The Trickle-Down Effect: Web Caching and Server Request Distribution

Ronald P. Doyle*
Application Integration and Middleware
IBM Research Triangle Park
rdoyle@us.ibm.com

Jeffrey S. Chase
Syam Gadde
Amin M. Vahdat †
Department of Computer Science
Duke University
chase, gadde, vahdat@cs.duke.edu

ABSTRACT

Web proxies and Content Delivery Networks (CDNs) are widely used to accelerate Web content delivery and to conserve Internet bandwidth. These caching agents are highly effective for static content, which is an important component of all Web-based services.

This paper explores the effect of ubiquitous Web caching on the request patterns seen by other components of an end-to-end content delivery architecture, including Web server clusters and interior caches. In particular, object popularity distributions in the Web tend to be Zipf-like, but caches disproportionately absorb requests for the most popular objects, changing the reference properties of the filtered request stream in fundamental ways. We call this the *trickle-down effect*.

This paper uses trace-driven simulation and synthetic traffic patterns to illustrate the trickle-down effect and to investigate its impact on other components of a content delivery architecture, focusing on the implications for request distribution strategies in server clusters.

Keywords

Zipf, Web Proxy Cache, URL Hashing, Load Balancing, Request Distribution

1. INTRODUCTION

As traffic on the Internet continues to grow, proxy caches and Content Delivery Networks (CDNs) are increasingly used to make Web service more efficient. These caching agents serve requests from content replication points closer to the clients, reducing origin server load, consumed network bandwidth, and client response latency.

Several factors contribute to increasing use of Web caching. A few years ago, Web caches were used only by clients who explicitly configured their Web browsers to use proxies. Today, access ISPs commonly reduce their bandwidth costs by automatically redirecting outgoing Web request traffic through interception proxies. At the same time, hosting providers often deploy surrogate caches to filter incoming request traffic for data center customers. Many Web sites also contract with third-party CDNs to serve their request traffic from caching sites at various points in the network.

Ubiquitous caching has the potential to reshape the Web traffic patterns observed by the other components of a content delivery system, including Web server sites and interior caches that receive requests "downstream" from forward caches. Upstream caches filter the request stream and shift the downstream loads from popular static objects to less popular objects and dynamic (uncacheable) content. We refer to this as the *trickle-down effect*. It is important to understand this effect because assumptions about traffic patterns drive many of the local design choices in Web delivery architectures.

This paper explores the effects of Web caches on the properties of the request traffic behind them. We use trace-driven simulation and synthetic traffic patterns to illustrate the trickle-down effect and to investigate its implications for downstream components of an end-to-end content delivery architecture. We focus on Web requests that follow a Zipf-like object popularity distribution [18], which many studies have shown to closely model Web request traffic [11, 8, 5, 17]. We then illustrate the importance of this effect by examining its impact on load distribution strategies and content cache effectiveness for downstream servers. The contribution of this paper is to demonstrate and quantify the key implications of the trickle-down effect on downstream servers at a given server request load:

- Forward caching disproportionately absorbs requests for the most popular objects. The miss stream consists of evenly distributed references to moderately popular objects and recently updated or expired objects, together with the heavy tail of the popularity distribution.
- By absorbing much of the locality in the request stream, upstream caches reduce the effectiveness of downstream content caches (e.g., server memory), increasing pressure on server storage systems. This phenomenon is well-known in the file system context [13].
- The reduced locality in the miss stream tends to increase the importance of content-aware request distribution (such as *URL hashing*) for effective caching in downstream server clusters.

*R. Doyle is also a PhD student in the Computer Science department at Duke University.

†This work is supported by the U.S. National Science Foundation through grants CCR-00-82912 and EIA-9972879. Gadde is supported by a U.S. Department of Education GAANN fellowship. Vahdat is supported by NSF CAREER award CCR-9984328.

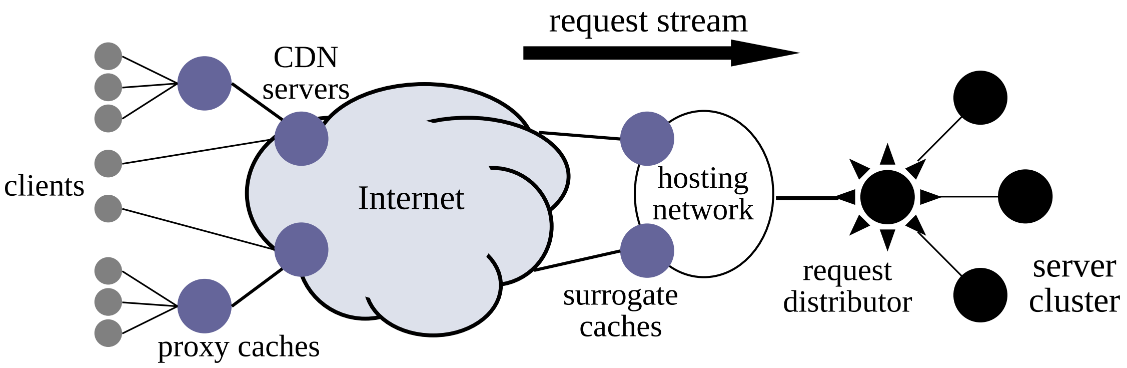

Figure 1: End-to-end architecture for Web content delivery.

- At the same time, forward caching suppresses the primary source of load imbalances in Web server clusters, reducing the need for dynamic load balancing when content-aware request distribution is used.

Our results emphasize the need to consider how deployment of caches and CDNs affects other functional components of the Internet infrastructure. More generally, in evaluating solutions and policies for managing Web content it is important to consider their role as components of a complete end-to-end content delivery architecture.

