Chapter 10

Simultaneous Proxy Evaluation (spe.tex)

10.1 Introduction

While Chapters 8 and 9 considered the value of simulation for estimating performance, they are primarily of value to researchers trying to test ideas. However, simulators are generally unable to evaluate the performance of *implemented* systems. Simulation often abstracts away many of the implementation-specific details — such as memory and disk speeds, network interrupts, processor characteristics, etc. — which become important when they vary between systems. Indeed, this is one reason why benchmarks are common — they provide a mechanism to test and compare performance on something close to a real-world workload, hopefully exercising those details in the process.

This chapter presents a new architecture for the evaluation of proxy caches. Initially, it grew out of research in techniques for prefetching in Web caches. In particular, we found that existing mechanisms for the evaluation of proxy caches were not well suited to prefetching systems. Objective evaluation is paramount to all research, whether applied or academic. Since this is certainly relevant when exploring various approaches to prefetching, we considered the range of existing proxy evaluation methods, and found that they each had distinct disadvantages. This led to the design of the more general architecture described below, and to its implementation which will be described in Chapter 11.

Our evaluation method, the Simultaneous Proxy Evaluation (SPE) architecture, reduces problems of unrealistic test environments, dated and/or inappropriate workloads, and allows for the evaluation of a larger class of proxies. It objectively measures proxy response times, hit rates, and staleness by forming a wrapper around the tested proxies to monitor all input and output. In the next section we motivate the need for the

SPE architecture by describing the space of proxy evaluation methodologies. We then specify our new evaluation architecture, and comment on implementation issues. We conclude the chapter by describing a set of sample evaluation tasks that would utilize it, and relate SPE to other work.

10.2 Evaluating Proxy Cache Performance

A variety of proxy caches exist in the market. (Web sites such as `web-caching.com` [Dav02b] provide a detailed list of products and vendors.) The features and performance provided sometimes vary more than the marketing literature would suggest. As more and more Internet users access the Internet through proxy servers everyday, these factors play a significant role in determining the perceived quality of the Internet.

Therefore, evaluating the various aspects of proxy performance is essential. There are, in fact, many characteristics of proxy caches that could be evaluated:

- The peak and average number of requests that can be serviced by the proxy cache per unit time.
- The object hit ratio — the number of objects that are serviced from the proxy's cache against the total number of objects served by the proxy.
- The byte hit ratio — the sum of the bytes served from the proxy's cache against the total bytes served by the proxy.
- The average response time as experienced by the user.
- Bandwidth used by the proxy. In addition to being affected by hit rates, a prefetching proxy is likely to use more bandwidth than a non-prefetching proxy.
- Robustness of a proxy.
- Consistency of the data served by a proxy.
- Conformance to HTTP standards.

Many of these are implementation characteristics, and thus can only be evaluated by techniques that work with complete systems. The SPE architecture (described in the next section) is designed to work at this level and is concerned with many of the characteristics of proxy cache performance: bandwidth usage, retrieval latencies, robustness and consistency. As a result, it provides benefits complementary to existing techniques described in Chapter 7. It is, however, the first mechanism to allow for the simultaneous black-box comparison of competing mechanisms on live, real-world data in a real-world environment.

10.3 The SPE Architecture

The SPE architecture can use actual client requests and the existing network to evaluate multiple proxies. It records timing measurements to calculate proxy response times and can compute page and byte hit rates. In summary, it forms a wrapper around competing proxies and produces logs that measure external network usage as well as performance and consistency as seen by a client. By simultaneously evaluating multiple proxies, we can utilize a single, possibly live, source of Web requests and a single network connection to provide objective measurements under a real load. Importantly, it also allows for the evaluation of content-based prefetching proxies, and can test for cache consistency.

10.3.1 Architecture details

In the SPE architecture, clients are configured to connect to a non-caching proxy, the *Multiplier*. Requests received by the Multiplier are sent to each of the proxy caches which are being evaluated. Each attempts to satisfy the request. To do so, each either returns a cached copy of the requested object, or forwards the request to a parent proxy cache, the *Collector*. The Collector then sends a single copy of the request to the origin server for fulfillment when necessary (or potentially asks yet another upstream proxy cache for the document, as shown back in Figure 2.5). By forcing the proxies to use the Collector, we can prevent two types of problems that might otherwise arise: 1) an increase in traffic loads on the network or at the origin server, and 2) side-effects from

