Server Switching: Yesterday and Tomorrow

Jeffrey S. Chase [*]
Department of Computer Science
Duke University
chase@cs.duke.edu

Abstract

Server switches distribute incoming request traffic across the nodes of Internet server clusters and Web proxy cache arrays. These switches are a standard building block for large-scale Internet services, with many commercial products on the market. As Internet applications and service architectures continue to evolve, the role of server switches and the demands on their request routing policies will also change.

This paper explores interrelated factors shaping the role of server switching. These factors include the emergence of content delivery networks, the increasing prevalence of dynamic content, persistent connections in the HTTP 1.1 standard, the changing nature of Web server clusters, and the opportunities to apply server switching techniques to service protocols other than HTTP. We outline the potential role of server switches in virtualizing IP-based storage protocols and as a foundation for adaptive resource provisioning in server clusters.

1 Introduction

Clustering technologies enable incremental scaling of Internet server sites at modest cost. It is increasingly common in cluster-based service architectures to distribute incoming request traffic among servers using redirecting switches. These *server switches* are called by various names including request distributors, front ends, redirectors, load balancers, network dispatchers, L4-L7 switches, interception switches, Web switches, and content switches. Redirecting switches are also used in Web proxy cache arrays.

During the last few years an active commercial market for server switching products has emerged. Many of these products are Ethernet switches supplemented with built-in processing power to examine the incoming packet stream and manage service traffic intelligently, assigning requests to servers based on request content, client session, and/or server status. Server switches and their request routing (server selection) policies play a key role in managing content and server resources for scalable Internet services.

The market for server switches is now relatively mature, and they are an accepted building block for large-scale Internet services. However, Internet services and their delivery architectures continue to evolve rapidly. This creates new challenges and opportunities for server switches, and raises fundamental questions about the future role of intelligent network switching elements.

This paper explores several factors shaping the future role of server switching.

- **The trickle-down effect.** Demand-side proxy caching and wide-area content delivery networks (CDNs) are now ubiquitous in the Web; these agents filter the incoming request stream, changing its properties in fundamental ways that affect request distribution strategies.
- **The dynamic Web.** Requests for *dynamic* content increasingly dominate server traffic. One factor driving this shift is that proxy caches and CDNs absorb an increasing share of the request stream for static content. At the same time, the Internet buildout is driving a shift toward server-based computing; the new generation of Web services increasingly incorporate dynamically generated, personalized content, and act as a basis for Internet delivery of server-based application utilities through Application Service Providers.
- **Persistent connections.** Content-aware request routing policies select the server for each request independently; some of the key policies in use (e.g., URL switching) rely on this. However, new Web standards (HTTP 1.1) emphasize use

[*]This work is supported by the National Science Foundation through grants CCR-00-82912 and EIA-9972879.

of persistent connections, in which multiple requests arrive on a single transport (TCP) connection. This creates significant new challenges for future server switches, and may constrain the policies they can support. Other emerging standards such as IPSEC may create similar pressures.

- **Server resource provisioning.** Internet services are increasingly hosted in shared data centers managed by third-party hosting providers. Shared hosting centers offer economies of scale and a potential to dynamically adjust capacity provisioning to respond to request traffic, quality-of-service specifications, and network conditions. Reconfigurable server switches are an enabling technology for global load management and resource provisioning in shared server clusters.
- **IP-based network storage.** Even as the Web evolves, other large-scale IP-based services are emerging. This creates new opportunities to apply server switching techniques to virtualize these services. In particular, storage is increasingly network-based, and network storage architectures are shifting from FibreChannel to IP networks. Server switching technology could enable scalable "virtual servers" for IP-based network storage protocols such as iSCSI and NFS (e.g., [2]).

This paper discusses the interplay of these factors, and speculates on their implications. These factors suggest a more limited role for sophisticated content-based switching features in the future, and an expanding role for server switches as a focal point for managing server resources rather than content. At the same time, the potential of server switching beyond HTTP raises new issues for transport protocols, service structure, and switch architecture.

This paper is organized as follows. Section 2 gives an overview of server switching for Internet service architectures. Section 3 deals with the changing role of server switching for delivery of static Web content, and Section 4 discusses the impact of the shift to dynamic content. The next two sections outline new roles for server switching in adaptive resource provisioning for hosting centers (Section 5) and scalable IP storage (Section 6), and research issues raised by those roles. Section 7 concludes.

