Managing Flash Crowds On The Internet[†]

Ismail Ari, Bo Hong, Ethan L. Miller, Scott A. Brandt and Darrell D. E. Long[‡]

Storage Systems Research Center
University of California Santa Cruz
1156 High Street,
Baskin School of Engineering,
Santa Cruz, CA, 95064, USA

Abstract

A *flash crowd* is a surge in traffic to a particular Web site that causes the site to be virtually unreachable. We present a theoretical model of flash crowd events and evaluate the performance of various multi-level caching techniques suitable for managing these events. Our results show that orders of magnitude savings can be achieved in client response times and server and network load during flash crowds using basic caching techniques, and even more savings can be realized with a bit more intelligence in replacement algorithms and placement of proxies.

Keywords: Flash crowds, Web server, hierarchical cache, Web proxy, adaptive caching, trace-driven simulation.

1 Introduction

A flash crowd [29] is a large spike or surge in traffic to a particular Web site[1]. Many times it is the major news-Webcast sites, such as MSNBC [7, 30] and CNN [4, 24], that experience this problem during major world events. Other times it is a previously unpopular Web site becoming extremely popular after being mentioned in a popular news feed. The latter case is called the *Slashdot effect* [31]. The end results of flash crowds are usually very poor performance at the server side and a significant number of unsatisfied clients.

Denial of Service (DoS) and Distributed DoS (DDoS) attacks [22], which are malicious requests to disrupt the normal operation of a Web service, have similar characteristics to flash crowds. Fortunately, there are ways to distinguish them from legitimate flash crowds [24] and ways to fight back via router-based "pushbacks" [22].

The cost of outsourcing the distribution of the Web site content towards the clients via Content Distribution Networks (CDNs) like Akamai [1, 5] is only justifiable for sites that experience high loads on average and expect flash crowds eventually. Our focus is on unexpected flash crowds or on flash crowds directed to "poor" Web sites that cannot afford the required hardware and software infrastructure even if they expect it to happen. These sites need a publicly available and Internet-wide infrastructure support to survive flash crowds. This infrastructure could either be a much more widely distributed form of Squid-like [17, 36] hierarchical Web proxies, or support from the network infrastructure as seen in caching routers [3, 11], or peer caching via peer-to-peer overlays [30, 31, 25].

We evaluate the multi-level caching solutions to flash crowds in this paper. Our findings show that a distributed caching infrastructure is much more effective in dealing with flash crowds than using a more powerful server. We find that one has to increase the CPU power of a Web server at least 4-5 times to deal with even moderate flash crowds, and doing this neither resolves the network bottlenecks close to the server nor reduces the client response times. Furthermore, most of the added capacities would be idle ($>$ 82%) during normal loads.

Since the unique flash content is not very large, a distributed caching infrastructure that is provisioned to provide improvement during normal loads is more than enough to handle flash crowds. The Greedy Dual Size with Frequency (GDSF) policy uses one fourth of the cache size used by LRU to achieve the comparable hit rates. Therefore, if cache amounts required

[†]This research is sponsored in part by Hewlett-Packard Laboratories, Storage Technologies Department.

[‡]{*ari,hongbo,elm,sbrandt,darrell*}*@cs.ucsc.edu*

[1]The term "flash-crowd" was first used by Larry Niven in his 1973 science-fiction story, where huge crowds would materialize in places of interesting events with the availability of cheap teleportation.

for a distributed infrastructure are a big concern then using better policies helps alleviate this problem.

We also find a trade-off between the benefits of client sharing at the upper levels and the cost of additional trip-times to get the content from there. We also find diminishing returns from sharing for larger communities similar to findings in other research [34]. Finally, using heterogeneous policies and size-based partitioning results in minor improvements over LRU-only or LFU-only policy caching; however, it does not beat GDSF-only caching.

Other contributions are: a capability to regenerate and simulate wide-scale events on the Internet, such as flash crowds for which traces are hard to obtain; and a complete simulation package with caching, networking, and a Web server model.

2 Modeling The Flash Scenario

We describe the pieces of models and tools that we developed separately and finally brought together to generate a wide-scale flash crowd scenario. We introduce the flash traffic trace generator, the Web server model, and the topology generator. Section 4 describes the network and cache simulator that glues all pieces together and enables the trace-driven simulations.

2.1 Flash Crowd Traffic Model

We define the *shock level* parameter to be equal to the *order of magnitude increase in the average request rate* seen by a Web site:

$$R_{flash} = shock\_level \times R_{normal}, \tag{1}$$

where R_{flash} and R_{normal} are the request rates of a server during the peak of a flash event (event that causes flash crowd) and the "normal" or expected-load operation, respectively. We calculate the average request rate for the normal operation from real Web traces. Plots of various real Web proxy traces reveal that the request rate of a Web proxy server varies up to 3–4 times within a normal day which makes the average request rate assumption reasonable for shock levels relatively larger than these variations.

A flash event has three major phases, a *ramp-up phase*, a *sustained traffic phase* and a *ramp-down phase* as shown in Figure 1(a). The shock level parameter determines the length of each of these phases. The flash event starts at time t_0. During the ramp-up phase of a flash event, more and more people are interested in the event, and the traffic level is raised from its normal level (in requests/second) to a sustained maximum level at time t_1. The maximum traffic level is sustained until time t_2 because people come to and leave the Web site at roughly the same rate during this period. The traffic level of this Web site gradually decreases and is back to its normal rate at time t_3.

We generally argue that the more shocking an event is the less time it takes to reach the maximum request rate of the flash event during the *ramp-up phase*. There is evidence in experimental psychology for the truth of this claim. Cicala and Corey [18] have measured the running speed of rats as a function of the magnitude of applied electrical shock. They found direct correlation between the two, but unfortunately did not provide information on the slow-down behavior. Therefore, instead of focusing on the shape of the curves we focus on the time relations between the time periods of a flash event. We choose to take the logarithm of the shock level to simulate the effect of "numbness" or disability to react proportionally to linearly increasing shock. Currently, the ramp up time, $l_1 = t_1 - t_0$, is given by:

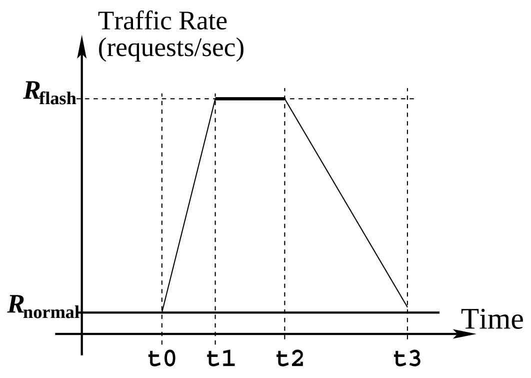
$$l_1 = \frac{1}{log_{10}(1 + shock\_level)} \tag{2}$$

and the ramp-up function is linear.

