Locality in a Web of Streams

Rodrigo Fonseca[‡] Virgílio Almeida[‡] Mark Crovella[§]

[‡] Department of Computer Science
Federal University of Minas Gerais
Brazil

[§] Department of Computer Science
Boston University
USA

May 19, 2003

The dramatic growth of the World Wide Web is well known to the computing community. Furthermore, beyond size (measured in terms of content, users, or number of servers), the Web has also grown considerably in complexity since its inception. Broadly speaking, the Web has evolved from a simple collection of clients and servers to a much more complex arrangement, including new kinds of components with roles that differ from those of either clients or servers.

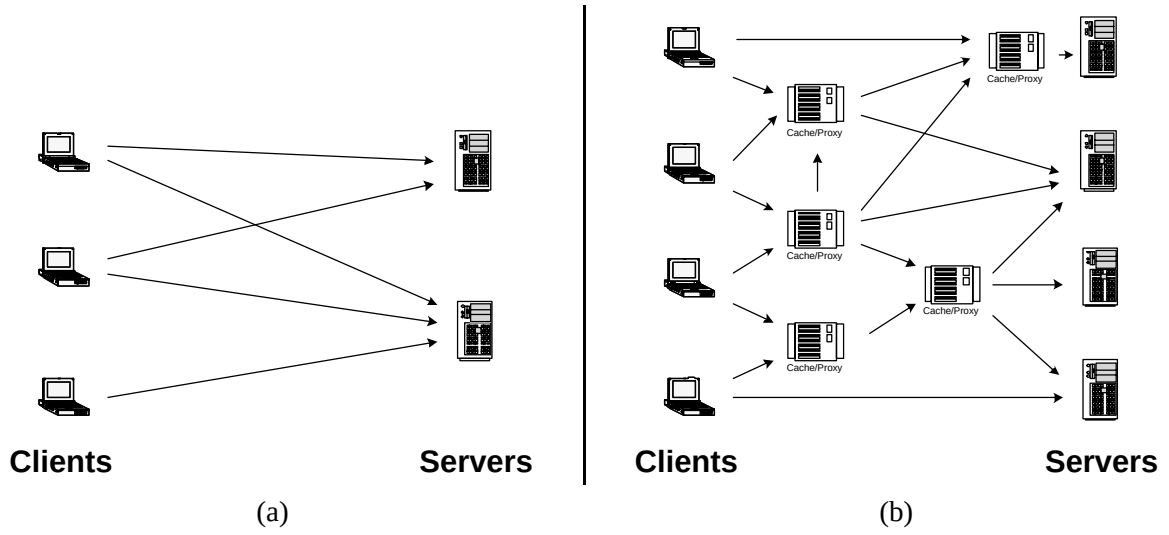
Figure 1(a) represents the way in which the Web operated in its earliest form, circa 1994. In this simple system, each client emits a stream of requests, which is split among the servers being accessed; and each server receives a set of requests for only the objects that it serves. In contrast, the complexity of the modern Web is better represented by the arrangement in Figure 1(b). The figure shows that nowadays, a request made by a client may be intercepted by a wide variety of intermediaries — caches or proxies.[1] First of all, most clients are equipped with internal caches, meaning that some requests made by users can be satisfied from local storage without generating network traffic. Furthermore, any request that is made by a proxy may itself be intercepted by later proxies. In some cases (as in the upper right part of the figure) caching proxies are placed directly in front of servers, as "server accelerators." In other cases, proxies may be aware of each other's presence and intentionally redirect requests to other proxies (as in the center of the figure). Finally, proxies may be organized into hierarchies (as in the lower part of the figure) in which a number of proxies forward requests to a "higher level" proxy.

As an illustration: the original HTTP/1.0 specification (RFC 1945), published in 1996, is 60 pages in length; three years later, the HTTP/1.1 specification (RFC 2616) had grown to 176 pages in length. This dramatic increase in the complexity of the HTTP protocol was partially due to the proliferation of the new kinds of Web devices shown in Figure 1(b) and the new semantic relationships they make possible.

