

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
ari,hongbo,elm,sbrandt,darrell@cs.ucsc.edu

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 model of flash crowd events and evaluate the performance of various multi-level caching techniques suitable for managing these events. By using well-dispersed caches and with judicious choice of replacement algorithms we show reductions in client response times by as much as a factor of 25. We also show that these caches eliminate the server and network hot spots by distributing the load over the entire network.

1 Introduction

A flash crowd is a large spike or surge in traffic to a particular Web site. Major news Web sites experience this problem during major world events. Sometimes unpopular Web sites instantly become extremely popular after being mentioned in a popular news feed. This is also called the *Slashdot effect* [7]. The end results of flash crowds are usually very poor performance at the server side and a significant number of unsatisfied clients.

Our focus is on unexpected flash crowds or on flash crowds directed to “poor” Web sites that cannot afford outsourcing the distribution of their Web content via Content Distribution Networks (CDNs) even if they expect this to happen eventually. To survive flash crowds these sites need a publicly available and Internet-wide infrastructure support such as widely distributed hierarchical Web proxies, caching networks [2], or peer caching via peer-to-peer overlays [6, 7].

Related work on flash crowds within Web context include solutions using Content Distribution Networks (CDNs) [5] and peer-to-peer overlays [6, 7]. We focus on the hierarchical caching solutions and quantify the effects of using different cache replacement algorithms, changing the placement of caches, using heterogeneous multi-level caching and size-based partitioning the document space.

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

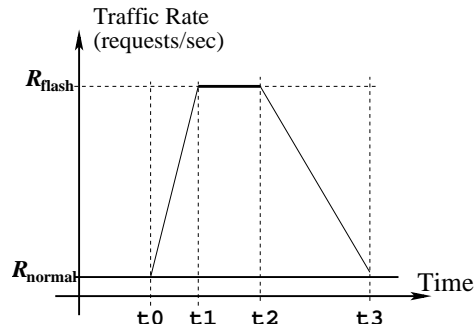


Figure 1: A model of flash crowd traffic.

2 Modeling the Flash Scenario

Our model for the flash crowd scenario consists of models for the flash traffic, a Web server, and a hierarchical topology that glues the clients, caches and the server together.

2.1 Flash Crowd Traffic Model

Several Web server traces containing flash crowds have been analyzed and presented in previous research [4, 6, 5]. Our proposed flash crowd model captures the most prominent flash crowd characteristics observed in these traces. Possible extensions to this model are mentioned at the end of this section.

We first define the *shock level* parameter to be equal to the *order of magnitude increase in the average request rate* [5, 4] seen by a Web site. Hence, if R_{normal} is the average daily (non-flash) load expected by the server in requests/sec and R_{flash} is the load on a server during the peak of a “flash event” (event that causes the flash crowd), then $R_{flash} = shock_Level \times R_{normal}$. We calculate the average request rate for the normal load from real Web traces.

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. The shock level parameter determines the length of each of these phases.

The flash event starts at time t_0 . In the ramp-up phase more and more people are interested in the event, and the traffic level rises from R_{normal} to a maximum level, R_{flash} , at time

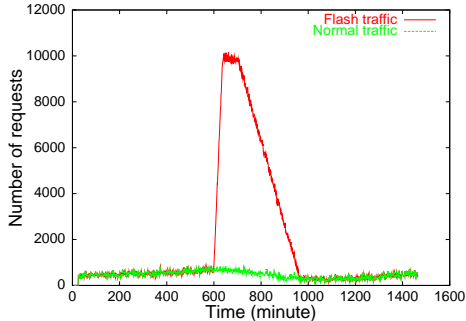


Figure 2: The flash workload generated by applying the model over “normal” (non-flash) Web server traffic.

t_1 . The rate is sustained at this level until time t_2 , since 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*. We choose to take the logarithm of the shock level to simulate the effect of “numbness” or inability to react proportionally to linearly increasing shock. Currently, the ramp up time, $l_1 = t_1 - t_0$, is given by $l_1 = \frac{1}{\log_{10}(1+shock_level)}$ and the ramp-up function is linear.

The second phase is also related to the shock level and we use 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 $l_2 = \log_{10}(1 + shock_level)$.

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. In this model, the ramp-down time $l_3 = t_3 - t_2$ is given by $l_3 = n \times \log_{10}(1 + shock_level)$, where n is a constant. The ramp-down function is also linear.

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 in the NLANR Squid cache hierarchy. It contains about 675,342 requests (≈ 4.20 Gbytes) to 269,031 unique Web objects (≈ 1.17 Gbytes). The infinite hit rate is 60% and infinite byte hit rate is 72%.

Figure 2 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/min (≈ 8 reqs/sec) increases to around $\approx 10,000$ reqs/min (≈ 160 reqs/sec) during the flash crowd with a *shock_level* of 20. We assume independent inter-arrivals for clients and use a Poisson arrival model for requests during a flash event. We choose 200 documents to be subject to the flash and fix their size at 10 KBytes. This makes it easier to track the cacheability of the cumulative flash content with various cache sizes.

