

data centers charge customers based on the 95th-percentile bandwidth usage over all five-minute periods in a month.

Unfortunately, common techniques for replica selection produce suboptimal results. Asking human users to select the best replica is both inconvenient and inaccurate. Round-robin and other primitive DNS techniques spread load, but do little for network locality.

OASIS, however, can automate the process of selecting nearby and/or lightly loaded servers, yet it remains easy to integrate into existing applications.

Architecture

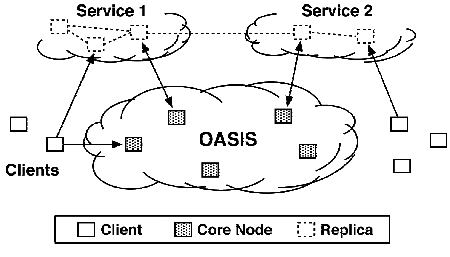

FIGURE 1. OASIS SYSTEM COMPONENTS

Figure 1 shows OASIS's two-tier architecture. The system consists of a network of *core* nodes that help *clients* select appropriate *replicas* of various services. All services employ the same core nodes (which we run as a public service); we intend this set of infrastructure nodes to be small enough and sufficiently reliable so that every core node can know most of the others. Replicas (which are deployed by service operators seeking to use OASIS for server selection) also run OASIS-specific code, both to report their own load and liveness information to the core and to assist the core with network measurements. Clients need not run any special code to use OASIS, because the core nodes provide DNS- and HTTP-based redirection services.

The primary function of the OASIS core is to return a suitable service replica to a server-selection request. Given that a request only provides the client's IP address and the service name (encoded in the domain name being resolved when using DNS), how does OASIS determine a client's location, which is needed to discover nearby replicas?

GEOLOCATING IP ADDRESSES

To discover the location of clients, OASIS probes Internet destinations using the replica servers as vantage points and, in doing so, finds the closest replica. One of OASIS's main contributions is a set of techniques that make it practical to measure the entire Internet in advance and therefore eliminate on-demand probing when clients make requests.

OASIS minimizes probing and reduces its susceptibility to network peculiarities by exploiting *geographic coordinates* as a basis for locality and by leveraging the locality of the IP prefixes [6] (e.g., NYU has IP prefix 216.165.0.0/17). We assume that every replica knows its own latitude and longitude, which already provides some information about locality before any network measurement. Then, in the background, OASIS uses service replicas as vantage points to probe each IP prefix to discover the replica with lowest round-trip-time (yet still does so in a manner that minimizes probing [13]). Finally, OASIS stores the geographic coordinates of the replica closest to each prefix it maps.

Because the physical location of IP prefixes rarely changes, an accurately pinpointed network can be safely reprobed infrequently (as rarely as once a week). Additionally, this approach amortizes bandwidth costs across the multiple services using OASIS, resulting in an acceptable per-node cost that only decreases as more services adopt OASIS. Such infrequent, background probing both reduces the risk of abuse complaints and allows the system to respond quickly to requests, with no need for on-demand probing.

RESOLVING SERVER-SELECTION REQUESTS

What happens when a client makes a selection request to a core node? First, a core node maps the client's IP address to an IP prefix of appropriate granularity to capture locality properties. It then attempts to map the IP prefix to geographic coordinates. If successful, OASIS returns the closest service replicas to that location (unless load-balancing requires further consideration of load as a primary selection metric). Otherwise, if it cannot determine the client's location, it returns random service replicas.

This server-selection process relies on four databases maintained in a distributed manner by the core: (1) A *service table* lists all services using OASIS (and records policy information for each service), (2) a *bucketing table* maps IP addresses to prefixes, (3) a *proximity table* maps prefixes to coordinates, and (4) one *liveness table per service* includes all live replicas belonging to the service and their corresponding information (i.e., coordinates, load, and capacity).

How are these tables managed in a distributed manner? OASIS optimizes response time by heavily replicating most information. Service, bucketing, and proximity information need only be weakly consistent; stale information only affects system performance, not its correctness. Thus, OASIS uses gossiping to efficiently disseminate such state—as well as for failure notifications regarding core nodes—throughout the network.