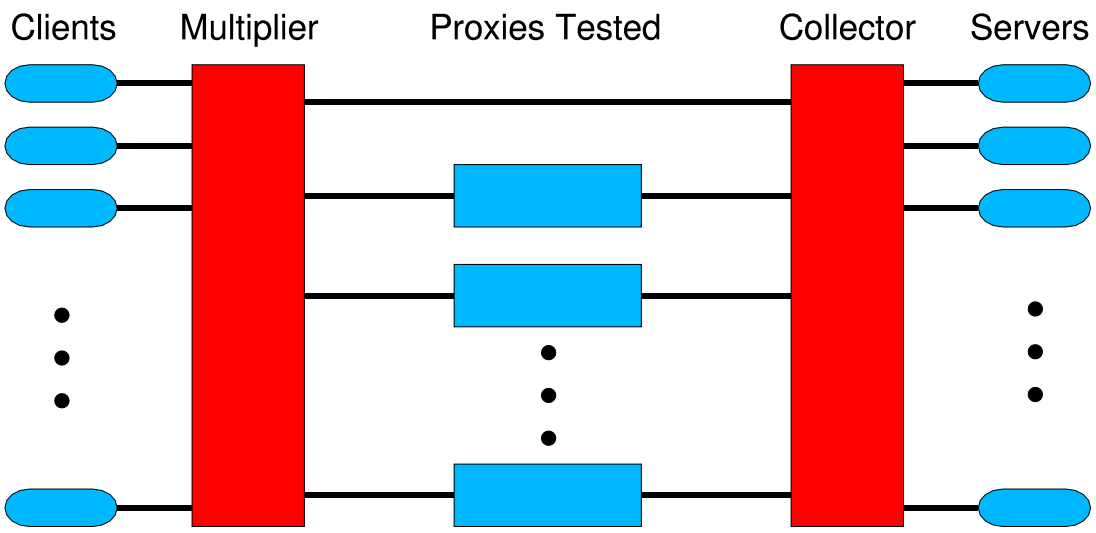

Figure 10.1: The SPE architecture for online evaluation of competing proxy caches.

multiple copies of non-idempotent requests [FGM+99, Dav01a]. The Multiplier also sends the requests directly to the Collector to use the responses to service the client requests and validate the responses from the test proxies. See Figure 10.1 for a diagram of this architecture.

Thus the Multiplier does not perform any caching or content transformation. It forwards copies of each request to the Collector and the test proxies, receives responses, performs validation and logs some of the characteristics of the test proxies. The clients view the Multiplier as a standard HTTP proxy.

Only minimal changes are required in clients, the proxies, or the origin servers involved in the evaluation. The clients need to be configured to connect to the Multiplier instead of the origin server and with popular browsers this is straightforward. The test proxies need to be configured with the Collector as the parent proxy. No changes are required to the Web servers which ultimately service the requests.

When a user of a proxy-enabled browser clicks on a link, the browser sends the request for that URL to the proxy. The proxy, in this case the Multiplier, takes the request and sends a duplicate request to each of the proxies under evaluation and directly to the Collector. Some proxies may have the object in the cache, in which case they return the object quickly. Others may experience a *cache miss*, or need to verify the contents of cache, and so have to send a request through the Collector.

Without a proxy Multiplier, client browsers would need modifications to send requests to each proxy under evaluation. Instead, one can configure off-the-shelf browsers to use the Multiplier as their proxy. In addition to request duplication, the Multiplier calculates request fulfillment times for each of the proxies being evaluated. The Multiplier also determines which response is passed back to the client — either arbitrarily, or by an algorithm such as by first response.[1]

Each of the proxy caches being evaluated can be treated as a black-box — we do not need to know how they work, as long as they function as a valid HTTP/1.0 [BLFF96] or 1.1 [FGM+99] proxy cache. This is helpful in particular for commercial proxies, but in general eliminates the requirement for either detailed knowledge of the algorithms used or source code which is needed for simulation and specialized code additions for logging [MR97] respectively. Typically the proxy caches would run on different machines, but for simple tests that may not be necessary. Alternatively, the same software with different hardware configurations (or vice versa) can be tested in this architecture.

In order to prevent each proxy cache from issuing its own request to the destination Web server, we utilize a proxy Collector which functions as a cache to eliminate extra traffic that would otherwise be generated. It cannot be a standard cache, as it must eliminate duplicate requests that are normally not cacheable (such as those resulting from POSTs, generated by cgi-bin scripts, designated as private, etc.). By caching even uncacheable requests, the Collector will be able to prevent multiple requests from being issued for requests that are likely to have side effects such as adding something to an electronic shopping cart. However, the Collector must then be able to distinguish between requests for the same URL with differing arguments (i.e., differing fields from otherwise identical POST or GET requests).