This paper is organized as follows. Section 2 discusses the relevance of our work to end-to-end content delivery, and outlines our experimental methodology. Section 3 presents analytical and experimental results illustrating the trickle-down effect, and examines its implications for server request distribution, load balancing, and cache management. Section 4 considers the question of how the trickle-down effect manifests in today's Web. Section 5 summarizes our conclusions.

2. BACKGROUND

Figure 1 depicts the key components involved in delivering content from a large Web site to a community of clients. The request stream originates at clients on the left-hand side, and filters through one or more caches; requests not absorbed by the caches pass through to the origin servers on the right-hand side. Caches filtering the request stream may include client browser caches, demand-side proxy caches acting on behalf of a community of clients or an access ISP, and supply-side surrogate caches (also called reverse proxies) acting on behalf of the Web site's hosting provider or a third-party CDN service. A variety of mechanisms route the request stream through caches: clients may be configured to use a cache, or the system may redirect requests by interposing on DNS domain name translation or by intercepting the request at the IP level. In addition, each cache may control the routing of its miss stream to the other components.

The last two years have seen an explosion of growth in Web caching and content delivery infrastructure. Key developments include CDN service offerings from Akamai, Digital Island, and others, increasing use of surrogate caching among hosting providers such as Exodus, and aggregation of content consumers into large ISPs employing transparent interception proxies based on L7 switch hardware from Cisco/Arrowpoint, Nortel/Alteon, Foundry, and others. Each of these development has fed the growth in demand for Web caching systems from vendors including Inktomi, Network Appliance, and CacheFlow.

One goal of this paper is to explore the effect of upstream caching on the effectiveness of downstream request distribution policies, e.g., at the origin site. The right-hand side of Figure 1 depicts a typical origin site for a large-scale Web service. Many of today's Web sites are hosted on server farms, where a group of servers are clustered together to act as a unified server to external clients. Any given request could be handled by any of several servers, improving scalability and fault-tolerance. The switching infrastructure connecting the servers to the hosting network includes one or more redirecting server switches to route incoming request traffic to the servers. We refer to these switches as *request distributors* because they select the server to handle each incoming request. The server selection policy plays a key role in managing cluster resources to maximize throughput and meet quality-of-service goals.

Commercial server switches use a variety of server selection policies to distribute requests to maximize throughput and minimize response latency. *Server load balancing* policies (SLB) monitor server status and direct requests to lightly loaded servers. SLB switches are often called L4 switches because they make server selection decisions at connection setup time, and examine only the transport (layer 4) headers of the incoming packet stream. *Content-aware* server selection policies prefer servers that can handle a given request most efficiently, for example, a server likely to have the requested data in cache. *URL hashing* (URLHASH) is a content-based policy that applies a simple deterministic hash function on the request URL to select a server. URL hashing is sometimes called an L7 policy because the switch must parse HTTP (layer 7) headers to extract the URL.

2.1 Zipf-Like Request Traffic

Observations of Web request patterns drive the design choices and policies for all of these components of a Web delivery architecture. In particular, a number of studies indicate that requests to static Web objects follow a Zipf-like popularity distribution [11, 8, 5, 17]. The probability p_i of a request to the ith most popular document is proportional

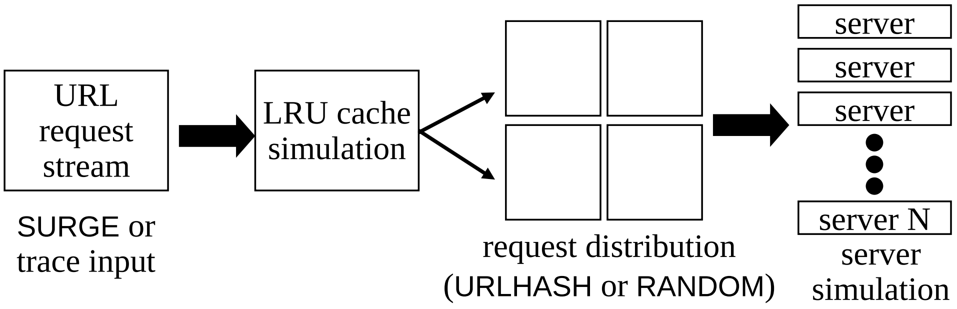

Figure 2: Simulation pipeline for study of caching and request distribution.

to $1/i^{\alpha}$, for some parameter α. A large number of requests target the most popular sites and the most popular objects, but the distribution has a long, heavy tail of less popular objects with poor reference locality. Higher α values increase the concentration of requests on the most 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 [17]. Unfortunately, caching is less effective on the heavy tail, which comprises a significant fraction of requests. This is why Web cache effectiveness typically improves only logarithmically with size, measured either by capacity or by user population.

Zipf-like behavior also has implications for request distribution strategies in server clusters. For example, it creates a tension between the competing goals of load balancing and locality. On the one hand, content-aware policies such as URLHASH effectively take advantage of the locality present in the request stream by preferring the same server for repeat requests, maximizing server memory hits for popular objects. However, URLHASH is vulnerable to load imbalances because the most popular objects receive the largest number of requests, and a single server handles all requests for any given object. SLB policies balance load, but they tend to scatter requests for each object across the servers, reducing server memory hits for moderately popular objects.

Recent research [15, 2, 3] has studied this tradeoff in depth, and developed *Locality Aware Request Distribution* (LARD) and related policies to balance these competing goals, combining the benefits of each approach. The LARD approach is commercialized by Zeus technology [1]. Other commercial request distributors use less sophisticated strategies. Server switches from Foundry, Nortel/Alteon and Cisco/Arrowpoint can assign multiple servers to each URLHASH bucket, and select from the target set using load information. The IBM Websphere Edge Server includes intelligent load balancing software for routing requests among servers based on content, server load, connection statistics, and customizable monitors. This product also contains a Web proxy cache.