Replica liveness information, however, must be fresher: DNS resolvers and Web browsers deal poorly with unavailable replicas, since such client applications cache stale addresses longer than appropriate. To tolerate replica failures robustly, replica information is maintained using soft-state: Replicas periodically send registration messages to core nodes (currently, every 60 seconds). This replica process also regularly connects to the local application seeking OASIS service to verify its liveness (i.e., every 15 seconds). These communications are shown in Figure 2.

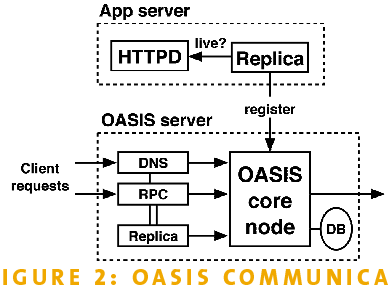

FIGURE 2: OASIS COMMUNICATION DIAGRAM

OASIS must know most replicas belonging to a service to answer corresponding selection requests. Therefore, OASIS aggregates replica liveness information for each particular service at a few core nodes known as *service rendezvous nodes*. To provide self-organizing properties within the core, different sets of core nodes are chosen via consistent hashing [9] to play the role of rendezvous nodes for each service.

Although all core nodes can map a client's IP address to geographic coordinates and determine the relevant service policy, when a core node receives a service request for which it does not play the role of rendezvous node, it must also send an RPC query to one of the requested service's rendezvous nodes. This rendezvous node uses its aggregated list of known replicas to determine the best-suited replicas for the client. In [5], I describe a variety of additional optimizations to reduce the load on a service's rendezvous nodes for increased scalability.

Evaluation

I now briefly present some wide-area measurements of OASIS on PlanetLab [10]. This section is meant simply to demonstrate that OASIS can greatly improve end-to-end latencies and load-balancing for replicated systems. For a full evaluation of OASIS and a complete explanation of the experiments, please see [5].

Figure 3 shows the end-to-end time for clients to download a Web page from a domain name being served by OASIS. This time includes a DNS

lookup and the subsequent TCP transfers. Using 250 PlanetLab hosts, we compare OASIS to a variety of other state-of-the-art and simplistic server-selection schemes, include using Meridian for on-demand probing [13], Vivaldi for virtual coordinates [3], and round-robin selection. The median response time for OASIS is 290% faster than Meridian and 500% faster than simple round-robin systems. These end-to-end measurements underscore OASIS's true performance benefit, coupling fast DNS response time (by using cached information) with accurate server selection.

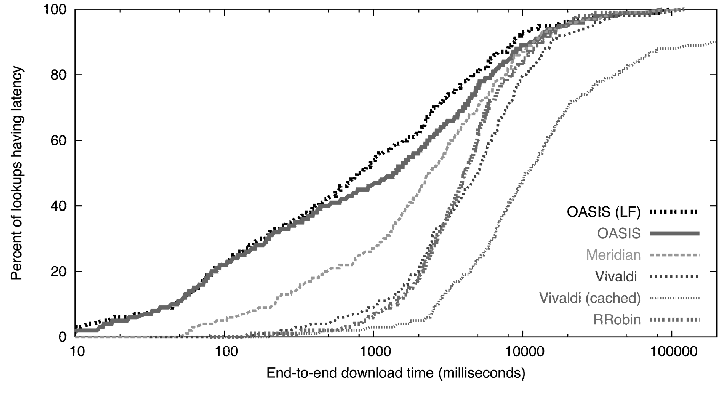

FIGURE 3: GRAPH (CDF) OF LOOKUP LATENCY VERSUS DOWNLOAD TIME

Table 1 shows how OASIS can reduce bandwidth costs associated with 95th-percentile billing. When multiple co-located clients (here, all in California) make requests against our four distributed Web servers, OASIS's load balancing ensures that 95% peak load remains evenly balanced. Purely locality-based selection, in contrast, yields a traffic spike at the nearest Web server.

Metric	*California*	*Texas*	*New York*	*Germany*
Latency	23.3	0.0	0.0	0.0
Load	9.0	11.3	9.6	9.2

TABLE 1: 95TH-PERCENTILE BANDWIDTH USAGE (IN MB PER MINUTE)

I next describe how services can adopt OASIS to enjoy similar performance benefits.

Using OASIS

Figure 4 shows various ways in which legacy clients and services can use OASIS to access a service. In our usage scenarios I use CoralCDN [4], an open content distribution network we have been running since early 2004. CoralCDN receives about 25 million requests daily from over 1 million clients; in fact, it motivated us to build OASIS in the first place to provide better proxy selection.