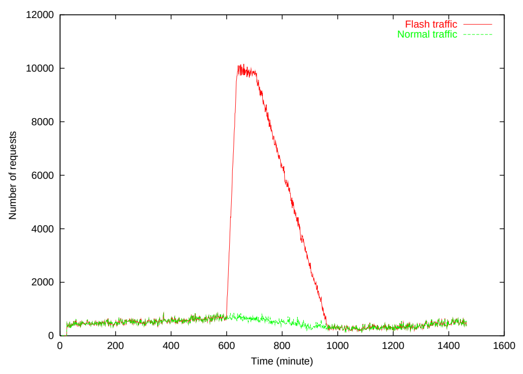
We model the second phase of the flash event with a *sustained request rate*, which is equal to R_{flash}. We argue that the magnitude of the increase in the normal request rate (*i.e.* the shock level) is an indication of more *information content* in the events that cause the flash interest. The argument is based on the observation that the flash crowds are coming to the Web site, because they are not satisfied with other news sources informing them about the event. For example, the news about the death of a famous person at an old age from natural causes may be shocking initially, but if there is nothing more to learn about the details the shock does not ramp-up to extremely high levels and does not sustain just as long. Therefore, we take the second phase to be related to the shock level and take the logarithm to slow down the amount of reaction of the crowd for the same reason as above. The sustained time $l_2 = t_2 - t_1$ is given by:

$$l_2 = log_{10}(1 + shock\_level) \tag{3}$$

In the *ramp-down* phase of a flash event, the server traffic decreases as the information in the Web site is exploited by the clients and as the interest shifts to other news sources or other events. A general observation is that the interest is lost slower than it is



(a) Flash Crowd Model



(b) Generated Flash Crowd Traffic

Figure 1: (a) Based on observations of flash traces used in previous research [30] we model a flash crowd scenario as a three-phase event: a ramp-up phase from the normal-traffic rate, R_{normal}, to a maximum flash-traffic rate, R_{flash}; a sustained traffic phase at R_{flash}; and a ramp-down phase from R_{flash} back to R_{normal} (b) Traffic levels seen in our real Web proxy trace varies around 200-500 reqs/minute. Then, the generated flash crowd traffic is superimposed on the normal traffic to model the overall flash crowd scenario. The request rates reach up to 10,000 requests/minute.

gained for the flash events with certain shock effects on the society [30]. In this model, we simply make the ramp-down time, $l_3 = t_3 - t_2$:

$$l_3 = n \times log_{10}(1 + shock\_level), \qquad (4)$$

where n is a constant. The ramp-down function is also linear.

Figure 1(b) shows the rates (in requests/minute) seen in a real Web proxy trace and the flash crowd zone. The average request rate which is ∼480 reqs/minute (8 reqs/sec) increases to around ∼10,000 reqs/minute (∼160 reqs/sec) during the flash crowd.

We use scripts to generate the flash traffic and then combine it with the traffic in the real Web trace to turn the ordinary day into a flash event. We assume independent inter-arrivals for clients and use a Poisson model for the request inter-arrival times during a flash event as done for synthetic Web proxy trace generations in related prior work [26]. The flash documents are considered to be separate from the normal traffic, so it takes a warm up time to cache these documents. We chose the objects to be fixed-size at 10 KBytes to make it easier to track the cacheability of the cumulative flash content with various cache sizes.

2.1.1 Shock Waves

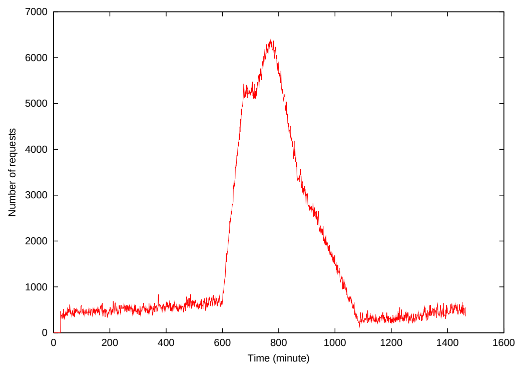
Our initial flash model considers an event with a single shocking subject. It is possible that events in real life trigger each other and multiple shocking events occur in one epoch. In this case one has to consider each event separately, use the model multiple times with time shifts and overlap the generated curves to create the combined flash effect that we call *shock waves*.

Figure 2 shows shock waves with two and three shocking subjects each created as single shocks using the model and then overlapped with time shifts from each other. MSNBC traces on September 11, 2001 [9] are similar to the shock waves shown in Figure 2.

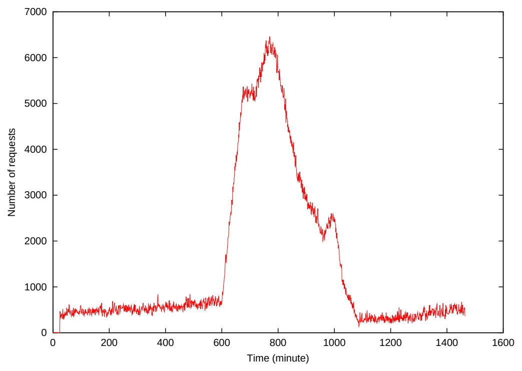
2.2 Web Server Model

We developed a Web server model based on the scalability tests done by the SPECWeb benchmarks [8] on real Web servers. SPECWeb [8] is designed to measure a system's ability to act as a Web server servicing static and dynamic page requests. SPECWeb99 measures the maximum number of simultaneous connections, thus the simultaneous requests that a server is able to support under a predefined workload. We omit models within SPECWeb for directory, size and popularity distributions of objects, since we gather this information from real Web traces. We also do not include the bit-rate restrictions enforced by the SPECWeb99 clients to the server under test.

We model a multi-threaded Web server that can scale up to simultaneously servicing a maximum number of requests. Figure 3 illustrates the model. The three parameters that we use in the model are, maximum number of *threads*, maximum *requests per second* (RPS) by each thread and the *server queue length*. The bursts of requests up to the queue length can be accepted by the server to receive service while the server threads are busy servicing prior requests. The more queuing capacity the server has the more



(a) 2-Wave Shock



(b) 3-Wave Shock

Figure 2: Our model for a single shock event can be used multiple times to create a series of events, called *shock waves*, each with a chosen shock level. Shock events with 2, 3 waves are shown here. In these figures we assume the events are related and thus reduce the shock level of the following events by the shock level of the first event.