2 Service Virtualization with Server Switches

Server switching is a technique to *virtualize* services at the IP level. An ensemble of servers cooperate to serve the request load. Clients interact with the service through a client/server protocol such as HTTP, addressing their request traffic to a *virtual IP address* representing the service. The server switch intercepts the incoming traffic stream and directs each request to a specific server according to some policy. The set of functioning servers to choose from — the *active set* — may grow and shrink dynamically, allowing a site to manage server resources locally to adapt to load changes. The switch isolates clients from internal details of the the service structure, so that the ensemble appears to clients as a single virtual server host that is powerful and reliable. Commercial server switches are available from Nortel (Alteon), Cisco (Arrowpoint), Extreme, Foundry, F5 Networks, Resonate, IBM, and other companies.

2.1 Request Routing Policies

Commercial switches use a variety of request routing policies for HTTP and other TCP-based protocols such as FTP. For example, a *server load balancing* (SLB) policy distributes requests evenly across the servers using weighted round robin or by monitoring server status and directing requests to servers with the lightest load, the lowest latency, or the smallest number of active connections. These SLB switches are often called L4 switches because they make server selection decisions only at connection setup time, and examine only the transport (layer 4) headers of the incoming packet stream.

Content routing policies prefer the servers that can handle a given request most efficiently, for example, a server likely to have the requested content in a cache. *URL hashing* is a content-based policy for Web servers that applies a simple deterministic function on the request URL to select a server from the active set. URL hashing is an L7 policy because the switch must parse HTTP (layer 7) headers to determine the URL (some view this as layer 5). Content routing policies are HTTP-specific in commercial L7 server switches; thus they are sometimes called Web switches.

There is a tension between SLB and content routing policies. SLB policies balance load effectively, but they do not preserve locality in the request stream. In particular, SLB policies tend to spew repeat requests for a given object across all of the servers, forcing each server to fetch the object and cache it redundantly. On the other hand, content routing policies preserve this locality by preferring the same server for repeat requests, but they are vulnerable to load imbalances. Recent research [16, 4, 5] has studied this tradeoff in depth, and developed *locality-aware request distribution* (LARD) and related policies to combine the benefits of both approaches. We use the term *URL switching* to encompass URL hashing as well as the more sophisticated LARD strategies when applied in a server switch.

Dynamic Web services and Web-hosted application services often route requests based on the identity of the client. To support this, some HTTP L7 server switches implement *cookie switching*, which extracts one or more named tokens (*cookies*) from an HTTP request. The cookie is typically unique to the client or client session; applying a routing function to the cookie enables a site to evenly distribute customer load across a cluster, while concentrating the load from each customer on a single server or a small set of servers. This preserves locality of server access to client-specific content such as user profiles, shopping carts, mailboxes, or accounts. Cookie switching also enables the server site to differentiate customer service quality levels.

Some server switches can also enable "transparent" Web caching through *interception proxies*. These switches intercept HTTP request traffic to arbitrary IP addresses as it enters the network from a client, redirecting requests to a proxy cache or cache array. Transparent caching enables an ISP to benefit from the bandwidth savings of Web caching without relying on users to configure their software to use the ISP's proxies.

2.2 Redirection in the Internet Architecture

Server switching is controversial because the Internet architecture standards implicitly assume that each IP datagram is addressed and delivered to a specific and uniquely defined end host. The concept underlying virtualized services is that clients address each request to a logical service endpoint, which may be served by many hosts.

For example, interception proxies clearly violate the Internet architecture as it is currently defined (e.g., by RFC 1122 [8]). In particular, they hijack IP packets addressed to the server IP host and generate responses that masquerade as the server. A connectivity provider (ISP) may interpose an interception proxy unilaterally, without the knowledge or consent of either the client or the server. One justification for interception proxies is that HTTP 1.1 standards allow the server to control or disable caching of any content object in conforming proxy caches; thus an HTTP server that does not disable caching could be viewed as implicitly permitting proxy caches to act as agents of the server.

One can reconcile server switching for server clusters with the assumptions of the Internet architecture is to view a server cluster with a virtual IP address as a single virtual "host" — effectively a multicomputer with internal policies for handling network flows addressed to it. The redirecting switch acts as an extension of the service, and must reside (logically) at the service edge of the connection, rather than in the "middle" of the network. For example, a key constraint is that a redirecting switch must receive all packets from a given client connection initiated through that switch, since the switch may maintain state associated with the connection. From the client's perspective, failure of the switch (or dynamic routing of the packet flow around the switch) is indistinguishable from failure of a server host; the connection is dropped, and the service protocol must recover by re-initiating the connection.