This additional complexity has also created new problems from the standpoint of Web performance evaluation, capacity planning, and system configuration. These problems tend to concern the relationship of clients, caches, and servers as a whole system. Such problems include: Where should caches be placed in the network? How large should cache sizes be at different locations in the network? When is inter-cache cooperation likely to be effective? How many caches should be deployed in a given network? When should caches be organized into hierarchies?

Unfortunately, most of the work to date on performance evaluation of Web systems has focused on clients, proxies, or servers in isolation. Isolated study of individual components is ineffective at answering the kinds of new, system-wide engineering questions just listed. Without answers to such questions, system designers and capacity planners resort to heuristics and intuition for guidance — in settings that often involve

[1] We use the term proxy to denote any intermediate node in the system (including caches) that can make requests on behalf of clients, deliver Web objects on behalf of servers, or do both.



(a) (b)

Figure 1: Flow of Requests in (a) The Early Web; (b) The Modern Web.

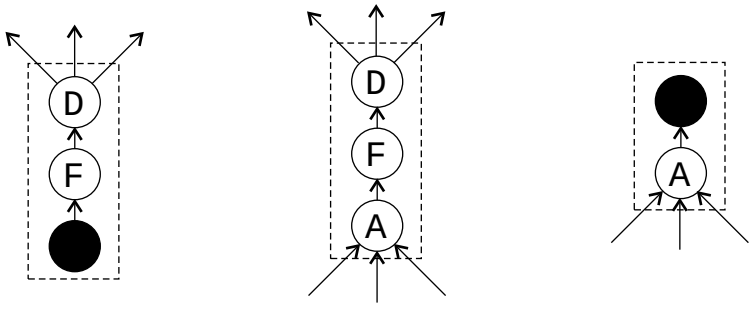
large-scale deployment and considerable capital investment. Thus it is clear that there is a pressing need to develop new methods that allow us to evaluate Web design questions that are system-wide in scope.

Our goal is to take a step towards a system-wide view of the Web and show that this new view holds promise for a higher level of performance engineering of the Web. Treating the Web as a collection of interacting components is a departure from the traditional focus of Web performance research, which has concentrated on individual components and their associated metrics, such as hit ratio, throughput and response time. Instead of focusing on Web *components* (clients, caches, and servers), we turn our attention to the characteristics of the *streams of requests* that flow through the Web system. In this view, each arc in Figure 1(b) corresponds to a stream or sequence of requests for Web objects sent from one component to another, and each component may be thought of as a stream transformer. The result is a "Web of Streams." Our strategy then becomes to uncover underlying principles that drive the transformations on these streams, in order to be able to better reason about the properties of the streams encountered at different points in the system.

Qualitative Reasoning About the Web

Interpreting data collected from many different points in the Web of Streams is challenging. A starting point is to realize that, despite the complexity of Figure 1(b), it is clear that individual components may be thought of as having a rough "location" in this system, according to whether they are near to clients, servers, or neither. A component may be said to be near clients if it receives requests from relatively few clients, and sends requests to relatively many servers; for components near servers, the situation is reversed. This notion of a component's location is useful in understanding how the properties of streams vary.

As the location varies from near clients to near servers, the properties of streams vary due to effects that can be thought of as stream *transformations.* In fact, there are three basic transformations that take place. First, streams may be aggregated, which happens when the requests of two or more different streams are interleaved by order of arrival to produce a single resulting stream. The second transformation is the reverse: streams may be disaggregated, meaning that requests are divided based on destination into multiple component streams that each proceed on a different path. Lastly, streams may be filtered, meaning that some of the requests initially present in the stream are removed, while the remainder of the requests is sent past



(a) Client (b) Proxy Cache (c) End Server

Figure 2: Abstractions of typical components of the Web topology in the "ADF" graph. The node types are Aggregator (A), Disaggregator (D), Filter (F), and End Nodes (dark), and correspond to points in the topology at which the Web request streams can be altered.

the filter.