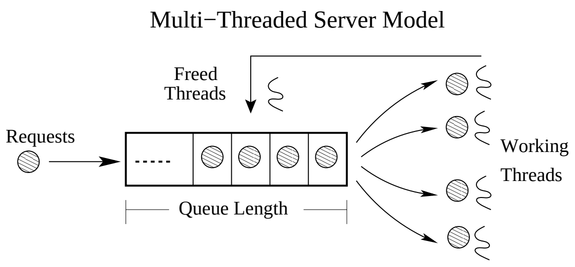


Figure 3: Model for the operation of a multi-threaded Web server.

utilized it can be under high loads. At full utilization the *server scalability*, S_{server}, is therefore approximately:

$$S_{server} = threads \times RPS/thread \qquad (5)$$

This model is analogous to a barber shop with, say, 10 seats where 10 barbers are servicing clients and 10 additional seats are reserved for incoming clients. When a barber is available he accepts an arriving client, otherwise the client sits in the waiting seats until these seats are also full. The clients arriving after the waiting seats are also full will be rejected. In Staged Event-Driven Architecture (SEDA) [32] our model is classified under “bounded thread pools” and is stated to be the right model for the Apache Web server [2] and some other Web servers.

Similarly, after the server queue is full the server rejects the incoming requests with a “SERVER_BUSY” message,[2] until a thread completes its task and is ready to service another request from the queue. Each request simply takes $1/RPS$ seconds to complete. The model is written in C++ and is embedded in our distributed cache simulator, called *multicache* [12].

[2]Similar to HTTP server error message 504.

2.3 Topology Modeling and Generation

Our network topology generator, *TopoGen*, was inspired by the Tiers network topology generator [16]. TopoGen generates wide-scale network topologies with user directed input for (1) the number of nodes in a Wide Area Network (WAN), (2) the number of Metropolitan Area Network (MAN) nodes per WAN, (3) the number of Local Area Network (LAN) nodes per MAN (4) the number of hosts in LAN, (5) the propagational delays of links between nodes, and (6) the bandwidths for each link. Delays and bandwidths are same for links of same type, *e.g.* all the links between the LAN and the MAN levels.

Figure 4 shows one of these topologies. We assume that the WAN nodes (also called GigaPOPs) are fully connected, as a mesh, to each other. In all other levels nodes are connected to other nodes in the same level via upper level nodes. We create a node adjacency matrix while parsing the topology files to avoid route calculations during the simulations.

3 Analysis Of Web Trace

National Laboratory for Applied Network Research (NLANR) operates a global cache hierarchy using Squid proxy caches and has provided important Web traces [20]. The base trace that we use to characterize normal operation for the Web server is a day long trace collected from a busy Web proxy server, SV, in the NLANR Squid cache hierarchy. It contains about 675,342 requests to 269,031 unique Web objects. The

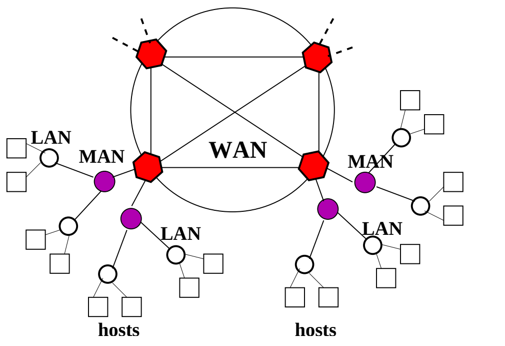


Figure 4: A sample topology generated by our topology generator, TopoGen. It has nodes at WAN, MAN, LAN and host levels. Only the WAN backbone has redundant connections. The other levels access peers at the same level via upper levels. We have not shown the MAN, LAN nodes for the upper WAN nodes for brevity and clarity.

Table 1: Parameters for the real Web proxy trace that is used to simulate the normal level traffic. This trace was then superimposed by the flash crowd traffic and used in flash crowd simulations.

Total Requests	675,342
Unique Requests	269,031
Total Bytes	4,507,009,740
Unique Bytes	1,255,894,407
Hit Rate ∞	60%
Byte Hit Rate ∞	72%
Average Document size	6,674 bytes
Max Object Size	23,087,855 bytes
Zipf Slope	0.65
Average Request Rate	8 requests/sec

characteristics of the trace are given in Table 1. We use a Web proxy trace rather than a Web server trace, due to the high traffic volumes witnessed in this proxy trace. We observed that Web server traces [6] and Web proxy traces have the similar Zipf popularity distributions.

As for the flash content we simply chose 200, 10 KB objects which can be cached by a 2 MB cache. This choice makes it easier to track how using various cache sizes, cache replacement algorithms and cache configurations may effect the end results. SPECWeb based size distributions of objects based on popularity is left as future work in this paper.

4 Simulation Setup

We explain the methodology used to combine the developed pieces together, the multicache simulator and the selected model values for the simulations.

4.1 Distribution of Trace Requests to Multiple Clients

Both the clients and the server are at the edges of the generated network topology, attached to Local Area Networks (LANs). The flash content is stored in a single server. This is one aspect of a flash crowd event, where a single server becomes subject to all the requests. All of the clients are assumed to be involved in the flash event. The requests are parsed from the trace file and sent to randomly selected hosts at the edges representing multiple clients. Then, the requests are directed by the network towards the server under test, which will be experiencing the flash crowd. Caches are placed between clients and servers at LAN, MAN, or WAN levels. Statistics are collected at all modeled entities.

The random distribution of the requests to the clients enables correlations between clients, since the unique object space is not partitioned based on client IDs. Also, sending requests from all the hosts makes the event global within the topology that we are using. It is also possible to push requests through a few clients to simulate Denial of Service (DoS) [24] or Distributed DoS (DDoS) scenarios.

4.2 Multicache Simulator

Multicache [12] is a trace-driven cache simulator developed to make the design, analysis, and comparison of cache placement and replacement algorithms in multi-level caching systems practical.

The simulator provides many building blocks such as well-known replacement algorithms (LRU, LFU, GDSF, etc. [3]), space management routines, container data structures, and simple topologies. Topologies like client-server, server-storage pairs, Web proxy hierarchies, and peer-to-peer overlays can be implemented. Modern caching related concepts such as using heterogeneous algorithms, filtering effects [33, 10], demotions [35] can also be evaluated. The code is written in C++.

4.3 Summary of Model Parameters

Table 2 gives a list of values we used during simulations to control the flash trace, Web server and the

[3]The list of algorithms implemented is around 15.