2.2 Goals and Methodology

This paper examines the impact of caching on downstream request traffic patterns, using the simulation pipeline depicted in Figure 2. We limit our experiments to static, cacheable content, and we do not consider the effect of object rate of change or expiration. We use a simple Web cache simulator [9] to filter representative request traces through caches of various sizes. Our model assumes a simple LRU replacement policy with no cache cooperation among the front-end cache and the servers. Our results are conservative in that we use small cache capacities relative to the aggregate content size. In these experiments we assume that a single cache filters requests from all clients; this assumption is reasonable for CDNs and proxies serving 20,000 or more users [17], but it exaggerates the trickle-down effect from proxies serving small populations.

We use simulations to explore the properties of the cache miss stream and their effects on downstream components, focusing on the effectiveness of server content caching and server request distribution policies. We study two server selection policies: URLHASH and RANDOM. These policies are sufficient to quantify the impact of the trickle-down effect on the performance potential of a range of policies for server request distribution. We substitute RANDOM for SLB because it is easier to simulate, and its locality properties are no worse than SLB.

Table 1 shows some characteristics of the server traces and synthetic request streams used in the study. The server traces are from IBM Corporation's main Web server (www.ibm.com). Because these traces are collected at the entry point(s) of a large, popular Web server, they capture the characteristics of request streams likely to be seen at the kind of server clusters we target in this study. The first trace contains 15.5M requests over a period of 3.5 days in June 1998, showing a best-fit Zipf α value of 0.764. We select this trace partly to facilitate direct comparison with the LARD study, which uses the same trace. The second IBM trace was collected on February 5-11, 2001. As shown in Table 1 the IBM traces deviate somewhat from expected Zipf-like behavior, e.g., they contain a small number of popular objects that account for an unusually large percentage of references.

We also use the SURGE Web server load generator [4] to synthesize Zipf-like request traces using α values between 0.6 and 0.9, to allow experimentation with a range of values observed in previous traffic studies. The synthetic traces were generated with default SURGE settings for temporal locality and object size distributions.

Trace	Objects	Footprint	References	References from top 1% of objects	References from top 5% of objects
1998 IBM	38527	1 GB	15.5M	77%	94%
2001 IBM	52763	1.6 GB	43M	93%	97%
Zipf Alpha 0.9	40000	900 MB	4.2M	42%	57%
Zipf Alpha 0.8	20000	440 MB	7.5M	27%	42%
Zipf Alpha 0.7	20000	440 MB	16M	17%	30%

Table 1: Trace Characteristics

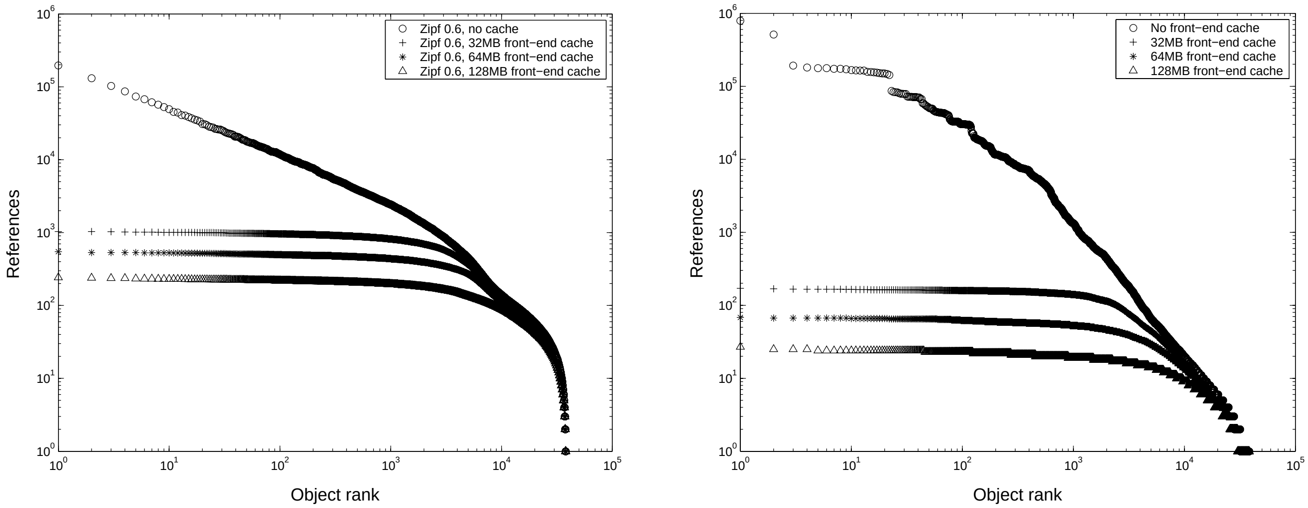

Figure 3: Number of references to each object by rank (log-log), before and after caching. The left-hand graph is from a synthetic trace with $\alpha = 0.6$; the right-hand graph is from the 1998 IBM trace.

3. EXPERIMENTAL RESULTS

This section presents experimental results in four subsections. Section 3.1 illustrates the traffic properties of the "trickle-down" cache miss stream. Sections 3.2 and 3.3 examine the impact of these properties on load balancing and cache locality respectively.