The aggregation-disaggregation-filtering (ADF) framework is a high-level description of the Web of Streams. Thus, the ADF framework is an example of a qualitative model — a representation that provides notations for describing and reasoning about properties of the physical world [4]. These models provide convenient starting points when analytical solutions or comprehensive simulations are inherently difficult, as is the case with the global Internet [2]. Such models can be useful because for many engineering problems, such as the Web performance questions that concern us here, one often prefers a rapid, approximate answer over a precise, but costly analysis.

Furthermore, the qualitative ADF model can provide a framework for organizing a quantitative analysis. This is because the ADF model is sufficient to capture the entire system shown in Figure 1(b). All the changes to streams performed by components in the figure can be represented in terms of the 3 ADF transformations. This is shown in Figure 2. The figure shows how these transformations may be combined to represent the effects that a client, a proxy cache, or an end server has on streams of requests.

The entire Web topology can thus be seen as a graph, in which the nodes are of the types of the transformations, and the edges are the connections between them. We define three types of nodes, corresponding to these three transformations: Aggregators (A), Disaggregators (D) and Filters (F). We also define two additional endpoint nodes, which can generate and absorb requests; these are part of the user agents and origin servers, respectively. Edges can represent both the flow of requests between different components, such as a client and a server, and also between internal parts of a component.

Quantitative Reasoning: Temporal Locality in the Web

Quantitative analysis using the ADF framework focuses on properties associated with streams, and the manner in which those properties are affected by stream tranformations. From the standpoint of Web caches, the most important property of a stream is its *temporal locality*. Thus, we next apply the ADF framework to the specific case of studying temporal locality, seeking to reason about the Web system as a whole.

Intuitively, temporal locality is the notion that in some stream, references to a given object tend to appear close to each other in time — closer than would occur if all objects were equally likely to be referenced, and references were made independently. The notion of temporal locality was first identified and explored in the 1960's by researchers including Belady, Dennis, and Denning in the context of program memory references,

as summarized in [1].

Temporal locality is closely tied to the performance analysis of demand-driven caches, and is important when analyzing caches of any kind (*e.g.*, caches for virtual memory pages, filesystem buffers, processor data values, or Web pages). This is because increased temporal locality generally improves the performance of demand-driven caches. Such a cache is designed to store some subset of the objects previously accessed; a good cache has the property that the objects held in the cache have a higher likelihood of being accessed in the near future than would a random collection of objects.

Temporal locality can arise in two ways [8, 5]: First, it can exist because an object is simply more popular than its peers; Second, it can exist when references to an object occur in a correlated manner. To see this intuitively, we use an example from [6]. The first effect can be seen in the reference sequence

$$...XABXXCXDXXXEFXX...$$

in which temporal locality arises from the popularity of 'X': its inter-reference times are generally much shorter than would be expected were all objects equally popular. The second effect can be seen in

$$...GGHHIIJJKKLL...$$

in which temporal locality arises from the pattern of letter repetition. An important difference between the two effects is that, in a random permutation of the sequence, the first property is preserved whereas the second is not.

To develop metrics for temporal locality that can be analyzed in the ADF framework, we rely on this decomposition of temporal locality into two effects — popularity, and correlation. These metrics are defined in terms of a *trace* – a finite sample or record of requests taken from a given stream.

Measuring Popularity To measure the degree of imbalance in the popularity of objects we require a metric that captures the degree that a particular distribution deviates from the uniform case. Such a metric is the empirical *entropy* of the request stream [10]. For any trace referencing N distinct objects, empirical entropy varies from a minimum value of 0 when the popularity distribution is maximally skewed (all references are identical), to a maximum value of $log_2 N$ when the references are uniformly distributed over all objects.