Table 2: List of values used for various parameters in the simulations.

Model	Parameter	Value
Normal Load	See Table 1.	-
Flash Load	Shock level	20
	Ramp-down constant, n	4
	Number of hot documents	200
	Hot document size	10 Kbytes
	Flash period	∼6 hours
Server	Number of threads	8
	Service rate per thread	5 reqs/sec
	Server Busy Penalty	5 secs
Topology	Nodes per WAN	2
	MAN per WAN	2
	LAN per MAN	2
	hosts per LAN	2
	WAN-to-WAN delays	50 ms
	WAN-to-MAN delays	2 ms
	MAN-to-LAN delays	1 ms
	LAN-to-host delays	0.1 ms
	WAN-to-WAN bw	1 Gbps
	WAN-to-MAN bw	100 Mbps
	MAN-to-LAN bw	4.5 Mbps
	LAN-to-host bw	10 Mbps

topology. The delays for network topology were chosen based on our `traceroute` collections from our university machines to various other universities all over the world (USA, Europe, Asia, etc.). The two WAN nodes were made to be approximately 10,000 kilometers away from each other (50 ms at the speed of light in glass). The network bandwidths (BW) were partially based on the results of a previous research [30] that reported 90% of the thousands of polled servers to have a bottleneck bandwidth less than 4.5Mbps (T3 lines).

The topology whose parameters are listed in Table 2 is illustrated in Figure 5. There are 58 directed links: the uplink and downlink are considered separately, since the loads witnessed on different directions are not equal. Uplinks transfer the small and fixed-size control packets (100 bytes) towards the server, whereas downlinks transfer the variable-size data objects that get transferred back to the clients. Figure 5 highlights the hot links in this topology. The details of network bottlenecks are discussed in Section 5.

Server model details were explained in Section 2. We add another parameter "server busy penalty" and set it to 5 seconds, a value which seems to be a reasonable wait time for getting the SERVER_BUSY message from the overly loaded server, *i.e.* the queue is full and all threads are servicing data requests.

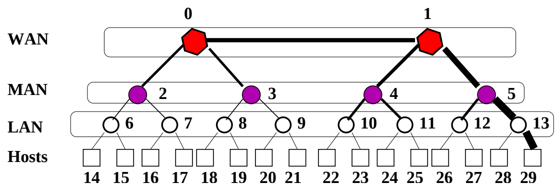


Figure 5: For both the normal and flash crowd traffic the hot links were the downlinks (from server towards clients). The ranking in decreasing traffic order was found to be directly related to the hop-distance from the server. This figure also shows the detrimental effects of a server under flash crowd traffic to its subdomain by the increased network load. Caching solutions distributed the load on networks better over the entire topology.

4.4 Replacement Policies for Caches

Time, frequency and object size are the most commonly used criteria for local replacement decisions. Least Recently Used (LRU) uses recency of access as the sole criteria for replacement, while Least Frequently Used (LFU) uses frequency or popularity of access. SIZE replaces the largest object. Greedy Dual Size with Frequency (GDSF) [14] replaces the object with the smallest key $K_i = (C_i \times F_i)/S_i + L$, where C_i is the retrieval cost, F_i is the frequency of access, S_i is the size and L is a running age factor. L is set to the key value of the objects that are replaced from the cache.

5 Results and Discussions

We measure the effects of flash crowd on Client Response Time (CRT), server load and network load in three experiments. Each experiment aims to test a different aspect of caching under the normal and the flash workloads. In Experiment 1, we use caching at the edges of the network at the LAN level only and compare LRU and GDSF policies. In Experiment 2, we evaluate the costs and the benefits of placing caches at various levels of the hierarchy. We remove caches from the LAN level and place an aggregated amount of cache to the MAN level; we repeat the procedure between the MAN and the WAN levels. In Experiment 3, we place caches at both the LAN and the MAN levels and compare three multilevel caching strategies. The first strategy is to use the same policy both in LAN and MAN levels. The second strategy is to use different or heterogeneous policies [15, 33] at different levels. The third strategy is to partition the unique document space to multiple levels in the hierarchy based on size as proposed by Williamson [33]. At the end we discuss about some

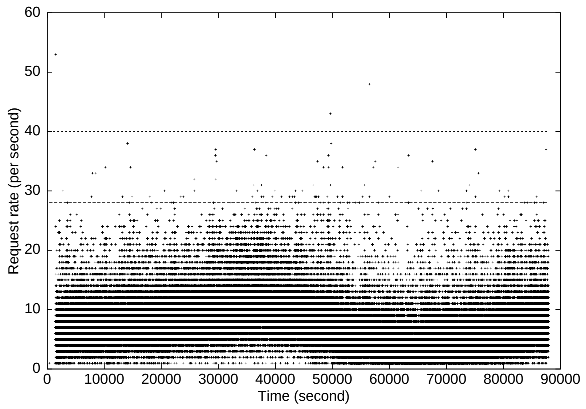


Figure 6: To discover the burstiness in the normal trace we change the time scale to increments of 1 second. Request rates observed in the real Web trace are mostly lower than 30 reqs/sec. However, very few cases between 40-60 reqs/sec are also observed.

of the recently proposed adaptive caching techniques for multi-level caches as future prospects.

5.1 Experiment 1: No Caching vs. LAN-Only Caching

We report results on caching only at the LAN level, closest to the clients. We compare Least Recently Used (LRU) and Greedy Dual Size with Frequency (GDSF) replacement policies under normal and flash loads. No other caches are used in the network. Also, node 13 in Figure 5 was never allowed to cache, since caches in our simulators can serve request coming from both directions and serving clients of other subdomains from node 13 would result in an HTTP-accelerator capability.

5.1.1 Server Load

We initially adjust the capabilities of the server to handle 99.9% of its load during normal operation even without caching in the network. Therefore, for the normal trace the percent of SERVER_BUSY responses is (100-99.9) $\leq$ 0.1% for all cache sizes[4]. We come to this value as follows. Figure 6 shows the request rates observed in the normal trace in 1 second buckets. We calculate that 99.9% of the request rates (in *requests/sec*) are under the horizontal line at 28 reqs/sec. We leave some slack capacity and choose the final server to be approximately 1.5 times more powerful than the initial server, so capable of handling 40 reqs/sec as shown by the upper horizontal line.

[4]The exact value starts at 0.03% without caching.

Table 3: Cache amounts are given as a percentage of the unique space seen at edge nodes, thus their infinite cache size. Percent of error messages returned to the clients drops drastically with increased amount of cache at the edges. Caches using GDSF replacement policy achieved more hits and reduced the load on the server quicker than LRU.