The Collector returns all documents with the same transmission characteristics as when fetched from the destination server. This includes the amount of time for initial connection establishment as well as transmission rate. By accurately simulating a

[1] Since the first response is likely to be a cached response, it will affect (positively) the users' perceived performance during the test. On the other hand, the response might be stale or erroneous, so a conservative approach would be to send back the response obtained from the Collector directly.

"miss", the Collector 1) enables the black-box proxy to potentially use the cost of retrieval in its calculations, and 2) allows the Multiplier to record response times that more accurately reflect real-world performance. Without it, any requests for objects that are in the Collector's cache will inappropriately get a performance boost.

Each proxy tested thus sees a single proxy (the Multiplier) as client, and a single proxy (the Collector) as parent. If the tested proxies only cache passively, then the overall bandwidth used by the system should be exactly that which is needed to serve the client requests. If one or more proxies perform active revalidation or prefetching, then the overall bandwidth used will reflect the additional traffic.

The architecture attempts to tolerate problems with the test proxies. Problems like non-adherence to protocol standards, improper management of connections, and the serving of stale responses can be caught and logged by the Multiplier. Note that while the SPE architecture is designed to preserve safety for passive proxy caches (allowing just one copy of a request to go to the origin server), it cannot provide guarantees of safety for non-demand requests that use the standard HTTP/1.1 GET mechanism [Dav01a].

To summarize, traditional evaluation of caching algorithms has been via trace-based simulations. We propose an online method, useful because it:

- takes existing proxy caches and uses them as black-boxes for evaluation purposes.
- can test the proxy caches on real-world data.
- eliminates variation in measured performance due to changes over time in network and server loads.
- eliminates variation in measured performance due to "dated" request logs.
- allows for the evaluation of content-based prefetching systems without having to locally mirror all pages referenced in a trace or to generate an artificial workload complete with internal links.
- allows for the evaluation of prefetching systems that might fetch pages never seen by the user (and thus not included in a traditional trace).

- allows for the measurement of consistency by comparing objects returned from proxies with that fetched directly.
- simultaneously compares different proxies on the same request stream.
- can evaluate cache freshness.
- can be used in the environment where the proxy will be used.

We realize that any method has drawbacks. We have identified the following disadvantages of the SPE architecture:

- Evaluating multiple prefetching schemes may place a higher demand for bandwidth on the network. If bandwidth usage is already high, the increased demand may cause congestion.
- Multiple copies of comparable hardware may be needed. At least $n + 1$ systems are needed to compare n different proxy caches.
- The Multiplier and Collector will introduce additional latencies experienced by users. However, our expectation is that this will be less than that of a two-level cache hierarchy (since the Multiplier does not do any caching).
- The proxy Multiplier effectively hides the client IP address. It may be useful for a proxy to perform tasks differently depending on the client, such as predicting future accesses.
- Caches are evaluated independently. Hierarchies are not easily included in the SPE architecture, even though better performance is often claimed when part of a hierarchy.
- SPE tests may not be replicable. This drawback applies to any method using a live workload or network connection.
- The Multiplier and Collector must manage large numbers of connections (discussed further below).

10.3.2 Implementation issues

The SPE architecture described has characteristics that may not be entirely feasible in some circumstances. For example, to eliminate any requirements for user intervention, there should be an existing proxy that can be replaced by the Multiplier (i.e., at the same IP and port address), or the Multiplier and Collector combination could be configured to transparently handle and route HTTP requests. In the earlier description of the SPE architecture, all HTTP requests go through all proxies when not filled by the previous proxy's cache. In particular, all black-box proxy misses go through the Collector. To ensure its use, the Collector should be placed on a firewall so that there is no chance of the proxies being tested getting around it.

To preserve client performance during the test, the Multiplier contacts the Collector directly. It also uses the object returned from the Collector as the standard against which the results of the proxies can be compared for consistency. When a proxy returns a stale object (in comparison) or an error, that can be recorded. By sending a request directly to the Collector, the Multiplier also eliminates the difficulty of determining which response to return to the client, and provides some degree of robustness in case of failure of the proxy caches being tested.

In general, any implementation of the SPE architecture should be as invisible to the participating software as possible. For example, caching connections has been shown to be a significant factor in cache performance [CDF+98], so the Multiplier should support HTTP/1.0 + Keep-Alive and HTTP/1.1 persistent connections to clients for performance, but should only attempt to contact the proxies at the same level at which the client contacted the Multiplier. In this way the Multiplier will attempt to generate the same ratio of HTTP/1.0 and 1.1 requests as it received. Likewise, the Collector may receive a range of request types from the proxies under evaluation depending on their capabilities. For highest fidelity, the Collector would record when it connects to destination Web servers that support HTTP/1.1 and use that information to selectively provide HTTP/1.1 support for requests from the tested proxies.