3.1 Properties of the Miss Stream

Figure 3 illustrates object reference frequency by rank before and after caching, on a log-log scale. As expected for Zipf-like distributions, the original traces roughly follow a straight line; the straight line breaks down for less popular objects because the traces are of bounded size. The figure shows that even a small forward cache "skims the cream" off the request stream, absorbing requests for the most popular objects. The effect is to "flatten" the distribution so that the most popular objects in the miss stream receive roughly the same number of requests. In fact, there are 1035 references to the most popular object in the $\alpha = 0.6$ miss stream from a 32MB cache, while the object with rank 1000 is referenced 816 times. The effect is even more pronounced in the right-hand graph from the 1998 IBM trace, which has a heavier concentration on the most popular objects in the original trace. Larger forward caches extend the flattening effect, but are qualitatively similar.

To illustrate the disproportionate effect of caches on popular objects, Figure 4 shows per-object hit rates for popular objects in the same input traces, as a function of popularity rank. This graph shows that cache effectiveness falls off rapidly with object popularity. The effect is more pronounced in the IBM trace; its most popular objects miss in the cache even less often than for the $\alpha = 0.6$ trace, explaining the smaller number of references to these objects in the miss stream in Figure 3.

The intuition behind the trickle-down effect is as follows. Suppose that requests to each object are evenly distributed in the input trace, and that it takes R requests of the trace to fill a cache of k distinct objects. The expected value of R is a function of the input distributions and the cache size, but we leave aside the question of how to compute it. (R is on the order of 10^5 for the 1998 IBM trace with a 32MB cache.) The probability that none of the R requests references the object with popularity rank i is $(1 - p_i)^R$; this gives the probability that a reference to object i misses in the cache after any R references. Thus the probability of a reference to object i in the miss stream is $q_i = p_i(1 - p_i)^R$; this gives the probability that any given element of an infinite request stream is a reference to object i and that the reference misses in the cache.

Figure 5 shows a scaled plot of $q_i = p_i(1 - p_i)^R$ for a Zipf $\alpha = 0.7$ and $R = 10^4$. The graph shows that it is the moderately popular objects that are most likely to appear in the miss stream; these become the most popular objects in the filtered distributions of Figure 3. This is because the probability of a forward cache miss $(1 - p_i)^R$ is vanishingly small for popular objects, thus few requests to

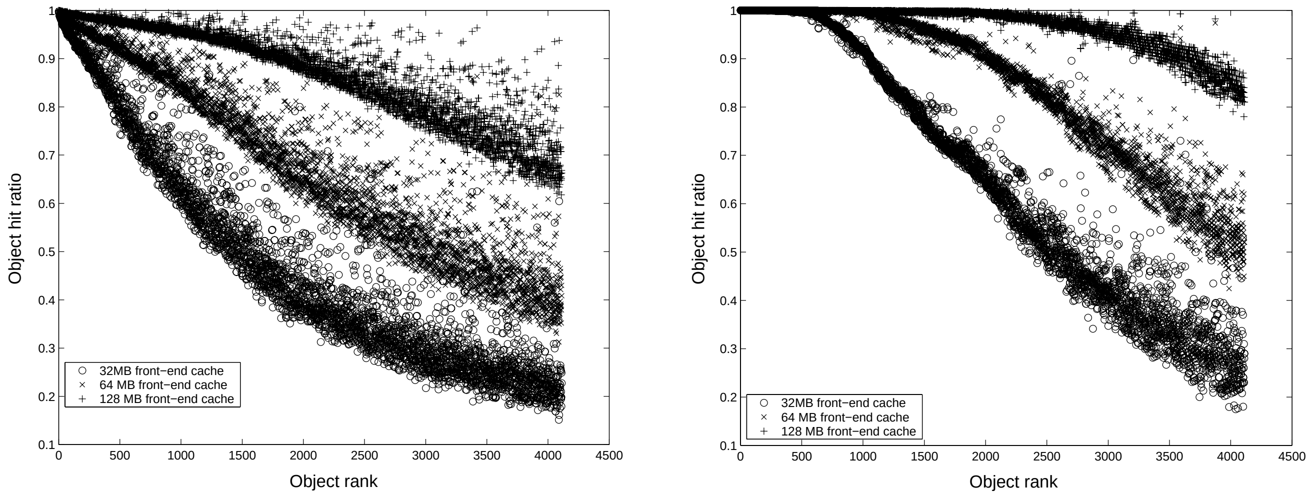

Figure 4: Object hit rate in front-end caches, by object rank in the input trace. The left-hand graph is from a synthetic trace with $\alpha = 0.6$; the right-hand graph is from the 1998 IBM trace.

Operation	*CPU time in μs*
Connection setup	145
Net transmit (512 bytes)	40
Disk transfer (4K bytes)	410

Table 2: CPU cost parameters for simulation.

popular objects pass through the cache. While this probability approaches unity for unpopular objects, requests for unpopular objects are rare to begin with. For comparison, Figure 5 also shows the number of references to each object after passing the 1998 IBM trace through a 32MB cache, as a function of object rank in the original input trace. This curve confirms our intuition.

3.2 Load Balancing

These effects of forward caching on the reference distributions in the miss stream have important implications for downstream components. To illustrate, we now consider the impact on server request distribution policies. One implication is that the trickle-down effect suppresses the load skew caused by popular objects in Zipf-like request streams. This means that simple content-aware request routing policies such as URLHASH become less vulnerable to load imbalances than they have been in the past.

We used a simple server simulator to quantify this effect. The simulator assigns a fixed CPU cost for each connection (we assumed a single request per connection) and for each block transferred. A server cache miss also imposes a per-block CPU transfer cost from disk. Table 2 gives the simulation parameters for these costs, which are identical to those used in the LARD study [15].