Cache Size (%)	**Unsuccessful Responses (%)**	
	flash-LRU	**flash-GDSF**
0	51.4	51.4
0.31%	11.1	9.5
0.62%	0.2	$\ll$ 0.1
1.25%	$\ll$ 0.1	$\ll$ 0.1

Next, we use the generated flash trace on the same topology and the same server. Table 3 reports the percent of incoming requests that were responded with a SERVER_BUSY response for the flash workloads with ($>$ 0MB) and without (0MB) caching for different cache sizes. When there is no caching almost half (51.4%) of the requests were responded with a busy message during flash crowd. This 51.4% number is comparable to the dynamic page phase of MSNBC September 11 trace analysis [30], that reported a 49.4% busy message value. When we use caches GDSF replacement policy achieved more hits and reduced the server load more than LRU caches. In Section 5.1.3 we show that GDSF has a slight disadvantage in terms if byte hit rates over LRU with large cache sizes and therefore it cannot reduce network loads as much as LRU.

Our flash trace has a shock level of 20 as listed in Table 2. According to equation (1) the upscaled server that aims to handle all the load during flash crowd has to handle at least (20 $\times$ 8) 160 reqs/sec. We increased the number of threads and the RPS for each thread until 99.9% of the requests in the flash crowd trace were handled. As expected, we reached this level of service when the new server had 16-20 threads each able to handle 10 reqs/sec, respectively. This means the initial server has to be upscaled 4-5 times before it can handle the flash load with a shock level of 20 and the initial average inter-arrival rate of 8 reqs/sec. Unfortunately, for 99.9% of the cases with the normal trace, more than 82% ($(160-28)/160$) of the capabilities of the upscaled server will be idle during normal operation; a cost which cannot be justified for most web sites. Therefore, caching support from the infrastructure is crucial.

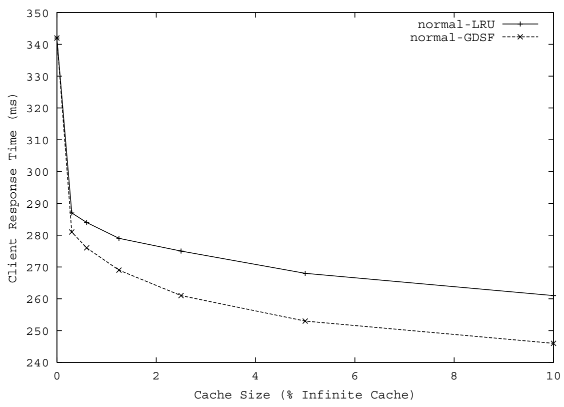


Figure 7: Client response times for GDSF and LRU algorithms using normal trace. Refer to Table 4 columns 1-2 for details.

5.1.2 Average Client Response Times (CRT)

Table 4 illustrates the average CRT for the normal and the flash workloads with and without caching at the LAN level for LRU and GDSF. The infinite cache size for the workload seen at the clients was found empirically [5], since the workload seen by each client is not equal to the complete trace. Due to random distribution of requests to the clients, the infinite cache size for the parts of the workload seen by all the clients were almost the same. Therefore, the percents in Table 4 are truly reported based on the infinite cache size for the workloads seen by each client.

Two different replacement policies, LRU and GDSF, are used to control the caches. As expected, the average CRT results for zero and infinite cache sizes are the same for both policies. For the normal load the infinite cache size can provide up to 33% (1-230/342) reduction in CRT. For other cache sizes GDSF is approximately 6% better in CRT than LRU. Note that 6% difference is significant for the normal load case, since due to diminishing returns it would take 4 times more cache space for LRU to achieve the same reductions as GDSF does for this workload. For example, Table 4 shows that LRU uses 10% cache space to achieve 261 ms CRT, while GDSF achieves 261 ms CRT with 2.5% cache space. Figure 7 plots the first two columns of Table 4 to emphasize the success of GDSF. The plot for the CRT results for the flash workload were very steep and the details were not visible, therefore we refer the readers to the results given in Table 4 for the flash trace.

For the flash trace (Table 4 columns 3–4) there are two sharp drops in average CRT levels, first from $\sim$2600 ms to $\sim$500 ms and the second is from $\sim$500 ms to $\sim$100 ms, that is as much as 25 times. These are the cache sizes at which approximately one half of the flash objects and then all of the flash objects can be cached, respectively. Note that after these two phases, the average CRT for the flash workload is much lower than the average CRT for the normal load. The two reasons for this difference are: (1) most requests for the flash contents are responded from the nearby LAN caches and (2) the number of requests to the flash contents outweigh the number of requests to normal content during the flash crowd event.

[5] Infinite cache size is 320 MB for normal workload.

Although response times for the flash crowd are expected to be extremely high, there are two concerns about how to compare the CRTs for cases including unsuccessful responses and those resulting in completely successful ones. To enable comparison we provided an empirical 5 second *busy message penalty* for the overly-loaded servers and assumed that the client would be “satisfied” with this response (*i.e.* the client considers any response a successful response) whether it is the data or the busy message. This is a model for *submissive clients*.

We also modeled *persistent clients* which repeatedly made requests to the server until they got a data response. However, this model had several disadvantages. First, it had the effect of pouring gasoline on fire during the flash crowd, since 50% of the messages returned as busy created just as many new requests back to the overly-loaded server. Second, we noticed that by doing this we changed the initial workloads both for the flash and the normal levels. Third, after the server queue was full it became a matter of luck as to which request would get a chance to receive the next available service and this increased the variation in response times. So, we chose to stay with the *submissive client* model.

The CRT values for the flash crowd trace in the network without caching are expected to be even higher in reality, since our network model does not consider packet drops and TCP level issues. The network details are discussed next.

5.1.3 Network Bandwidth Usage

Our simulations currently run at the Web document level, therefore we omit the transport layer details (TCP slow start, congestion avoidance, packet drops and retransmits, explicit congestion notifications) and simulate the transfer of objects over the network links by calculating the total transfer delay, D as:

$$D = P + (size \times 8)/(BW \times 10^6) \tag{6}$$

where P is the propagational delay in seconds (distance/speed of light), *size* is the document size in bytes and *BW* is the link bandwidth in Mbits/sec.

Table 4: Client response times in ms with and without caching for the normal and the flash traffic with LRU and GDSF replacement policies.