Thus, the stream is treated as if it were independent samples of a random variable X, and we are concerned with the entropy of X, denoted $H(X)$:

$$H(X) = -\sum_{i=1}^{N} p_i \log_2 p_i \tag{1}$$

where p_i is the probability of a reference to object i, estimated by its empirical frequency in the trace. Note that $H(X)$ only depends on the probabilities of occurrence of the different objects, and not on the relative order in which they occur.

It has been shown that the number of distinct references in a segment of a trace increases with the size of the segment, even for very large traces [7]. Thus, to compare different traces, we need to normalize $H(X)$ for the number of distinct objects referenced; the appropriate normalization is the largest possible value of $H(X)$, which is $\log_2 N$. Hence we define *normalized entropy* H^n as:

$$H^n = \frac{H(X)}{\log_2 N} \tag{2}$$

where N is the number of distinct objects referenced in the trace.

In practice, entropy is a useful measure of imbalance in popularity. For example, in [3], we find that across a large set of traces and cache sizes, normalized entropy is strongly correlated with the performance

of an LRU cache (measured in terms of hit ratio). This relationship is given theoretical grounding in [9], which shows that source entropy can be used to bound the hit ratio of a cache, given a stream of references from a memoryless source.[2]

Measuring Correlation Absence of correlation in a request stream is captured by the Independent Reference Model (IRM) [1]. In the IRM, each object i has an associated probability of being referenced (p_i), and each reference is independent of any other reference. In this model the inter-access time (IAT) for a object i (where time is measured in number of references) follows a geometric distribution with parameter p_i.

If accesses to an object are in fact correlated, then the object's IAT distribution will deviate from the geometric; so to measure correlation we introduce a metric that is very sensitive to this deviation. The *coefficient of variation* (CV) of a distribution is its standard deviation divided by its mean. To form a metric for a given trace, we take an average of IAT-CV over all references in the trace, as described in [3].

For the geometric distribution with parameter p_i, the CV is $\sqrt{1-p_i}$. In Web reference streams, in which even the most heavily accessed objects have a very low probability of reference (generally much less than 1%), the expected IAT-CV in the case of no temporal correlation is very close to 1. Therefore, in Web reference streams, CV values close to unity suggest that reference patterns are close to the IRM, *i.e.,* contain little temporal correlation; while values larger than one represent a distribution with large relative variance, and so suggest the presence of strong inter-reference correlations.

Thus, IAT-CV is an effective metric for capturing temporal correlation. In a manner similar to normalized entropy, the importance of correlation in a reference stream can be measured by its effect on hit ratio in an LRU cache. When a trace is scrambled (removing correlations) the resulting hit ratio tends to decrease; in [3] we find that this decrease is strongly correlated with IAT-CV.

Exploring Stream Locality

So far, we have been devoted to decomposing the complexity of Web systems along two axes: the transformations operating on Web streams have been decomposed into aggregation, disaggregation, and filtering; and the locality property of request streams has been decomposed into popularity and correlation. Using this decomposition, we can begin to attack the underlying problem, namely the system-wide engineering of the Web. Organizing the various processes of the system along these two axes yields a set of tractable analyses that can ultimately be re-combined into a system-wide view. The idea is to consider how each transformation operates on each source of locality; we organize our findings using the tableau in Figure 3. The results in this table are derived from measurements of Web traces, experimental evaluation of transformations of Web traces, and analytic considerations, as described in [3].

Filtering. Filtering by a cache (under commonly used cache replacement policies like LRU and LFU) tends to remove both components of locality. It removes popularity (increases entropy) because highly popular objects are more likely to be found in the cache; and it tends to remove correlation (decreases IAT-CV) because recently referenced objects are more likely to be found in the cache.