2.3 Alternatives to Server Switching

Server switching is only one of several ways to virtualize a service. In *DNS redirection* (e.g., [14]) the target site's DNS server selects from multiple server host IP addresses during domain name lookup as the client first connects to the service. DNS lookup is now widely used as a "hook" to redirect HTTP connections to wide-area CDN caches or content replicas. However, DNS redirection constrains the possible routing policies because service request state (e.g., HTTP URL and cookies) is absent from the DNS lookup request. In *dynamic URL rewriting* a Web server transforms URL links embedded in objects returned to a client. By modifying the domain names in these URLs, the service can induce different clients to direct their subsequent requests through those URLs to different servers, distributing the request load without using a server switch.

Server switching offers more control over the server selection function than these alternatives, and it is fully transparent to clients. DNS redirection and URL rewriting both expose the server selection choices to the clients, who then send requests directly to the selected server. The client may cache the selection choice in a local DNS cache or demand-side Web proxy, and may reuse the same mapping repeatedly until the cache entry expires. If other local clients share this cache then they may direct their requests to the same server host without offering the site an opportunity to select different servers for their requests. While the service has some control over the cacheability of these mappings, restricting caching can significantly increase the request load at the servers.

In contrast to these alternatives, server switching interposes server selection on each request or each connection, making it more responsive to rapid changes in load or server status. Since server switching virtualizes "on the wire", a client cannot circumvent or even observe the request switching choices. However, server switching for high-performance server sites involves proprietary switch hardware and firmware that can redirect incoming requests at wire speed. Good performance is particularly difficult to achieve for URL switching and other content routing policies.

2.4 Persistent Connections

It is still not widely recognized that many advanced server switching schemes in use today are incompatible with persistent connections in the HTTP 1.1 standard. All server switches on the market essentially perform *connection switching*; they select a target server at connection setup time, i.e., by sniffing TCP SYN packets. Common use of HTTP 1.0 sets up a separate connection for each request, so connection switching is sufficient to route each request independently. However, connection setup is expensive, and short connections complicate congestion management. For this reason, HTTP 1.1 implementations amortize setup costs by pipelining multiple requests to the same service over the same persistent connection [15].

The problem with persistent connections is that server switches cannot independently redirect different requests arriving on a single connection to different servers. In particular, it is more difficult to route requests with URL switching to improve server cache performance. Some content switches now on the market support both URL switching and HTTP 1.1, but they route all requests on each connection to the server selected to handle the first request arriving on the connection. This sacrifices any cache affinity benefit from URL switching.

Aron et. al. [4], in a follow-on to the LARD study [16], discuss this problem in detail and propose a solution using server-to-server transfers of requested data among the back ends. This approach is similar to cooperative caching [3, 12] of content objects among the servers, a solution found to be inferior in the original LARD study. The authors based these results on identical trace-based simulation models, except that the second study uses different parameters: it assumes a fixed per-request CPU cost for HTTP processing and 40% lower per-byte overhead for network transfers. The second paper also shows that an approach using TCP handoff of persistent connections among back-end servers may be competitive with front-end server switching. However, the study does not explore sensitivity of this scheme to the connection migration cost. A more recent study of TCP connection migration [17] suggests that this cost is too high for request distribution in Web server clusters, although it is a useful approach for failover.

In short, the relative merits of the various solutions to the persistent connection problem for fine-grained request routing — such as URL switching and other content routing policies — are still open to debate. However, Section 3 and Section 4 explore factors that may render URL switching policies less relevant for tomorrow's Web services than they were in 1998, when the traces used in the LARD studies were collected. Even so, the fact remains that persistent connections fundamentally limit the range of request routing policies that a server switch can support with current transport protocols and switch technology. Section 6 returns to this issue in the context of IP storage virtualization.

3 The Trickle-Down Effect

The Web research community now accepts that requests to static Web objects follow a Zipf-like popularity distribution [13, 9, 19]. The number of requests to the ith most popular document is proportional to $1/i\hat{}\alpha$, for some α. Thus a large majority of requests target the most popular sites and the most popular objects, but the distribution has a long, heavy tail of less popular objects.

One implication of the Web's Zipf-like behavior is that caching is highly effective for the most popular static (cacheable) objects, assuming that popularity dominates rate of change [19]. The last two years have seen an explosion of growth in Web caching and content delivery (CDN) cache infrastructure. Key developments include the emergence of Akamai and other commercial offerings in the CDN space, growing use of supply-side *surrogate* ("reverse") proxy caching among hosting providers such as Exodus, and aggregation of content consumers into large ISPs employing demand-side proxies.