Cache Size	**Client Response Times (ms)**			
	Normal		**Flash**	
	LRU	**GDSF**	**LRU**	**GDSF**
0	342	342	2600	2600
0.31%	287	281	506	440
0.62%	284	276	108	96
1.25%	279	269	89	86
2.5%	275	261	87	84
5%	268	253	85	82
10%	261	246	84	80
20%	252	238	81	79
100% (∞)	230	230	75	75

Propagational delays and bandwidths of links between various levels in the hierarchy are given in Table 2.

We assumed the network nodes to have enough buffering capability to handle transmission of bursts of requests. Our assumption is valid for all request rates in the normal operation and during most of the flash crowd. However, this assumption is optimistic only for the flash workload case in the network without any caching or with very little caching. The top curve in Figure 8(b) shows this case. The only bottleneck links in the topology during flash crowd are the 4.5 (T3) link between the LAN and MAN and with less intensity the 10 Mbps link between the LAN and the server.

At 4.5 Mbps links can transfer approximately 570 KBytes/sec. With an average object size of 6-10 KBytes within the proxy trace this would mean 57-95 reqs/sec. A quick scan of Figure 6 reveals that for the normal trace all requests can be handled even by the 4.5 Mbps links. Whereas during flash crowd 160 reqs/sec are seen at the sustained peak rates. With an average size of 10KB objects 160 reqs/sec results in 12.8 Mbps data rates, which could render both the 4.5 Mbps MAN-LAN link and 10 Mbps LAN to server links to be bottlenecks. Therefore, only under the flash crowd load in the network with no or very little (0.31%) caching CRTs, which are (in Table 4 columns 3–4) already high, are expected to be much higher due to queuing in the network routers and finally due to packet drops when the router cannot buffer the bursts. Finding the exact value for the already high CRT in the flash crowd case with little or no caching is left as future work.

Figure 8 shows the traffic reduction in highly trafficed links. The links are ranked based on the amount of traffic they witness. Note that in addition to lowering the total traffic (areas under the curves), caches also provide a better load distribution over all the network links seen as a flattening on the curves. This effect is especially prominent during the flash crowds as illustrated in Figure 8(b). The load balancing property of distributed caching during flash crowds is just as critical as the CRT reductions, since the LAN, the MAN and the WAN of the server will still be reachable.

A comparison of GDSF and LRU with 5% cache size reveals that LRU is better than GDSF in reducing the network load on hot links. This is due to the lower byte hit rates of GDSF compared to LRU also recognized by various other prior work [14, 33].

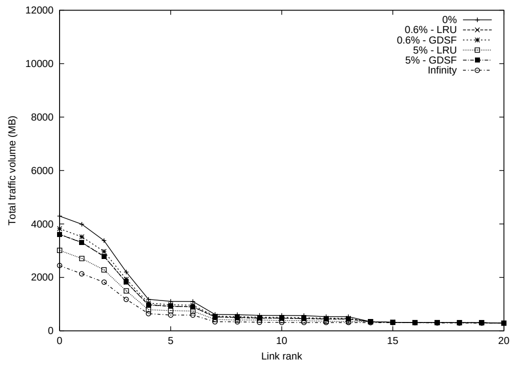
5.2 Experiment 2: Caches At MAN and WAN Levels

In this experiment we first remove caches from the LAN level and place the aggregated amount of cache at the MAN level. For example, if we had 1% (of infinite) amount of cache at the LAN level, we place a 2% at the MAN level using LRU policy in both caches. We compare cache placement choices at different levels of the hierarchy without any filtering [33, 10] at lower levels. Then, we repeat the same cache aggregation procedure between the MAN and the WAN.

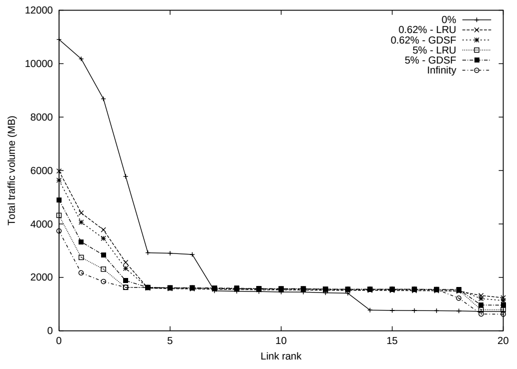
Aggregating the caches at an upper level has two benefits both due to *client sharing*. First, duplications of objects at the lower levels is avoided and some space is saved. These savings can be used to hold more unique objects, so to avoid some of the *capacity misses*. Second, since the cache serves a larger community there is a higher chance of avoiding the first-time or *compulsory misses*.

Table 5 shows the CRT results with normal and flash traces using LRU at each level. We replicate the LAN data from Table 4 for convenience of comparison. During comparison we keep the total cache size equal, ı.e. a 10% WAN cache is the aggregation of two 5% caches from MANs, which are in turn aggregation of two 2.5% caches from LAN.

In Table 5 for the normal trace we see that for all cache sizes except 100% (infinite size) the two aspects of client sharing have resulted in reductions in CRT (3-9%). The 100% cache size for the normal load requires special attention, since there are no capacity misses and the only effects on CRT are the compulsory misses and the network delays. For the normal workload we see that for LAN to MAN cache aggregation benefit of client sharing outweighed the cost of additional 1ms trip-delay to MAN and the CRT dropped from 230 ms to 157 ms (32% reduc-



(a) Normal



(b) Flash

Figure 8: The links are ranked based on the amount of traffic that passes through. The reduction in traffic on highly trafficed links via caching is more significant during the flash crowds. Another benefit of caching is the better distribution of the remaining traffic load over all network links.

tion). However, due to diminishing returns of client sharing with larger communities [34] the additional sharing gained from MAN to WAN cache aggregation did not pay off for the additional 2 ms trip-delay for each request. With WAN only caching there are also more clients (from nodes 26 through 28) that directly go to the server since the MAN cache in server's domain has been removed. This adds to the higher response times due to queuing and processing at the server.

The cumulative size of the hot data set for the flash content is much smaller compared to the normal trace content. Therefore, whenever the infinite cache size for the flash content is aggregated large reductions in CRT are found, as seen between LAN and MAN for small cache sizes and between LANs with smaller caches and LANs with larger caches. After all the flash content is cached it pays off to move this content closer to the clients. Therefore, LAN caching results in lower CRTs than MAN caching and MAN caching results in lower CRTs than WAN caching. One exception is again the MAN CRT results with 100% cache size, where sharing provides a small additional CRT saving for the proxy part of the flash trace.

In summary, we find a trade-off between *benefit of client sharing at upper levels* and *cost of additional delays spent* for caching away from the clients. Also, the cacheability of flash content plays a significant role on average CRTs.