Aggregation. Aggregation of streams coming from distinct upstream sources tends to increase locality, especially near servers. This occurs because entropy decreases, while IAT-CV changes very little. Entropy decreases because the popularity of documents in the merged stream is generally more skewed than in the component streams being merged. This surprising effect, observed in empirical traces, occurs because globally popular documents tend to occur in each component stream, while unpopular documents do not.

[2]Note that the $H(X)$ we use is a first-order approximation to the source entropy considered in [9].

	Filtering	Aggregation	Disaggregation
Popularity	Increases Entropy	Decreases Entropy, especially near Servers	Little Effect
Correlation	Decreases IAT-CV	Little Effect	Decreases IAT-CV, especially near Clients

Table 1: Analyzing Locality Changes in the ADF Framework

This effect is most pronounced near servers, where the largest number of independent streams are being merged.

Disaggregation. Disaggregation into separate streams tends to reduce locality, but its effect is only pronounced near clients. Disaggregation has little effect on entropy; this seems to be because resulting downstream component tends to maintain skewed object popularity approximately like the original stream. This is consistent with the many studies that have shown that even among objects on a single Web server, Zipf's law is quite pronounced. Disaggregation does tend to reduce IAT-CV, but this effect is only pronounced when disaggregation occurs close to the client. At most other places in the Web system there is little correlation in either incoming or outgoing streams.

Location in the Web of Streams. Applying these insights to the Web system, we can start to understand how locality changes as a function of location in the system. Referring back to Figure 1(b), we say that a component is near clients if it receives requests from relatively few clients, and sends requests to relatively many servers; and if the situation is reversed, the component is near servers. We can then ask how locality varies as we move along the continuum of location from "near clients" to "near servers" – that is, from left to right in Figure 1(b).

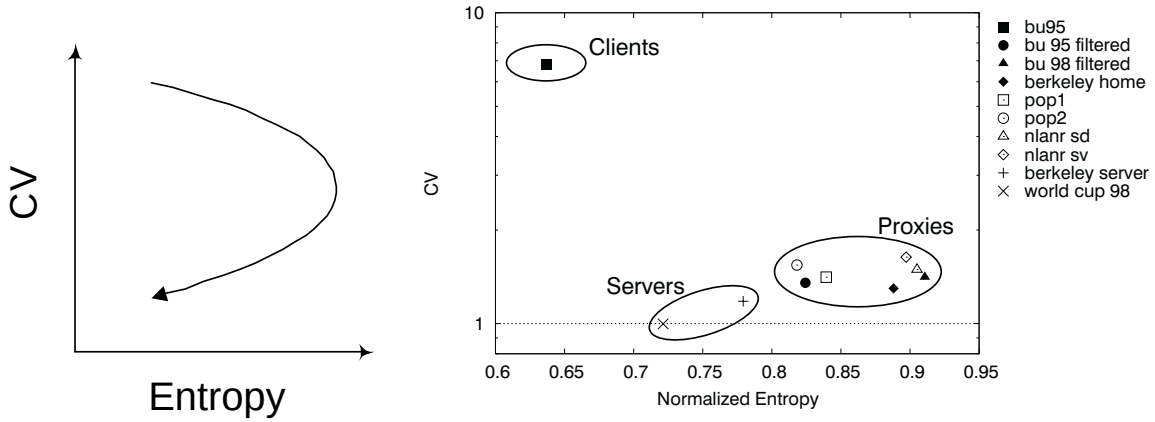
Figure 4(a) shows the answer. Near clients, locality is particularly strong (high CV and low entropy; upper left). As we move away from clients, the effects of filtering (due to intermediate caches) and disaggregation lower locality — entropy increases and CV decreases. However as we move near to servers, locality increases again, because of the increasing ability of aggregation to lower entropy.

The pattern shown in Figure 4(a) is in fact observed in empirical measurements. Figure 4(b) shows the entropy and CV of ten traces, collected from diverse locations in the Web. In the upper left of the figure is a trace captured before the browser caches, directly on client machines; in the middle right are traces from proxies in intermediate locations in the Web; and in the lower left are traces captured at servers.