A key impact of these trends is that Web server sites increasingly receive only the miss stream from these caches, presumably shifting their dominant loads from popular static objects to less popular objects and dynamic content. We call this the *trickle-down effect* [10]. This section considers the implications for server request distribution for static content; Section 4 discusses the demands on server switching for dynamic content.

In the absence of caching, Zipf request patterns are difficult for a server cluster to handle effectively. Because the requests are so heavily skewed to the most popular objects, load imbalances are likely to result from any request routing policy that is based on a fixed assignment of objects to servers (e.g., URL hashing). On the other hand, any request routing policy that

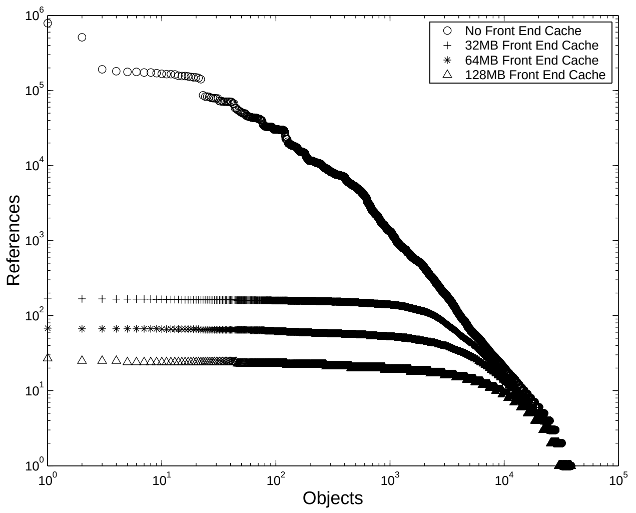

Figure 1. Effect of front-end caching on object popularity distribution in the 1998 IBM trace (log-log scales).

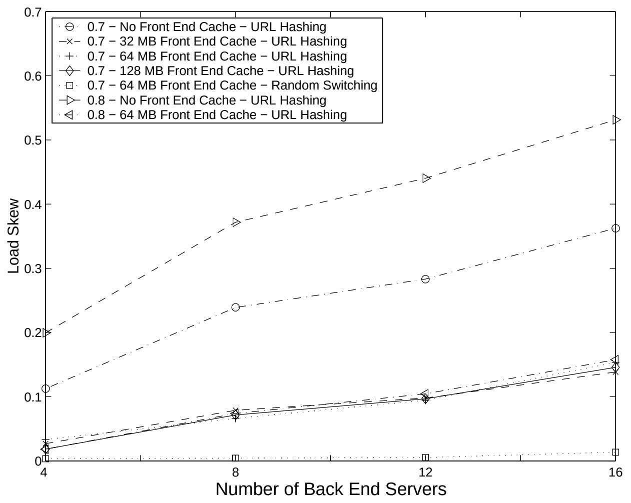

Figure 2. Server load skew from request distribution (synthetic traces: $\alpha = 0.7$ and $\alpha = 0.8$).

is not content-aware (e.g., SLB) does not benefit fully from caching the popular objects in server memory, even if it balances the load.

This observation motivated the LARD strategy for request distribution in server clusters [16], as discussed in Section 2. A LARD server switch maintains a cache of recently requested URLs. If a request URL misses in the cache, then LARD routes the request using an SLB policy. If the request hits, then LARD routes it to a server that recently fetched the URL, if one exists that is not overloaded. Most commercial content switches do not yet support LARD, despite the advantages of this hybrid strategy. Support for any form of URL switching is relatively recent in commercial switches, and LARD's cached mapping of URLs to servers is shared across all switch ports, which may have slowed its adoption.

Caches and CDNs significantly change the properties of the request stream — or trickle — that filters through to server sites. In particular, references to popular objects — the primary sources of both load skew and locality — are largely absent, assuming that those objects are cached effectively. To illustrate, Figure 1 shows the effect of front-end caching on the object popularity distribution for a 1998 trace from *ibm.com*, which was also used in the LARD studies. We filtered the trace through a simple LRU cache simulator for small cache sizes, then plotted the number of references to each object in each miss trace, as a function of its popularity rank in each trace. As expected, the original distribution shows a pattern approximating a Zipf-like distribution with $\alpha = 0.76$. Even small caches effectively flatten this distribution; the references to popular and moderately popular objects are now evenly balanced, with most references concentrated in the heavy tail.