5.3 Experiment 3: Multi-Level Caching

In this experiment we allow caching at multiple levels of the hierarchy. Therefore, upper levels are subject to the filtering effects [33, 10] from the lower level caches. We compare three major strategies: *single policy caching*, *heterogeneous caching* and *size-based partitioning*. The cache sizes are fixed to 0.62% at the LAN and 1.25% at the MAN levels, which can hold most of the hot objects if the correct policy is used.

For *single policy caching*, which means using the same replacement policy in all cache nodes in the hierarchy, we repeat the experiment for LRU, LFU and GDSF policies. Table 6 gives the CRTs for each approach under the normal and flash loads. LRU performs better than LFU under normal load, but LFU performs better than LRU under flash loads. This is because LFU can keep the popular flash content in the caches, while LRU flushes out some flash objects from the cache to place some recently retrieved non-flash objects. GDSF is the best static policy to be used in all cache levels, since it considers all of the frequency, size, and aging criteria instead of a single recency or frequency factor.

Previous research [15, 35, 13] shows the benefits of using *heterogeneous policies* at various levels in a hierarchy to avoid inclusive caching. Inclusive caching may render upper level caches useless. For heterogeneous policies we try various combinations of LRU, LFU and GDSF in different levels. We find that combinations of recency (LRU) and frequency (LFU) were not good enough to perform better than GDSF, which considers all of recency, frequency and size under one roof.

Size-based partitioning was recently proposed by Williamson [33], where the lower level caches only hold objects smaller than a certain size, *S*, and the upper level caches hold objects larger than *S*. We evaluated size-based partitioning with different replacement policies and two different *S* values, 9 KB and 12 KB.

For the SIZE replacement policy that replaces the largest size objects the 12 KB threshold decision had

Table 5: Client response times for experiment 2.

Aggregate Cache Size	**Client Response Times (ms)**					
	Normal			**Flash**		
	LAN	**MAN**	**WAN**	**LAN**	**MAN**	**WAN**
1.25%	287	278	277	506	115	180
2.5%	284	272	271	108	104	173
5%	279	265	264	89	100	169
10%	275	257	255	87	98	171
20%	268	247	244	85	96	159
100% (∞)	230	157	174	75	73	117

a detrimental effect, since the lower level cache was cluttered with objects smaller than 10KB flash objects and that the flash objects were quickly being replaced. The upper cache was unfortunately not even caching these hot objects. We dropped the threshold to 9KB to force flash objects to be cached at the upper level as these objects would be the fairly small compared to other objects in upper level, therefore would not get replaced. The CRT dropped to 114 ms. None of the results for other replacement policies using size-based partitioning performed better than their counterparts that did not use this technique. Tuning the threshold around the fixed-size flash objects and not getting additional benefits from it tells us that frequency and recency provide a more powerful ordering than a manually split ID space based on size, no matter how well the threshold is tuned.

5.4 Adaptive Algorithms

Recently proposed adaptive algorithms [27, 21, 13] can select the best of two or more algorithms at the current level of a hierarchy and under changing workloads. We have preliminary results indicating that adaptive techniques can pick the best replacement policy with a given workload [21]. We leave the application of these algorithms to flash crowd scenarios as future work.

6 Related Work

Distributed caching is a well-plowed research topic for file systems [28, 19], Web hierarchies [17] and databases. The taxonomy of solutions for flash crowds within Web context include Content Distribution Networks (CDNs) [24], peer-to-peer solutions [30, 31], and other hierarchical and distributed caching solutions as we partially analyze in this research. This paper focused on the effects of using different cache replacement algorithms, changing the placement of caches, using heterogeneous multi-level caching and partitioning the id space based on document size on flash crowd problem.

Content Delivery Networks (CDN) [1, 5] push the content that is expected to be popular towards the clients. CDNs are likely to provide load distribution and high availability [30] during both flash crowds and normal operations. However, the on average low traffic of the Web sites that experience a flash event once in a lifetime and the relatively high cost of CDNs for these sites make CDN solutions unsuitable for this case.

Jung *et al.*[24] discuss that the protection some current CDNs offer for flash crowds might be weaker than claimed. They propose an *adaptive CDN* architecture using a *dynamic delegation* technique. Also, choosing the correct server that leads to the fastest response to the client requests is still a big challenge for CDNs, especially "in the complexity of the real Internet" [23]. Therefore, an infrastructure support for widely distributed caching would help both to poor web sites and to CDNs.

When the clients causing a flash crowd at a particular Web site are a small crowd dispersed all over the Internet, proxies at local or organizational level do not filter out the load. However, because of the temporal locality of these requests, caching support from the network devices or well-distributed caches can enable the load to be slowly pulled away from the server in various directions.

Cooperative networking [30] is a peer-to-peer caching solution that complements traditional client-server and client-Web proxy communication rather than replace it. Similar to a preceding work called *pseudoserving* [25] previously registered clients "kick in during flash crowds to share the load and get out of the way when the client-server communication is working fine [30]. Backslash [31] is also a peer-to-peer caching solution, but it replaces the current Web servers and proxies to make deployment of distributed caching transparent to the clients.

IP multicast is another mechanism for the delivery of content with reduction in network traffic. IP

Table 6: CRT for various caching strategies are listed. In this experiment, when GDSF policy is used in all caches the average CRT is lower, thus better, than both heterogeneous caching and the size-based partitioning solutions.

Cache Design	**Policy**	**Client Response Time (ms)**	
		normal	**flash**
Static-All	LRU	270	90
	LFU	273	88
	GDSF	252	83
Heterogeneous	LAN(LRU), MAN(LFU)	272	88
	LAN(LFU), MAN(LRU)	268	88
	LAN(GDSF), MAN(LRU)	263	87
Size-Partition	LAN(SIZE), MAN(SIZE), 12KB	275	512
	LAN(SIZE), MAN(SIZE), 9KB	275	114
	LAN(LRU), MAN(LRU), 12KB	279	95
	LAN(LRU), MAN(LRU), 9KB	278	104
	LAN(GDSF), MAN(GDSF), 12KB	273	95
	LAN(GDSF), MAN(GDSF), 9KB	272	102

multicast does not have any redundancy in terms of sending multiple copies of packets over the same link for clients on a multicast session. However, IP multicast urges senders and receivers to be online at the same time on the same multicast IP address. Clients in a flash crowd have no agreement and start requesting the same content with various time-shifts. Distributed caching allows insertion of time-shifts between requests and is therefore superior to IP multicast during flash crowds. Adaptive Web Caching (AWC) [36] uses IP multicast to automatically configure cache groups to boost sharing among caches.