Unfortunately, the requirements for an accurate Collector are high. If one is unavailable, a traditional proxy cache may need to be used as a Collector, which means it would not attempt to cache the "uncacheable" objects, nor return objects with accurate timing characteristics. This allows for the calculation of hit-rates but not comparative average response time values. Additionally, some proxies may not be able to handle firewalls, so they may attempt to fetch some objects directly. Therefore, instead of passing all requests to the proxies being tested, the Multiplier may need to be configured to fetch non-cachable requests (cgi, forms, URLs containing "?", etc.) directly to ensure that only one copy gets delivered to the origin server.

Other difficulties include implementing a sufficiently robust Multiplier and Collector — for high-load measurements (such as that needed for enterprise-level proxies [MR97]), they need to be able to handle a higher load than the best black-box proxy being tested, and support $k + 1$ times as many connections (where k is the number of black-box proxies). In general, this difficulty is significant — under the SPE architecture, the Proxy and Collector have to manage a large number of connections. As a result, this architecture is not well suited for stress-testing proxies, but instead provides insight to their performance on (perhaps more typical) non-peak workloads.

10.4 Representative Applications

To demonstrate the potential of the SPE architecture, we present the following scenarios. Each exhibits how some aspect of the SPE architecture can be valuable in evaluating alternative proxies.

10.4.1 Measuring hit-rates of black-box proxy caches

Consider first the task of comparing bandwidth reduction across multiple black-box proxy caches, including some that perform prefetching on a believable client load. Since the internal workings may not be known, simulation is not viable, and in order to evaluate the prefetching caches, a realistic workload with full content is desirable. This is a typical evaluation task for corporate purchasing but is also similar to that needed

by researchers evaluating new prototypes.

Properly implemented, the SPE architecture can be used to channel all or part (with load balancing hardware) of the existing, live client request load through the multiple proxies under evaluation. Since both incoming and outgoing bytes to each proxy can be measured, the calculation of actual bandwidth savings on the current workload and network connection can be performed. Additionally, the SPE architecture is sufficient to measure the staleness of data returned by each proxy. Finally, note also that with some loss of generality, a minimal test of this sort can be made without implementing the Collector but instead test the proxies on cachable requests only (with the Multiplier fetching uncacheable requests directly).

10.4.2 Evaluating a cache hierarchy or delivery network

In an early version of this work [Dav99b] we proposed the use of the SPE Multiplier for evaluating the latency benefits of using a cache hierarchy, such as NLANR [Nat02]. In this model, the Collector is not used, and the Multiplier replicates requests, sending them to different places. To test a cache hierarchy, we would send one request to the hierarchy, and the other directly to the origin server. Likewise, to test a content delivery network, one would send the request to the CDN and to the origin server.

There are many concerns with this kind of evaluation, however. Most worrisome is the replication of non-idempotent requests (those causing side-effects) that eventually get back to the origin server. We discuss this issue further in Chapter 12. If that were not an issue (say, for static, highly cacheable objects), then the remaining concerns are in methodology. One must take care in measuring external services as they cannot be restarted and the evaluation itself can affect their performance. For example, by sending the request to an origin server, that request will warm the server, potentially placing the object in a server-side cache, or at least loading the relevant files in a file buffer. Then, in the case that the cache hierarchy or CDN does not have a fresh copy of the content requested, they can get it from the origin at a faster rate than would otherwise be the case. Likewise, servers that employ CDNs to handle delivery of static content may not often serve such content or be tuned to do so, and thus be at a disadvantage

for a comparison which ideally would test equally best cases or average cases on both sides.

10.4.3 Measuring effects of varying OS and hardware

In this task, a single proxy caching solution is used, such as Squid [Wes02]. The question being asked is that of which version of UNIX is best performing, or what hardware is sufficient to get the best performance for a particular organization's workload. If the various installations of Squid are identical, the application-level operations of the cache will be almost identical. Thus, the difference in performance from using an IDE versus SCSI drives, or performance from a stock OS installation versus one that is custom-tuned can be determined. Since SPE allows for each to receive the same request load and generate a single network load, the relative performance of each configuration can be calculated under the current loads and network capabilities of the particular organization.

10.4.4 Evaluating a transparent evaluation architecture

Finally, consider the problem of evaluating the overhead of a transparent evaluation architecture. An implementation of SPE, for example, imposes two additional proxies between client and server (the Multiplier and Collector). Each of these is likely to add some latency to handle connection requests and to copy data. In addition, the Collector portion of SPE provides caching services, and like other proxy caches, has the potential to serve stale data.