The trickle-down effect implies that effective caching architectures and CDNs will tend to narrow the performance differences between various request routing policies for Web server clusters. First, load skew caused by popular objects is largely removed from the miss stream, so that a simple content routing strategy (e.g., URL hashing) is less vulnerable to load imbalances. Second, there is less locality left in the miss stream because the caches absorb references to the most popular objects, leaving the heavy tail. Thus content-aware policies have less potential to yield better server memory hit ratios.

To illustrate, Figure 2 shows predicted load imbalances in a server cluster as a function of the server cluster size N. The experiment distributes requests by a simple URL hash, which yields server cache effectiveness comparable to LARD, but is vulnerable to load imbalances. The idle time on the y-axis is predicted by a simple and conservative simulation parameterized similarly to the LARD study. It gives the average CPU idle time of all servers over the interval needed for the cluster to serve the trace at peak throughput (i.e., all client requests are present at the beginning of the experiment). For these experiments we used synthetic input traces (generated by SURGE [7]) to show the effect of varying the Zipf α coefficient.

With no caching, the load skew for URLHASH on the $\alpha = 0.7$ Zipf trace with 8 servers is almost 25% and jumps to over 35% when using 16 servers, confirming the LARD result that load imbalances under URLHASH can significantly reduce cluster throughput, and that load imbalances grow with cluster size. However, these load imbalances are less severe after filtering the trace through caches; for example, a small cache drops the load skew to 7% with 8 servers, even for the more heavily skewed $\alpha = 0.8$ trace. This effect is relatively insensitive to front-end cache size. This indicates that explicit load balancing for content-based routing in Web server clusters is less important in the presence of upstream caching.

This result is conservative in that we use small cache capacities relative to the aggregate content size. However, we also assume that a single cache filters requests from all clients; this exaggerates the trickle-down effect from many proxies serving small populations, but it fairly approximates multiple proxies serving modest populations [19]. It is clear that Web caches and CDNs will reshape the request stream for server clusters to the degree that they are effective at all, and that this will reduce the importance of sophisticated request routing policies. The handling of object updates will become increasingly dominant in determining server performance for cacheable objects. For example, one promising strategy uses a simple SLB switch together with push-caching of modified popular objects to all servers before cached copies of those objects expire in the proxies and CDNs.

This discussion leaves open the question of the role of server switching in the cache arrays themselves. Zipf distributions yield cache hit rates that are logarithmic with population size, and most of the benefits of caching are obtained at populations in the low tens of thousands [19]. Small proxy cache arrays can serve this load easily, and load imbalances are less severe in small clusters. Moreover, on this scale simple cooperation schemes exist to share content across decentralized proxy caches (e.g., [11]). Finally, demand-side edge caches combine objects from all sites accessed by their user populations. These small cache arrays may switch on the target site IP address; this policy is simple and fast, it preserves locality, and it easily accommodates HTTP 1.1 persistent connections.

4 The Dynamic Web

The boom in wide-area caching reduces the demands on Web server clusters to serve static, cacheable content. At the same time, large-scale Web services increasingly incorporate content that is personalized for the client, or generated dynamically by executing an application program on the server. Apache and commercial Web service software such as BEA WebLogic and IBM WebSphere have evolved from simple Web servers into *Web application servers* extensible through CGI, Java server technology, Microsoft's Active Server Pages, and others. These server platforms are growing a new generation of server-based applications delivered to customers as "software rentals", and a new industry segment to host them: Application Service Providers.

Together, these factors suggest that the role of server switching will shift toward request distribution for dynamic content servers. Dynamic servers differ from static content servers in several relevant respects:

- The load burden to process each request shifts away from data movement and toward CPU processing and back-end database access. Dynamic services deliver much of their data to clients in the form of static templates, wrapped around smaller amounts of dynamically generated content using XML/XSL or HTML inlining; content assembly may occur at the network edge. These factors suggest a lower relative penalty for direct back-end server-server interaction — or connection migration — as an alternative to server switching.

- Much of the data driving dynamic services is customer-specific, undermining any advantage from URL switching for requests within a session. Examples include user profiles, shopping carts, mailboxes, or accounts. In these cases, the best way to preserve locality is to assure session affinity or session persistence, in which all requests from a session arrive at the same server, e.g., using cookie switching or dynamic URL rewriting rather than URL switching. This also reduces the cost to manage session encryption keys if the service uses secure sockets (SSL).

- Dynamic services tend to have a multi-tiered structure, in which Web application servers initiate processing for the request at back-end databases or processing nodes. The level of indirection allows content-based request routing featurs to be implemented in software within the Web application server, rather than in a server switch.