The simulation results compare load distributions from URLHASH and RANDOM request distribution of filtered traces across varying numbers of servers. The results exaggerate load imbalances relative to the LARD study in the following respect. We assume that all client requests are available at the start of the experiment, to eliminate server wait time between requests; this "squeezes out" all idle time of underutilized servers to the end of the run. We then quantify load skew as the average percentage CPU time difference between the slowest server and all other servers:

$$\sum_i^N (T_{slow} - Time_i)/(N * T_{slow})$$

where T_{slow} is the CPU time for the slowest of the N servers. This conservative load skew metric gives the average idle time percentage of servers left underutilized due to load imbalances. A lower load skew figure indicates a more balanced load across the servers. This simulation assumes adequate I/O system bandwidth, and considers only the CPU cost to manage the I/O.

Figure 6 shows the load skew results for RANDOM and URLHASH on the cache miss streams, as a function of the server

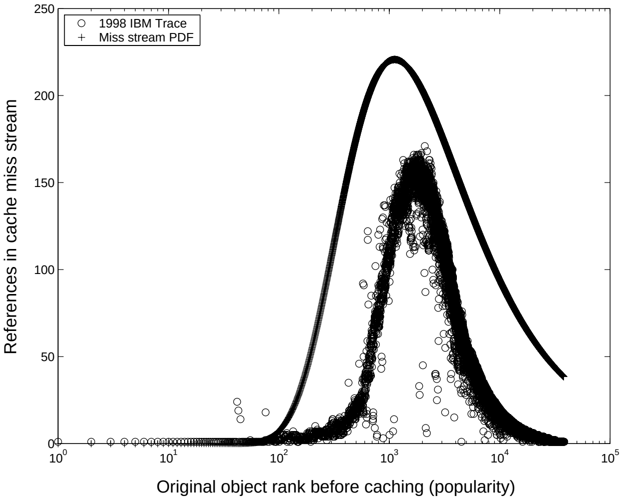

Figure 5: Number of references to each object in the 32MB cache miss stream, ordered by rank in the 1998 IBM trace. The superimposed curve is a scaled plot of $q_i = p_i(1 - p_i)^R$ for a Zipf $\alpha = 0.7$ and $R = 10^4$.

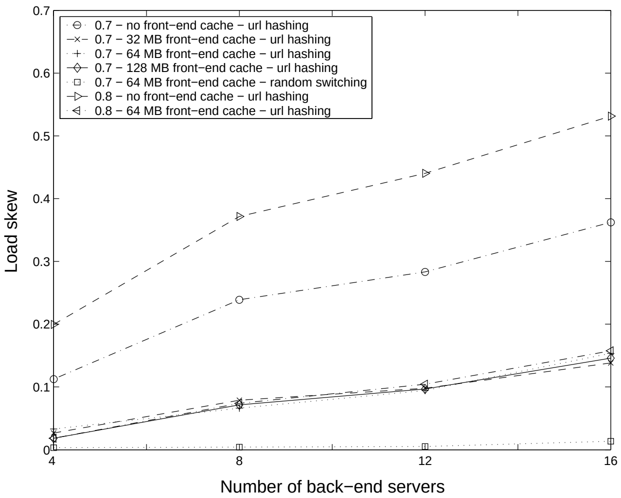

Figure 6: Server load skew (synthetic traces with $\alpha = 0.7$ and $\alpha = 0.8$).

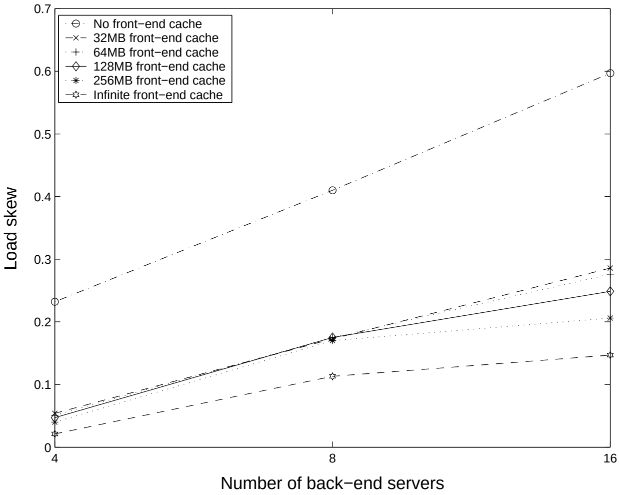

Figure 7: Server load skew (2001 IBM trace)

cluster size N. These experiments use a server memory size of 32MB, as in the LARD study. RANDOM balances the load as expected. 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. The imbalance is worse for the $\alpha = 0.8$, which concentrates references to popular objects even more. However, these load imbalances are less severe after filtering the request streams through forward caches; for example, a small cache drops the load skew to roughly 7% with 8 servers, independent of the α or the front-end cache size.

This effect is also visible with the IBM server traces, although a small number of very large objects introduce some skew (see below). For example, Figure 7 shows the load skew results for URLHASH on the 2001 IBM trace with the nine largest objects removed (all over 8MB). While forward caching is less effective at eliminating load imbalances than for the synthetic traces, the 8 server load imbalance is never more than 20% with a forward cache of 64 MB or larger. To further validate our simulation, we ran the 2001 IBM trace against a cluster of 8 machines running FreeBSD and Apache Web Servers, gathering load information from the cluster while using URLHASH to distribute the requests among the cluster machines. The results of this experiment are qualitatively similar to Figure 7, but the load skews are significantly lower for larger clusters and larger front-end cache sizes. One difference is that the test platforms have larger server memories (512 MB) than the simulated systems.