Given a SPE implementation and an appropriate (artificial) workload, the SPE implementation could be used to evaluate itself, to find out how much latency it introduces and to attempt to verify the consistency of the Collector implementation.

10.5 Related Work

A number of researchers have proposed proxy cache (or more generally just Web) benchmark architectures (e.g., the Wisconsin Proxy Benchmark [AC98b, AC98a], WebMonitor [AAY97a, AAY97b], hbench:Web [MSC98], and Surge [BC98b]). Some use artificial traces; some base their workloads on data from real-world traces. They are not, however, principally designed to use a live workload or live network connection, and are generally incapable of handling prefetching proxies.

Web Polygraph [Rou02] is an open-source benchmarking tool for performance measurement of caching proxies and other content networking equipment. It includes high-performance HTTP clients and servers to generate artificial Web workloads with realistic characteristics. Web Polygraph has been used to benchmark proxy cache performances in multiple Web cache-offs [RW00, RWW01].

Koletsou and Voelker [KV01] built the Medusa Proxy, which is designed to measure user-perceived Web performance. It operates similarly to our Multiplier in that it duplicates requests to different Web delivery systems and compares results. It also can transform requests, e.g., from Akamaized URLs to URLs to the customer's origin server. The primary use of this system was to capture the usage of a single user and to evaluate (separately) the impact of using either: 1) the NLANR proxy cache hierarchy, or 2) the Akamai content delivery network. The first test is quite similar in concept to that made by Davison [Dav99b] as a partial application of the SPE architecture, which also explored the potential for use of the NLANR hierarchy to reduce retrieval times, but which assumed the use a small local proxy cache in between the browser and the hierarchy. While acknowledging the limitations of a small single-user study, their paper also uses a static, sequential ordering of requests – first to the origin server, and then to the NLANR cache. The effects of such an ordering (such as warming the origin server) are not measured. Other limitations of the study include support only for non-persistent HTTP/1.0 requests and fixed request inter-arrival time of 500ms when replaying logs.

Liston and Zegura [LZ01] also report on a personal proxy to measure client-perceived

performance. Based on the Internet Junkbuster Proxy [Jun00], it measures response times and groups most requests for embedded resources with the outside page. Limitations include support for only non-persistent HTTP/1.0 requests, and a random request load.

Liu *et al.* [LAJF98] describe experiments measuring connect time and elapsed time for a number of workloads by replaying traces using Webjamma [Joh98]. The Webjamma tool plays back HTTP accesses read from a log file using the GET method. It maintains a configurable number of parallel requests, and keeps them busy continuously. While Webjamma is capable of sending the same request to multiple servers so that the server behaviors can be compared, it is designed to push the servers as hard as possible. It does not compare results for differences in content.

While all of the work cited above is concerned with performance, and may indeed be focused on user perceived latencies (as we are), there are some significant differences. For example, our approach is designed carefully to minimize the possibility of unpleasant side effects — we explicitly attempt to prevent multiple copies of a request instance to be issued to the general Internet (unlike Koletsou and Voelker). Similarly, our approach minimizes any additional bandwidth resource usage (since only one response is needed). Finally, while the SPE Multiplier can certainly be used for measuring client-side response times if placed adjacent to clients, it has had a slightly different target: the comparative performance evaluation of proxy caches.

10.6 Summary

In this chapter we presented an online evaluation method, called the SPE (Simultaneous Proxy Evaluation) architecture, which simultaneously evaluates different caching strategies on the same request stream. This is in contrast to existing evaluation methodologies that use less realistic test environments and potentially out-of-date or irrelevant workloads. Importantly, it also allows for the evaluation of content-based prefetching proxies, and can test for cache consistency. To help motivate our unique approach to cache evaluation, we have provided some sample applications to demonstrate the

potential of the proposed architecture.

The SPE architecture has been designed in particular for use in an online environment with real traffic. However, it can still be useful as an evaluation technique with artificial workloads or trace logs as in the experiments reported here. In fact, artificial workloads are often useful for evaluating systems under workloads that cannot easily be captured or are in excess of current loads. In this way one can capture the statistical properties of specialized workloads and still retain the benefits of simultaneous evaluations on a live network.

The SPE architecture is particularly useful for evaluation of black-box proxies and prefetching systems. It can be used to help validate manufacturer claims and provide comparisons between systems using live networks. Evaluation systems that implement the SPE architecture will be welcome additions to the set of measurement tools available to cache designers and network engineers alike. In the next chapter we describe such an implementation.