These properties suggest a more limited role for content-based switching features — such as URL switching — for HTTP services in the future. However, cookie switching will continue to be important for dynamic services.

5 Server Management

While the role of server switches for HTTP content management may narrow in the future, server switches have an expanding role as a focal point for managing server resources. Increasingly, Internet services are hosted in shared data centers managed by third-party hosting providers. Shared hosting centers offer economies of scale and a potential to dynamically adjust capacity provisioning to respond to request traffic, quality-of-service specifications, and network conditions. Customers (services) rent resources from the hosting provider on a "pay as you go" basis; the hosting provider insulates its customers from unplanned demand surges and unnecessary costs for excess capacity.

Hosting centers face resource management challenges common to any shared computing resource. In particular, an efficient hosting center must determine how much server resource to allocate to multiple independent services with varying resource requirements and traffic loads. Each service consists of a body of data and software, such as a Web application. Service software executes on pools of generic servers with common hardware/software combinations. The hosting center allocates to each service a suitable share of the server pool(s) that it needs to serve its load. The center may use operating system enhancements such as *resource containers* [6] in the server operating systems to ensure performance isolation on shared servers.

Resource provisioning in a hosting center must be continuous and adaptive for maximum efficiency. Individual server components export interfaces to monitor status and reserve resources; the resource manager uses these mechanisms to estimate global service load and react to observed changes in load or resource availability by adjusting resource allotments. The Oceano project at IBM Research is one project addressing these challenges in a comprehensive way. Similar approaches to automatic resource provisioning have been proposed in systems for wide-area distributed services [18].

Request redirection through server switches is a fundamental enabler for dynamic adaptive resource provisioning in hosting centers and other large-scale server clusters. Since service load is driven by a large number of relatively small requests, the resource manager can implement fine-grained resource allocation adjustments by reconfiguring the server switches to balance load across the server resources allotted to each service. To external clients, each service appears as a single virtual server whose power grows and shrinks with request load and available resources.

The role of server switches in this context is to dynamically redirect incoming request traffic to eligible servers in the active set for each service. The switches may use load status to balance incoming request traffic across the servers in the active set. To fully support adaptive resource provisioning, a server switching infrastructure must support dynamic, coordinated adjustments to the active sets for each service.

Adaptive resource provisioning in server clusters leads to several important research challenges. For example, it motivates load estimation and feedback mechanisms to dynamically assess the impact of resource allotments on service quality, and a richer framework for Service Level Agreements (SLAs) to specify tradeoffs of service quality and cost. This would enable data centers to degrade service intelligently during unanticipated load spikes, resource failures, or other resource constraints.

6 Server Switching Beyond HTTP

Virtualization using redirecting server switches is a powerful technique for building scalable cluster-based network services, while isolating clients from the details of service structure. What is the potential for server switching for Internet service protocols other than HTTP? This section discusses the role of virtualizing server switches for IP-based storage.

Network storage services share many of the demands and characteristics of Web services, including a fine-grained request/response communication structure. Demand for large-scale storage services is growing rapidly along with the Web, driven not just by Web services but also by new content capture devices (e.g., digital cameras), new digital content standards such as MP3, and new sources of scientific and business data. Moreover, there is increasing interest in IP-based network storage solutions as Gigabit Ethernet enters widespread deployment and 10 Gigabit Ethernet approaches its launch. With these standards the network is the fastest path to external storage in IP LANs, and storage outsourcing over the network is now cost-effective. IP storage is competitive with FibreChannel from a price/performance standpoint, and will soon advance with a new crop of devices based on the emerging iSCSI block storage standard. All of these factors are feeding demand for scalable IP-based network storage solutions.

In our work on the Slice storage system [2, 1] we are using server switching techniques to construct a scalable "virtual storage appliance". Slice uses content routing to distribute file service traffic across a dynamic ensemble of servers and network-attached block storage devices. An important goal of Slice is to automatically balance the load across the ensemble, without imposing user-visible file volume boundaries or a volume management burden on administrators. The architecture achieves this goal by interposing a request switching filter — called a *μproxy* — along each client's network path to the storage service. The Slice prototype supports the Network File System (NFS) V3 protocol, which is currently the primary storage access protocol in the Internet.

The Slice μproxy was designed to reside in a server switch. Like an HTTP server switch, the μproxy intercepts NFS requests addressed to the virtual NFS server, and routes the request to a physical server by applying a request routing function. The μproxy rewrites fields in the requests and responses to redirect requests and to represent the ensemble as a unified file service.