Our initial flash model considers an event with a single shocking subject. It is possible that events in real life trigger

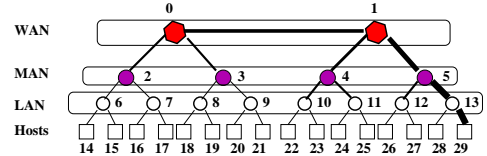


Figure 3: Topology used for the simulation. The thicker lines are subject to higher traffic.

each other and multiple shocking events occur in one epoch in the form of *shock waves*. Other interesting parameters that could be included in the model are burstiness of requests and the client dependencies.

2.2 Web Server Model

We developed a multi-threaded Web server model based on the scalability tests done by the SPECWeb benchmarks [1] on real Web servers. This server can simultaneously service a specified maximum number of requests. The three parameters that we use in the model are, maximum number of threads (*threads*), maximum requests per second completed by each thread (*RPST*), and the server queue length (*qlen*). 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. Higher queuing capacity in the server results in better utilization under high loads. At full utilization the *server scalability*, S_{server} , is therefore approximately $S_{server} = threads \times RPST$. This is a simple, yet accurate model of the Apache Web server and some other Web servers.

2.3 Topology Modeling and Generation

We generate 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 a LAN, (5) the propagational delays, and (6) the bandwidths for each link. Delays and bandwidths are the same for links of the same type, *e.g.* all the links between the LAN and the MAN levels are the same.

The topology used in the simulations is illustrated in Figure 3. There are 58 directed links. Figure 3 highlights the hot links in this topology. The network delays were based on our **traceroute** collections from our university machines to different universities all over the world. The $\langle delay, bandwidth \rangle$ parameters for links are as follows: WAN-to-WAN $\langle 50ms, 1Gbps \rangle$, WAN-to-MAN $\langle 2ms, 100Mbps \rangle$, MAN-to-LAN links $\langle 1ms, 4.5Mbps \rangle$, and LAN-to-Host $\langle 0.1ms, 10Mbps \rangle$.

Both the clients and the server (host 29) are at the edges of the topology (Fig.3). The flash content is stored in a single server as this is one aspect of flash crowd events. All clients

are assumed to be involved in the flash event. Requests from the trace file are assigned to randomly selected clients. These requests are then directed by the network towards the flash server. Caches are on the paths from clients to the server and a cache replacement policy (e.g. LRU, LFU, GDSF, etc) is assigned to each caching node by the simulator. Detailed description of replacement policies can be found in previous work [3, 2].

3 Results and Discussions

We measured the effects of flash crowds on Client Response Times (CRTs), server and network loads in three experiments.

3.1 Replacement Policies

In this experiment we compare LRU and GDSF policies with caches only at the LAN level. Using the initial normal Web trace we calculate that a server with a 28 requests/sec capacity would be able to handle 99.9% of the requests. We leave some slack capacity and choose the final server to be capable of handling 40 reqs/sec (8 threads \times 5 RPST). Our goal is to show that even the upgraded server can not cope with the flash. The server queue length is 8 and when the queue is full “server busy” messages are returned after 5 seconds.

Next, we use the generated flash trace on the same topology and the same server. When there is no caching, almost half (51.4%) of the requests receive a “server busy” message. This 51.4% number is comparable to the result of dynamic page phase of MSNBC September 11 trace analysis [6], which reported a 49.4% busy message value. When we use caching at the LAN level, the GDSF replacement policy achieved more hits and reduced the server load more than caches using LRU. For example, with 0.31% of infinite cache size LRU results in 11.1% “busy” responses to clients and whereas this value is 9.5% for GDSF.

Since our flash trace has a shock level of 20, R_{flash} is 160 reqs/sec (20 \times 8 reqs/sec). This means the initial server at 40 req/sec capacity has to be upscaled 4-5 times before it can handle this flash load. Unfortunately, for 99.9% of the cases with the normal trace, more than 82% ($\frac{160-28}{160}$) of the capabilities of the upscaled server will be idle. The server solution also can not alleviate the network bottlenecks that cause extra delays.

Also with an average object size of 10 KB the 160 req/sec would result in a 12.8 Mbps data rate that renders both the 4.5 Mbps MAN-LAN link and 10 Mbps LAN to server link to be bottlenecks. Therefore, only under the flash crowd load in the network with no or very little (0.31%) caching, CRTs, which are already high, are expected to be much higher due to queuing and finally packet drops in the routers.

Table 1 illustrates the average CRTs for the normal and the flash workloads for different cache sizes at the LAN level.