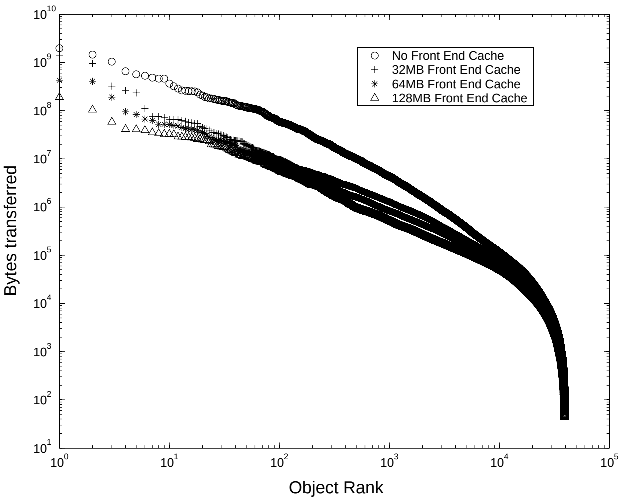

Figure 8: Request distribution by size*references (1998 IBM trace)

We ran additional experiments to investigate the impact of large objects on our results. We determined that much of the remaining load imbalance occurs because the caches in our simulations are ineffective at removing load skew due to variations in object size. In particular, large objects do not live in a small cache long enough to yield hits. For example, the objects in the IBM traces range in size up to 61 MB. Some of the largest objects are moderately popular, but their size makes them effectively uncacheable in our experiments.

To illustrate, Figure 8 shows the distribution of *bytes transferred* ($size * requests$) for objects in the 1998 IBM trace after filtering through caches of 32MB, 64MB, and 128MB. This metric approximates the amount of server work to serve requests for each object. The graph shows that although the caches suppress popularity skew, they do not eliminate the skew in bytes transferred from the server for each object. In practice, effectiveness of forward caching is typically limited by population rather than size, and size-related imbalances may be resolved at the origin sites by segregating continuous media files and other large objects.

These results indicate that explicit load balancing for content-aware request distribution in Web server clusters is less important in the presence of upstream caching. That is, a simple URLHASH strategy at the back end is unlikely to develop significant load imbalances when forward caching

Trace	256MB Aggregate Server Cache					
	0 Front-End Cache		64MB Front-End Cache		128MB Front-End Cache	
	URL	Random	URL	Random	URL	Random
1998 IBM	99%	92%	65%	3.5%	34%	1.6%
2001 IBM	98%	94%	46%	3.6%	28	1.6%
Zipf α 0.9	81%	43.7%	45%	3.0%	28%	1.6%
	512MB Aggregate Server Cache					
1998 IBM	99.4%	94.4%	85%	7.5%	74%	3.3%
2001 IBM	99%	95.3%	70%	7.8%	55%	4.2%
Zipf α 0.9	93%	50%	76%	6.5%	65%	4.1%

Table 3: Server cache hit rates for a 16-server cluster with various traces and front-end cache sizes, 256MB and 512MB aggregate server cache.

is effective.

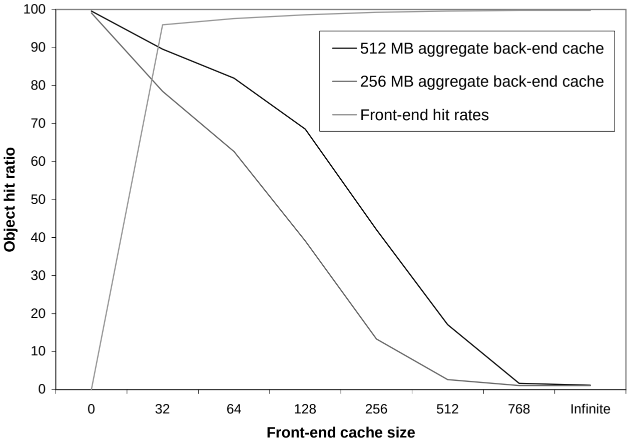

Figure 9: Cache hit rates for front-end and back-end caches of varying sizes in the 1998 IBM Trace.

3.3 Effects on Server Locality

A key advantage of content-aware request distribution is that it maximizes server cache effectiveness. However, the heavy-tailed nature of Web popularity distributions implies a low marginal benefit from multiple levels of caching [10]. Thus, the trickle-down effect reduces the ability of a server cluster to serve requests from memory for a given server request rate, independent of the request distribution policy. Thus servers tend to be increasingly I/O-bound in the presence of upstream caching, although there may be significant locality remaining in requests from caches to reload recently modified or expired objects.

Figure 9 illustrates this effect for ideal 256 MB and 512 MB aggregate server caches, as a function of front-end cache size. For the 1998 IBM trace, a 256 MB back-end cache is sufficient to serve more than 99% of references in the original trace from server memory. Even a 32 MB front-end cache absorbs 95% of the references in this trace, as predicted by the data in Table 1. With a 32 MB front-end cache the server caches still yield hit ratios above 80% for the miss stream. As the front-end cache grows, the back-end caches are left with just the tail of the original distribution, so back-end hit rates drop. When front-end and back-end caches can each store 25% of the content, the back-end hit rate is only 15%.

This shows that the potential of content-aware request distribution to reduce server I/O load is diminished in the presence of upstream caching. Simulations with URLHASH show that back-end cache hit ratios closely follow Figure 9, independent of the number of servers and their memory sizes. That is, URLHASH delivers an equivalent back-end hit ratio to an ideal unified cache of the same aggregate size for the traces we tested. For example, with a 128 MB front-end cache and 8 servers with 32 MB of memory each, URLHASH delivers the ideal back-end hit ratio of 41%.