Experiments with the Slice prototype show the potential of server switching for IP storage. The Slice prototype scales to a terabyte of storage with aggregate disk access bandwidths approaching one gigabyte per second. It delivers scalable request throughput (NFSOPS) on SPECsfs97, an industry-standard NFS benchmark, demonstrating compatibility with standard NFS V3 clients.

The Slice experience illustrates three important properties for server switching targeted to NFS V3, which may extend to other storage protocols:

- **Effective content partitioning.** Most NFS V3 operations apply to a single content item — a directory, name entry, or file — which is readily determined by examining the request type and arguments. These arguments include the NFS file ID (file handle); Slice places keys in the file handle to index content routing tables in the μproxy. NFS does not impose unnecessary response ordering constraints that might prevent operations on different content items from proceeding in parallel on different servers. Although some operations that cross content partition boundaries are more expensive, they are relatively rare in practice.

- **Simple request routing.** The content routing policies in the Slice μproxy are simple and fast. The amount of data that the μproxy must maintain and examine in each request is tightly bounded, and traversal of variable-length strings is never required as in HTTP URL switching or cookie switching. While file system protocols are more complex than HTTP, most of these can be resolved without cluttering the μproxy, by pushing more responsibility for coordination into the back-end servers. File system issues faced by the Slice architecture include reliable block allocation, reconfiguration of the active server set, and transparent scaling of the directory name space while preserving ordering and reliability guarantees.

- **Scalable and robust switching functions.** The Slice μproxy maintains a limited amount of state for each client session. Because this state is soft, transient failure of a μproxy does not compromise correctness. Because the μproxy requires no state that is shared across clients, it is not a barrier to scalability. The μproxy functions are freely replicable: each port of a server switch may function autonomously, and client traffic may be partitioned across multiple switches.

The key limitation of the Slice μproxy architecture is that it is subject to the persistent connection problem discussed with reference to HTTP 1.1 in Section 2.4. The Slice prototype works with NFS/UDP, but if clients use a connection-based transport such as TCP then a switch-based μproxy cannot route multiple requests arriving on the connection independently. This is necessary for any content routing policy with persistent connections. One alternative is to run the μproxy as a full proxy acting as a terminal endpoint for connections to the clients and servers, but this entails significant buffering and protocol overhead, which is unacceptable given the performance demands for network storage.

NFS V4 and other important IP services assume that their transport layers provide reliable and sequenced delivery with congestion control, such as TCP. How can we support these properties while accommodating server switching with persistent connections? Achieving this goal requires a transport protocol that explicitly permits a more decentralized notion of what constitutes a connection "endpoint", in which multiple IP hosts join together in an ensemble that appears to the connection peer as a single virtual host. We call this *Anypoint* communication, because the switch may route traffic on the connection to any of several end nodes, at the discretion of a service-specific routing policy. A key aspect of an Anypoint-capable transport is that connection endpoint state and functions are distributed between the edge switch and the physical end nodes, and some protocol properties and guarantees are defined as *end-to-edge* rather than *end-to-end.* For example, to perform fine-grained request routing, a server switch must be capable of distinguishing independent request and response frames in the traffic stream, and it must be permitted to weaken end-to-end ordering guarantees for frames routed to different end nodes.

Explicit support for Anypoint communication would enable fine-grained request routing for HTTP 1.1, NFS, and other service protocols using the transport. It could also be a basis for a general approach to extending services by interposing "wrapper" functions in the network, transparently to clients. It is an open research problem to define such a transport model, and to evaluate its benefits relative to other virtualization approaches for the next generation of IP-based services.

7 Conclusion

This paper explores interrelated factors shaping the role of server switching and request distribution for server clusters. It outlines several factors affecting the demands on mechanisms and policies for server switching in the Web.

Wide-area cache infrastructures and Content Delivery Networks are absorbing a larger share of requests for cacheable static objects. Simulation results suggest that this creates a "trickle-down effect" undermining the key sources of load skew and locality in Web server clusters. This reduces the potential benefit from content-based server switching — URL switching — at the same time that HTTP 1.1 deployments invalidate the assumptions on which URL switching implementations are based.

Dynamic content constitutes an increasing share of server cluster traffic, as a result of the trickle-down effect and because the most popular Web sites are incorporating more dynamic content. Web-based application services will accelerate the shift to server clusters dominated by dynamic content. This will also affect the choices for server switching policies: a combination of cookie (session) switching and simple server load balancing may offer the best request routing solution for tomorrow's Web clusters. These approaches are well-understood and scalable, and they are compatible with HTTP 1.1 persistent connections. While content routing policies such as URL hashing may continue to be attractive, they must be re-evaluated in light of recent shifts.