A CLIENT'S STEP-BY-STEP BEHAVIOR

The top diagram of Figure 4 shows how to make legacy clients select replicas using DNS redirection. The service provider advertises a domain name served by OASIS (e.g., coralcdn.nyuld.net). (OASIS currently uses the domain name .nyuld.net for its core nodes.) When a client looks up that domain name (Step 1), OASIS first redirects the client's resolver to a nearby

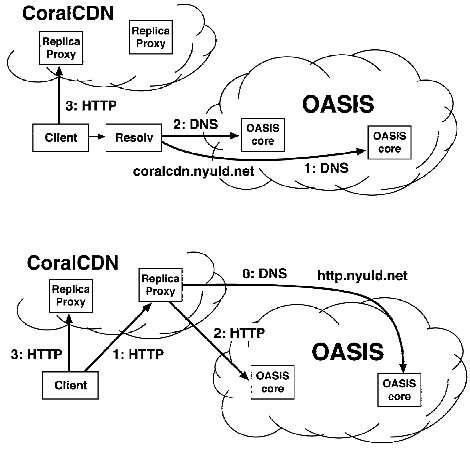

FIGURE 4: USES OF OASIS FOR ACCESSING A SERVER

OASIS nameserver (by resolving dns.nyuld.net with respect to the client's IP address and returning the results as NS records). The client's resolver caches these nameservers for future accesses. Then the resolver queries this nearby nameserver (Step 2) to determine the address of nearby, unloaded CoralCDN Web proxies (returned as the domain's A records). This approach can be accurate, provided that clients are near their resolvers.

The bottom diagram shows an alternative based on application-level HTTP redirection. Here, the CoralCDN replicas are also clients from OASIS's point of view. Each replica connects to a nearby OASIS core node that provides HTTP service, as selected by DNS redirection for http.nyuld.net (Step 0). When a client connects to a replica (Step 1), that replica queries OASIS to find a better replica (Step 2), now asking for service by explicitly specifying the client's IP address in an HTTP query string. Finally, an HTTP redirect is returned to the client, causing it to contact the selected replica for service (Step 3). Such an approach does not require that clients be located near their resolvers in order to achieve accurate locality.

In fact, OASIS supports several variations on this same theme: The CoralCDN replica can query the OASIS core using RPC, instead of HTTP. Alternatively, the replica's query can simply ask for the estimated distance between two IP addresses, which only uses the core's location database and does not require that it maintain a service-specific replica state (although the service itself would then need to maintain liveness information). Furthermore, the HTTP server on core nodes can perform HTTP redirection for clients themselves, avoiding the need for clients to contact the initial replica proxy (Step 1).

The rest of this section is devoted to the concrete steps a service operator needs to perform in order to integrate OASIS into their distributed system.

REGISTERING A SERVICE

A service policy must be registered with the core so that OASIS can handle its server selection. This policy currently includes a service's name (e.g., coralcdn), the number and expiration time of replica addresses returned per request, and the selection criteria. By default, OASIS selects replicas based on locality, unless the nearer replica's load exceeds its capacity. Other policies support pure locality-based selection or a load-balancing algorithm meant to reduce costs associated with 95th-percentile billing.

To enable sites to publish their own top-level domain names, OASIS supports aliases. Thus, in the context of CoralCDN, requests to nyud.net will be interpreted as coralcdn.nyuld.net or, in the case of the OverCite service [12], overcite.org gets interpreted as overcite.nyuld.net. To support this aliasing, however, a server operator must also point the nameserver records for their top-level domain to some subset of OASIS's nameservers.

DEPLOYING REPLICAS AND INTEGRATING APPLICATIONS

On every host running a service application—such as a CoralCDN Web proxy—the service's administrator should deploy an OASIS replica. (The source code is released under the GPLv2 and is available from http://oasis.coralcdn.org/.) Service replicas should be configured with their geographic coordinates and, in order to monitor its liveness, the service name and listening port of their local application.

On the application side, the application (or some stand-alone daemon monitoring it) simply needs to listen on a TCP server socket on the config-

ured port. Then, when the local OASIS replica connects to the application (every 15 seconds by default), the application should simply accept the connection, respond with its application status (a shared secret code for verification, its current load, and its maximum capacity), and then close the connection.