Conclusions

The system-wide approach to analyzing the Web presented here helps shed light on a wide range of engineering questions.

For example, it has been commonly noted that intermediate caches in a multi-level hierarchy are generally relatively ineffective [11]; this results in a "diminishing returns" effect as cache space is added to intermediate levels of the hierarchy. The further "up" (*i.e.*, away from clients) in the hierarchy a cache is



(a) (b)

Figure 3: The two components of temporal locality in varying locations. (a)The curved arrow shows how locality changes with location, moving from near clients to near servers.(b)CV versus Normalized Entropy for real traces studied.

placed, the lower hit ratio it tends to achieve. Furthermore, simply adding additional storage space to an intermediate level cache does not generally improve its hit ratio significantly. These effects can be understood in light of Figure 4 as a consequence of the high entropy and low IAT-CV expected when far from either clients or servers. We conclude that the design of cost-effective multi-level caching hierarchies should incorporate consideration of the location of the component caches and the properties of their arriving reference streams, rather than on the performance of each proxy cache in isolation, as is the current practice.

The choice of cache replacement policies is also an important question. Policies like LFU exploit low entropy, which policies like LRU are more effective when IAT-CV is large. From Figure 4, one can conclude that LFU-like policies gain effectiveness over LRU-like policies as one moves further from clients and closer to servers.

A final system engineering question concerns cache placement. The ADF model indicates that aggregation of request streams decreases the entropy of the stream that is offered to the next-level cache, providing an opportunity for higher hit ratios. In light of this type of analysis, we may expect that caches are more likely to be effective at locations where many streams of requests merge, as at network access points; such placement would exploit the reduction in entropy that occurs when streams are merged. In general, we anticipate that network engineers may choose to analyze the properties of request streams at different points in the network so as to optimize cache placement.

These examples show that although the Web's complexity has grown enormously since its inception, it is still possible to discern system-wide structure and regularity in its characteristics. The Web of Streams provides a useful tool in attacking this challenge. Armed with the understanding that stream-based analysis provides, we can make distinct progress in understanding and engineering the Web system as a whole.

References

[1] Peter J. Denning. Working sets past and present. *IEEE Transactions on Sofware Engineering*, SE-6(1):64–84, January 1980.

[2] S. Floyd and V. Paxson. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, August 2001.

[3] Rodrigo Fonseca, Virgílio Almeida, Mark Crovella, and Bruno Abrahao. On the intrinsic locality properties of web reference streams. In *Proceedings of the IEEE INFOCOM 2003*, April 2003.

[4] K. Forbus. Qualitative physics: Past, present, and future. In *Exploring Artificial Intelligence*. Morgan-Kaufmann, 1988.

[5] Shudong Jin and Azer Bestavros. Sources and Characteristics of Web Temporal Locality. In *Proceedings of the 8th MASCOTS*. IEEE Computer Society Press, August 2000.

[6] Shudong Jin and Azer Bestavros. Temporal locality in web request streams. Technical Report BUCS-TR-1999-014, Boston University Computer Science Department, July 2002.

[7] Terrence Kelly and Jeffrey Mogul. Aliasing on the world wide web: Prevalence and performance implications. In *Performance Track, 11th Intl. World Wide Web Conference*, 2002.

[8] A. Mahanti, D. Eager, and C. Williamson. Temporal locality and its impact on web proxy cache performance. *Performance Evaluation Journal: Special Issue on Internet Performance Modelling*, 42(2/3):187–203, September 2000.

[9] Gopal Pandurangan and Eli Upfal. Can entropy characterize performance of online algorithms? In *Symposium on Discrete Algorithms*, pages 727–734, 2001.

[10] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley and Sons, 1991.

[11] Carey Williamson. On filter effects in web caching hierarchies. *ACM Transactions on Internet Technology*, 2(1):47–77, February 2002.