In addition to these changes, server switching may play an increasingly prominent role in dynamic resource provisioning for shared hosting centers. To adequately support this need, server switches must support flexible reconfiguration.

Experience with server switching in the Web shows that network-based request redirection is a powerful technique for virtualizing services. In the future, server switches may play a role in virtualizing IP storage and other services. Experience with the Slice storage system project has shown that server switches can virtualize an NFS service in the network, enabling a scalable storage architecture that is compatible with existing NFS clients. However, supporting this approach in a fully general way requires addressing the challenge of independently routing multiple requests arriving on the same transport connection, which is also necessary to support content routing policies over HTTP 1.1 with persistent connections.

Acknowledgements

The ideas and research issues in this paper are a product of collaborations with Darrell Anderson, Ron Doyle, Syam Gadde, Richard Kisley, Amin Vahdat, and Ken Yocum.

References

[1] Darrell C. Anderson and Jeffrey S. Chase. Failure-atomic file access in an interposed network storage system. In *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC)*, August 2000.

[2] Darrell C. Anderson, Jeffrey S. Chase, and Amin M. Vahdat. Interposed request routing for scalable network storage. In *Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI)*, October 2000.

[3] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. Serverless network file systems. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 109–126, December 1995.

[4] M. Aron, P. Druschel, and W. Zwaenepoel. Efficient support for P-HTTP in cluster-based Web servers. In *In Proceedings of USENIX'99 Technical Conference*, 1999.

[5] Mohit Aron, Darren Sanders, Peter Druschel, and Willy Zwaenepoel. Scalable content-aware request distribution in cluster-based network servers. In *In Proceedings of the USENIX 2000 Technical Conference*, 2000.

[6] Gaurav Banga, Peter Druschel, and Jeffrey C. Mogul. Resource containers: A new facility for resource management in server systems. In *Third Symposium on Operating Systems Design and Implementation*, February 1999.

[7] Paul Barford and Mark E. Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proceedings of Performance '98/ACM SIGMETRICS '98*, pages 151–160, June 1998.

[8] Bob Braden. Internet Engineering Task Force, Network Working Group, RFC 1122: Requirements for Internet hosts – communication layers, 1989.

[9] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of IEEE Infocom '99*, March 1999.

[10] Ronald P. Doyle, Jeffrey S. Chase, Syam Gadde, and Amin M. Vahdat. The trickle-down effect: Web caching and server request distribution. In *Proceedings of the Sixth International Workshop on Web Caching and Content Distribution*, 2001.

[11] Li Fan, Pei Cao, Jussara Almeida, and Andrei Broder. Summary Cache: A scalable wide-area Web cache sharing protocol. In *Proceedings of ACM SIGCOMM98*, September 1998.

[12] Michael J. Feeley, William E. Morgan, Frederic H. Pighin, Anna R. Karlin, and Henry M. Levy. Implementing global memory management in a workstation cluster. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, 1995.

[13] Steve Glassman. A Caching Relay for the World Wide Web. In *First International World Wide Web Conference*, 1994.

[14] Eric Dean Katz, Michelle Butler, and Robert McGrath. A Scalable HTTP Server: The NCSA Prototype. In *First International Conference on the World-Wide Web*, 1994.

[15] J. Mogul. The case for persistent HTTP connections. In *In Proceedings of SIGCOMM*, pages 299–313, 1995.

[16] Vivek S. Pai, Mohit Aron, Gaurav Banga, Michael Svendsen, Peter Druschel, Willy Zwaenopoel, and Erich Nahum. Locality-aware request distribution in cluster-based network servers. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1998.

[17] Alex C. Snoeren, David G. Andersen, and Hari Balakrishnan. Fine-grained failover using connection migration. In *Proc. of the Third Annual USENIX Symposium on Internet Technologies and Systems (USITS)*, 2001.

[18] Amin Vahdat, Thomas Anderson, Michael Dahlin, Eshwar Belani, David Culler, Paul Eastham, and Chad Yoshikawa. WebOS: Operating System Services for Wide-Area Applications. In *Proceedings of the Seventh IEEE Symposium on High Performance Distributed Systems*, Chicago, Illinois, July 1998.

[19] Alec Wolman, Geoff Voelker, Nitin Sharma, Neal Cardwell, Anna Karlin, and Henry Levy. On the scale and performance of cooperative Web proxy caching. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, December 1999.