Table 1: Client response times 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

LRU and GDSF policies are compared. For the normal load the infinite cache size can provide up to 33% ($1 - 230/342$) reduction in CRT. GDSF is approximately 6% better in CRT than LRU and this difference is significant, since due to diminishing returns it would take 4 times more cache space for LRU to achieve the same reductions as GDSF. As for the flash trace, there are two sharp drops in average CRT levels, first from ≈ 2600 ms to ≈ 500 ms and the second is from ≈ 500 ms to ≈ 100 ms, which is as much as 25 times. These are the cache sizes at which approximately half and then all of the flash objects can be cached, respectively.

Figure 4 shows the traffic reduction in high-traffic links during flash crowd. 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 as seen by a flattening of the curves. When the load is distributed over the entire network the other servers in the LAN, MAN and WAN of the server will still be reachable.

A comparison of GDSF and LRU with no less than 5% cache size (Fig.4 surprisingly 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, which is also recognized by various other prior work [8, 2].

3.2 Placement of Caches

In this experiment we evaluate the costs and benefits of placing caches at various levels of the hierarchy. We remove caches from the LAN level and place an aggregated amount of cache at the MAN level; we repeat the procedure between the MAN and the WAN levels.

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 that can be used to hold more unique objects to avoid some of the *capacity misses*. Second, since the aggregated cache serves

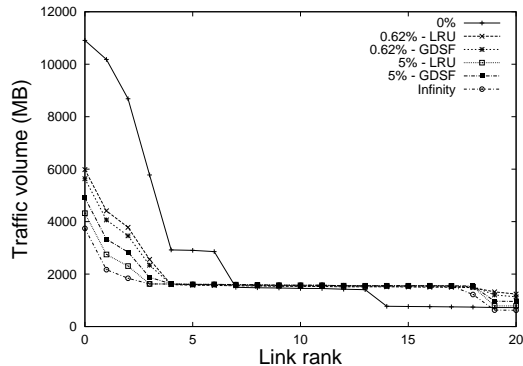


Figure 4: Caching causes both reductions in and better distribution of the network load.

a larger community there is a higher chance of also avoiding the *compulsory* or first-time misses.

Due to limited space we summarize our findings here. For the normal trace and with LRU policy we found that for all cache sizes (except infinite) the two aspects of client sharing resulted in CRT reductions by 3–9%. For LAN to MAN cache aggregation the benefit of client sharing outweighed the cost of additional 2ms round-trip-delay to MAN and the CRT dropped from 230 ms to 157 ms (32% reduction). However, due to diminishing returns of client sharing within larger communities the MAN to WAN cache aggregation did not pay off for the additional 4 ms round-trip-delay for each request. For the flash trace whenever the infinite cache size for the flash content was met via aggregation, *i.e.* all flash content was cached, large reductions in CRT were found. In summary, we found a trade-off between the benefit of client sharing at upper levels and the cost of additional delays spent for caching away from the clients.

3.3 Multi-Level Caching

In this experiment we allow caching at multiple levels of the hierarchy. 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.

Table 2 shows that LFU which can keep popular objects performs better than LRU during flash crowds. 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. For heterogeneous policies we find that combinations of only recency (LRU) and frequency (LFU) together (only one shown for brevity) were not good enough to perform better than GDSF for the same reason.

Size-based partitioning was recently proposed by Carey Williamson [8], where the lower level caches only hold ob-

Table 2: Client response times for Section 3.3.

Method	Policy	CRT (ms)	
		normal	flash
Static	LRU	270	90
	LFU	273	88
	GDSF	252	83
Heterog.	LRU-LFU	272	88
	GDSF-LRU	263	87
Size-Part.	SIZE-SIZE-12KB	275	512
	SIZE-SIZE-9KB	275	114
	LRU-LRU-12KB	279	95
	GDSF-GDSF-9KB	272	102

jects smaller than a certain size, S , and the upper levels hold the rest. For the SIZE replacement policy that replaces the largest size objects, the 12 KB threshold decision had a detrimental effect. It cluttered the lower level cache with objects smaller than 10KB flash objects. The upper cache was, unfortunately, not even caching these hot objects. We also tried other replacement policies with size-based partitioning (some omitted here), but could not perform better than their counterparts that did not use this technique. Our conclusion is that frequency and recency metrics provide a more powerful ordering than a manually split document space based on size, no matter how well the threshold is tuned.

4 Concluding Remarks

We presented a model for flash crowd traffic scenarios and evaluated some caching solutions that alleviate this problem. We found that a caching infrastructure provisioned to handle normal Web loads can be enough to handle flash crowds. Upgrading the servers does not help, GDSF is the best the replacement policy among all the tested and finally adding caches at the MAN level has benefits due to client sharing.

References

- [1] SPECWeb, <http://www.specweb.org>.
- [2] 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.
- [3] 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.
- [4] M. Arlitt and T. Jin. A workload characterization of the 1998 world cup web site. *IEEE Network*, 14(3):30–37, May 2000.
- [5] 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.

- [6] 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.
- [7] 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.
- [8] C. Williamson. On filter effects in web caching hierarchies. *ACM Transactions on Internet Technology (TOIT)*, 2(1):47–77, 2002.