Even so, upstream caching *increases* the marginal benefit from content-aware request distribution in many cases. Figure 10 shows the effectiveness of RANDOM request distribution to quantify the server cache hits that are sacrificed by a locality-blind policy such as SLB. The graph gives hit ratios in a 256 MB aggregate back-end cache for the 1998 IBM trace filtered through front-end caches of varying sizes. The lines show cache hit ratios as a function of the number of servers in the cluster; larger clusters exacerbate redundant caching as described in Section 2, reducing overall cache effectiveness. With no front-end cache, RANDOM yields back-end hit ratios above 90% for this trace even with 16 servers. With a front-end cache absorbing requests for the most popular objects, RANDOM never delivers a back-end hit ratio above 38%, while Figure 9 shows that a content-aware policy such as URLHASH could deliver hit ratios as high as 80% with a 32 MB front-end.

As the size of the front-end cache grows and the locality present in the miss stream drops, RANDOM becomes progressively less sensitive to the number of servers, and the performance gap beween RANDOM and the ideal back-end narrows. Yet URLHASH significantly outperforms RANDOM on the filtered request streams across a range of configurations. Table 3 gives results from the 2001 IBM trace and a synthetic Zipf-like trace with similar locality but more typical request concentrations for the most popular objects. In each case, even RANDOM delivers reasonable or competitive hit ratios for the original trace, but is not effective on the filtered traces. In contrast, URLHASH is effective at extracting the remaining locality from the filtered traces, yielding close-to-ideal hit ratios.

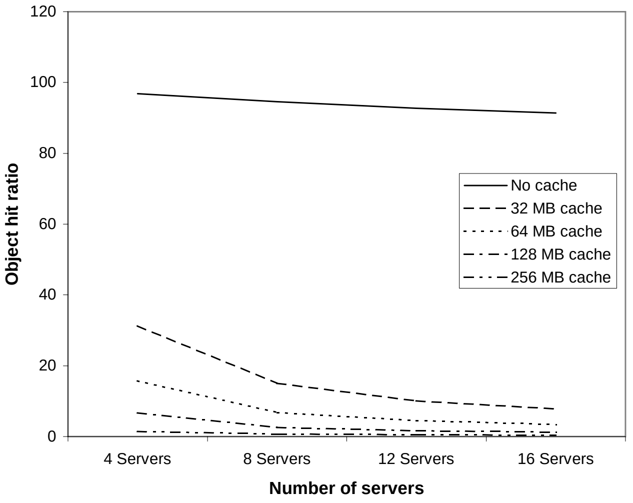

Figure 10: Server cache hit rates (256 MB aggregate) with RANDOM request distribution as a function of cluster size, for various front-end cache sizes (1998 IBM trace).

4. IS THE TRICKLE-DOWN EFFECT VISIBLE TODAY?

The simulation data presented in earlier sections clearly show the potential impact of widespread caching on popularity distributions. Can we see preliminary evidence of such a trend in current Web traces, and if not, why not?

A first step is to isolate the effects of other non-caching related trends, such as observed changes in Zipf distributions. Padmanabhan and Qiu [14] show that Zipf α values for the MSNBC site during 1998 and 1999 were higher than in previous proxy and server traces. They note that these traces were taken at a time when CDNs and surrogate caches were not employed at this site, and observe that α values for Web traffic are increasing, i.e., a relatively small number of documents are becoming more popular. This increases the potential benefit from caching and replication. The study also notes that caching and CDNs will reduce the load on the servers. Our results explore how this caching might affect the properties of the request "trickle" reaching the MSNBC server after CDNs or surrogates are deployed.

We looked for evidence of upstream proxy caches in the client request stream of the 2001 IBM trace. The IBM site uses the Network Dispatcher component of the Websphere Edge Server to balance load across servers, but, like the MSNBC site, IBM does not yet use surrogate caches or a CDN for this site. A few dozen clients generate more than 10,000 requests per day. While these are likely to be caches, there are no very large caches: no client generates more than 1% of requests. The top three clients account for 2% of the requests. This confirms that the potential of proxy caching in the Web is not yet fully realized, and that existing demand-side caches do not yet serve large enough user communities to have a significant impact on the request stream for sites that do not use CDNs. One factor may be that the IBM site is accessed primarily by businesses rather than consumers.

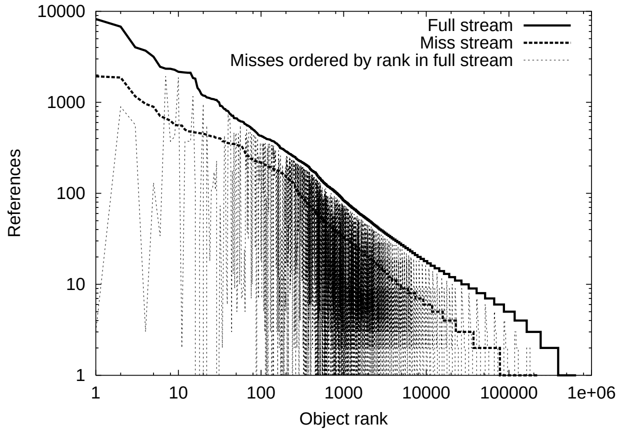

Figure 11: Popularity distributions for the full trace and the miss stream from NLANR caches on May 30, 2001, plotted logarithmically on both axes. We ignore objects whose requests hit in the NLANR caches less than 10% of the time.