We chose to develop our own models and simulators, since other available tools were not flexible enough to fully support our research needs. Simulators such as "Wisconsin web cache simulator", DavisSim, Proxycizer, NCS and NS (links to all can be found at **http://www.web-cache.com/simulators.html**) can be used for analysis of Web workloads.

7 Concluding Remarks

We presented models for flash crowd traffic, Web servers, network topologies and brought these together in a distributed cache simulator to analyze flash crowd events and evaluated some caching solutions that alleviate this problem. We found that a distributed caching infrastructure provisioned to handle normal loads is enough to handle flash crowds, while it is not possible to achieve a complete recovery by scaling up the capabilities of the servers under flash loads. Two additional design steps resulting in significant improvements to the client response times and server and network loads are: using GDSF as the replacement policy in caches and adding caches at the MAN level to benefit from client sharing.

Our future work includes analysis of flash crowds using of variable-sized documents, more detailed network modeling, and cache consistency issues. We are also planning to compare peer-to-peer caching techniques with distributed caching solutions.

References

[1] Akamai, http://www.akamai.com.

[2] Apache Software Foundation. The Apache web server. http://www. apache.org.

[3] Cisco Content Networking: "Flash Crowd Insurance", http://www.cisco.com.

[4] CNN, http://www.cnn.com.

[5] Digital Island, http://www.digitalisland.com.

[6] Internet Traffic Archieve, http://ita.ee.lbl.gov.

[7] MSNBC, http://www.msnbc.com.

[8] SPECWeb, http://www.specweb.org.

[9] V. N. Padmanabhan, The case for CoopNet, http://research.microsoft.com/ padmanab/projects/CoopNet/.

[10] A. Amer and D. D. E. Long. Adverse filtering effects and the resilience of aggregating caches. In *Proceedings of the Workshop on Caching, Coherence and Consistency (WC3 '01)*, Sorrento, Italy, June 2001. ACM.

[11] I. Ari. Adaptive Caching using Multiple Experts (ACME) and Storage Embedded Networks (SEN). Technical Report UCSC-CRL-03-01, University of California Santa Cruz, Santa Cruz, CA, 2003.

[12] I. Ari. Multicache simulation environment version 1.0 reference guide. Technical Report UCSC-CRL-03-02, University of California Santa Cruz, Santa Cruz, CA, 2003.

[13] I. Ari, A. Amer, R. Gramacy, E. L. Miller, S. A. Brandt, and D. D. E. Long. ACME: adaptive caching using multiple experts. In *Distributed Data and Structures*, volume 4. Carleton Scientific, 2002. Extended version of the WDAS 2002 workshop paper.

[14] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin. Evaluating content management techniques for web proxy caches. In *Proceedings of the 2nd Workshop on Internet Server Performance (WISP '99)*, Atlanta, Georgia, May 1999.

[15] M. Busari and C. Williamson. Simulation evaluation of a heterogeneous web proxy caching hierarchy. In *Proceedings of the 9th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '01)*, pages 379–388, Cincinnati, OH, Aug. 2001. IEEE.

[16] K. L. Calvert, M. B. Doar, and E. W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35(6):160–163, 1997.

[17] A. Chankhunthod, P. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical Internet object cache. In *Proceedings of the 1996 USENIX Annual Technical Conference*, San Diego, CA, 1996.

[18] G. A. Cicala and J. R. Corey. Running speed in the rat as a function of shock level and competing responses. *Journal of Experimental Psychology*, 70:436–437, 1964.

[19] M. Dahlin, R. Wang, T. Anderson, and D. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *Proceedings of the 1st Symposium on Operating Systems Design and Implementation (OSDI)*, pages 267–280, Nov. 1994.

[20] S. G. Dykes and K. A. Robbins. A viability analysis of cooperative proxy caching. In *INFOCOM*, pages 1205–1214, 2001.

[21] R. Gramacy, M. Warmuth, S. A. Brandt, and I. Ari. Adaptive caching by refetching. In *Proceedings of the 2002 Neural Information Processing Systems (NIPS)*, 2002. Accepted for publication.

[22] J. Ioannidis and S. M. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Proceedings of Network and Distributed System Security Symposium*, San Diego, CA, Feb. 2002.

[23] K. L. Johnson, J. F. Carr, M. S. Day, and M. F. Kaashoek. The measured performance of content distribution networks. *Computer Communications*, 24(2):202–206, 2001.

[24] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites. In *Proceedings of the 11th International World Wide Web Conference*, pages 252–262. IEEE, May 2002.

[25] K. Kong and D. Ghosal. Mitigating server-side congestion in the Internet through pseudoserving. *IEEE/ACM Transactions on Networking*, 7(4):530–544, Aug. 1999.

[26] N. Markatchev and C. Williamson. Webtraff: A GUI for web proxy cache workload modeling and analysis. In *Proceedings of the 10th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '02)*, pages 356–363, Fort Worth, TX, Oct. 2002.

[27] N. Megiddo and D. S. Modha. ARC: A Self-Tuning, Low Overhead Replacement Cache. In *Proceedings of the 2003 Conference on File and Storage Technologies (FAST)*, pages 115–130, San Francisco, CA, Mar. 2003.

[28] M. N. Nelson, B. B. Welch, and J. K. Ousterhout. Caching in the Sprite network file system. *ACM Transactions on Computer Systems*, 6(1):134–154, 1988.

[29] L. Niven. *Short story "Flash Crowd" in the book "The Flight of the Horse"*. pages 99-164. http://www.larryniven.org, 1973.

[30] V. N. Padmanabhan and K. Sripanidkulchai. The case for cooperative networking. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS'02)*, pages 178–190, Cambridge, MA, USA, March 2002.

[31] T. Stading, P. Maniatis, and M. Baker. Peer-to-peer caching schemes to address flash crowds. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS'02)*, pages 203–213, Cambridge, MA, USA, March 2002.

[32] M. Welsh, D. E. Culler, and E. A. Brewer. SEDA: An architecture for well-conditioned, scalable internet services. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 230–243, Banff, Alberta, Canada, 2001.

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

[34] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. R. Karlin, and H. M. Levy. On the scale and performance of cooperative web proxy caching. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP '99)*, pages 16–31, 1999.

[35] T. M. Wong and J. Wilkes. My cache or yours? making storage more exclusive. In *Proceedings of the 2002 USENIX Annual Technical Conference*, pages 161–175, Monterey, CA, June 2002. USENIX.

[36] L. Zhang, S. Floyd, and V. Jacobson. Adaptive Web caching. In *2nd International WWW Caching Workshop*, Boulder, Co, June 1997.