We already run OASIS replicas on most PlanetLab hosts [10] as a public service to the PlanetLab community. Thus, system developers seeking to deploy their services on PlanetLab need only configure their application to respond to our local liveness checks and need not deploy replicas themselves, as a single OASIS replica can monitor multiple local services and their applications.

ACCESSING THE SERVER-SELECTION AND GEOLOCATION SERVICE

Once a service's policy and some of its replicas are registered with the OASIS core, core nodes can immediately respond to client server-selection requests. OASIS currently provides DNS, HTTP, and RPC interfaces for server selection, as shown in Figure 4, above. To access a CoralCDN Web proxy via DNS redirection, for example, a client need only connect to the hostname coralcdn.nyuld.net. To use HTTP redirection, the client simply accesses the URL http://http.nyuld.net:8096/redir.html?pol=coralcdn&ip=<ip>, which causes the client first to discover a nearby core node running an HTTP proxy (via DNS), then to ask that HTTP proxy for a nearby CoralCDN replica. The optional query string <ip> performs the request with respect to that specified IP address, as opposed to the client's own IP address.

OASIS exposes additional information through its HTTP and RPC interfaces. For example, a client can query OASIS for the geographic coordinates of a particular IP address or the distance between any two such addresses: http://http.nyuld.net:8096/distance.xml?src=<ip1>&dst=<ip2>.

Output from the HTTP proxy can be either in HTML or XML, the latter allowing for simple integration with third-party Web services.

Conclusion

OASIS is a global, distributed, server-selection system that allows legacy clients to find nearby or unloaded replicas of distributed services. Two main features distinguish OASIS from prior systems. First, OASIS allows multiple application services to share the selection service. Second, OASIS avoids any on-demand probing when clients initiate requests by geolocating all IP prefixes in advance.

Publicly deployed since November 2005, OASIS has already been adopted by a number of distributed services [1, 2, 4, 7, 8, 11, 12]. Experimental measurements and third-party experiences suggest that OASIS produces highly accurate results, ensures server liveness, and provides simple system integration and client use. For more technical information on OASIS's design, evaluation, and integration, as well as relevant source code, please visit http://oasis.coralcdn.org/.

REFERENCES

[1] F. Annexstein, K. Berman, S. Strunjas, and C. Yoshikawa, "Adaptive Client-Server Load Balancing Using Persistent Demands," Technical Report ECECS-TR-2006-06, University of Cincinnati, July 2006.

[2] B.-G. Chun, P. Wu, H. Weatherspoon, and J. Kubiatowicz, "ChunkCast: An Anycast Service for Large Content Distribution," *Proceedings of the 5th International Workshop on Peer-to-Peer Systems* (February 2006).

[3] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A Decentralized Network Coordinate System," *Proceedings of SIGCOMM* (August 2004).

[4] M. J. Freedman, E. Freudenthal, and D. Mazières, "Democratizing Content Publication with Coral," *Proceedings of the First Symposium on Networked Systems Design and Implementation* (March 2004).

[5] M. J. Freedman, K. Lakshminarayanan, and D. Mazières, "OASIS: Anycast for Any Service," *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation* (May 2006).

[6] M. J. Freedman, M. Vutukuru, N. Feamster, and H. Balakrishnan, "Geographic Locality of IP Prefixes," *Proceedings of the Internet Measurement Conference* (October 2005).

[7] R. Grimm, G. Lichtman, N. Michalakis, A. Elliston, A. Kravetz, J. Miller, and S. Raza, "Na Kika: Secure Service Execution and Composition in an Open Edge-side Computing Network," *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation* (May 2006).

[8] D. Joseph, J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle, "OCALA: An Architecture for Supporting Legacy Applications over Overlays," *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation* (May 2006).

[9] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy, "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web," in *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing* (May 1997).

[10] PlanetLab: http://www.planet-lab.org/.

[11] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "OpenDHT: A Public DHT Service and Its Uses," *Proceedings of SIGCOMM* (August 2005).

[12] J. Stribling, J. Li, I. Councill, M. F. Kaashoek, and R. Morris, "OverCite: A Cooperative Digital Research Library," *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation* (May 2006).

[13] B. Wong, A. Slivkins, and E. G. Sirer, "Meridian: A Lightweight Network Location Service without Virtual Coordinates," *Proceedings of SIGCOMM* (August 2005).