We also examine one day of May 2001 traces from the top-level NLANR caches [16], which cooperate to act as a unified root proxy cache. Intuition suggests that the popularity distributions for the incoming NLANR cache request stream would be flat for the most popular objects, because the request stream itself originates from caches in lower levels of the hierarchy. However, the effect is not visible in Figure 11, which shows reference frequency distributions for "cacheable" URLs in the request stream (upper line) and the root cache miss stream (lower line). We consider a URL "cacheable" if it yields at least a 10% object hit ratio in the NLANR root cache. Even with this restriction, neither request stream shows the flattening characteristic of the trickle-down effect to the degree expected.

Further analysis reveals that the NLANR root cache effectiveness is highly variable, even for popular objects that appear to be cacheable. The fuzzy cloud below the upper line in Figure 11 is a plot of object request frequency in the miss stream, ordered by object rank in the original trace. This shows an unexpectedly low correlation between object popularity and cache effectiveness, even for "cacheable" objects (compare to Figure 4).

The NLANR traces do not contain enough HTTP header information to determine why any given request was not served from the cache. One possibility is that expiration headers in responses caused the caches to discard many objects that would have yielded hits. Another possibility is that cookies or other headers rendered the responses uncacheable for other reasons. The cache software in this system used a complex procedure for identifying cacheable responses.

In summary, there is little empirical evidence to indicate that Web caches significantly impact the request traffic of recent server studies, even in popular Web sites such as IBM and MSNBC. However, this paper shows that Web caches will reshape the request stream, to the degree that they are effective at all. A range of factors compromise the effectiveness of Web caching today, including incomplete deployment of CDNs, small population sizes on demand-side caches, and confusion about what content is cacheable during the transition to HTTP 1.1.

5. CONCLUSIONS

This paper shows that key assumptions about Web server request traffic may lead us astray as proxy caches and Content Delivery Networks (CDNs) become increasingly ubiquitous and effective. To be sure, perfect caching would absorb *all* static object requests, leaving origin sites to handle dynamic content and propagate object updates to the caching networks. But even imperfect caches can absorb the majority of requests to the most popular objects, shifting downstream traffic loads toward the heavy tail of Zipf-like Web request distributions. This "trickle-down effect" impacts the design of downstream components.

Our results illustrate the importance of considering the impact of caching on other components of an end-to-end content delivery architecture. This paper explores key implications of the trickle-down effect for request distribution alternatives in back-end server clusters. The trickle-down effect suppresses the primary source of load imbalances in Web server farms, reducing the potential benefit from sophisticated load balancing strategies for static content. It also absorbs much of the locality in the request stream, reducing the benefit from downstream content caches (e.g., server memory). In the limit, upstream caches weaken the differences between request distribution strategies downstream. The various strategies are motivated largely by the locality and load skew inherent in the "head" of Zipf-like request distributions, and behave similarly on the heavy tail. Even so, the partial reduction in locality from moderately effective forward caches makes content-aware request distribution (e.g., URL hashing) even more important in downstream server clusters to benefit from the locality that remains; content-blind policies such as server load balancing (SLB) severely compromise server cache effectiveness, even with small front-end caches.

This study focuses on static content. Dynamic Web requests are traditionally not cacheable. However, with techniques such as dynamic caching [12] and fragment caching [6] [7] it is possible to cache part of the result of dynamic requests (which may apply to many different requests), gaining some benefit from locality. These effects must also be considered when studying the impacts of increased dynamic content in Web requests.

6. ACKNOWLEDGEMENTS

We would like to thank Erich Nahum of IBM Research for providing us the 1998 IBM web server traces used in the LARD study. We would also like to thank Alister Lewis-Bowen and Andrew Frank-Loron for their help in obtaining the 2001 IBM server logs.

7. REFERENCES

[1] Improving Web Server Performance with Zeus Load Balancer. `http://www.zeus.co.uk/products/lb1/`.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] J. Challenger, A. Iyengar, and P. Dantzig. A scalable system for consistently caching dynamic data. In *Proceedings of IEEE Infocom 1999 Conference*, March 1999.

[7] J. Challenger, A. Iyengar, K. Witting, C. Ferstat, and P. Reed. A publishing system for efficiently creating dynamic web content. In *Proceedings of the IEEE Infocom 2000 Conference*, March 2000.

[8] Carlos Cunha, Azer Bestavros, and Mark Crovella. Characteristics of WWW client-based traces. Technical Report 1995-010, 1, 1995.

[9] Syam Gadde. The Proxycizer Web proxy tool suite. `http://www.cs.duke.edu/ari/Proxycizer/`.

[10] Syam Gadde, Jeffrey S. Chase, and Michael Rabinovich. Web caching and content distribution: A view from the interior. *Computer Networks and ISDN Systems (Selected Papers from the Fifth International WWW Caching and Content Delivery Workshop)*, 24(2):222–231, February 2001.

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

[12] A. Iyengar and J. Challenger. Improving web server performance by caching dynamic data. In *Proc. USENIX Symp. on Internet Technologies and Systems*, December 1997.

[13] D. Muntz and P. Honeyman. Multi-level caching in distributed file systems, or your cache ain't nuthin' but trash. In *Proceedings of the Winter USENIX Technical Conference*, January 1992.

[14] V. Padmanabhan and L. Qiu. The content and access dynamics of a busy web site: Findings and implications. In *Proc. of SIGCOMM, Stockholm, Sweden, Aug. 28 – Sept. 1, 2000.*

[15] Vivek S. Pai, Mohit Aron, Gaurav Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel, 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.

[16] Duane Wessels, Traice Monk, k claffy, and Hans-Werner Braun. A distributed testbed for national information provisioning. `http://ircache.nlanr.net/Cache/`.

[17] 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.

[18] G. Zipf. *Human Behavior and the Principle of Least Effort.* Addison Wesley, 1949.