Going Viral: Flash Crowds in an Open CDN

Patrick Wendell[*]
U.C. Berkeley
Berkeley, CA, USA
pwendell@eecs.berkeley.edu

Michael J. Freedman
Princeton University
Princeton, NJ, USA
mfreed@cs.princeton.edu

Abstract

Handling flash crowds poses a difficult task for web services. Content distribution networks (CDNs), hierarchical web caches, and peer-to-peer networks have all been proposed as mechanisms for mitigating the effects of these sudden spikes in traffic to under-provisioned origin sites. Other than a few anecdotal examples of isolated events to a single server, however, no large-scale analysis of flash-crowd behavior has been published to date.

In this paper, we characterize and quantify the behavior of thousands of flash crowds on CoralCDN, an open content distribution network running at several hundred POPs. Our analysis considers over four years of CDN traffic, comprising more than 33 billion HTTP requests. We draw conclusions in several areas, including (i) the potential benefits of cooperative vs. independent caching by CDN nodes, (ii) the efficacy of elastic redirection and resource provisioning, and (iii) the ecosystem of portals, aggregators, and social networks that drive traffic to third-party websites.

Categories and Subject Descriptors: C.4 [**Performance of Systems**]: Measurement Techniques

General Terms: Measurement, Performance

Keywords: content distribution networks, flash crowds

1. INTRODUCTION

Volatile request rates complicate the delivery of Internet services. Most websites experience significant diurnal fluctuations in request volume. Over shorter time scales, traffic may fluctuate due to the changing popularity of content or localized events. The most severe (and least predictable) type of volatility are *flash crowds*, which occur when services see sudden, large, and often unforeseen increases in request rates. Flash crowds are infamous for overwhelming even well-provisioned services, causing HTTP clients to timeout when trying to access starved server resources.

[*]Work performed while author was studying at Princeton University.

To insure themselves against the risk of a flash crowd, sites can outsource static content distribution to CDNs such as Akamai, which distribute traffic surges over a global network of content caches. This strategy, however, does not generally address dynamically generated content, where CPU cycles are required to construct user-customized pages. To fill this gap, services are increasingly using cloud infrastructure (such as Amazon EC2) to scale dynamic content creation in a similar fashion. Provisioning new resources in the cloud, while not instantaneous, can be done orders-of-magnitude more quickly than purchasing and installing new hardware.

While flash crowds have long been used as anecdotal motivation for elastic-computing platforms, little research exists characterizing such events in an operational CDN. To address this limitation, this paper analyzes flash crowds based on four years of data from CoralCDN [4, 5], an open CDN comprised of globally cooperating proxies. Using data from thousands of flash crowds detected amongst hundreds of thousands of domains, it answers several questions about the nature of flash crowds, both from the perspective of an online service and from that of a large content distributer.

This paper uses flash-crowd data to shed light on three broad aspects of online service delivery.

Cooperative Caching. In the best case, websites use a global network of caches to prevent origin overload during flash crowds. Whether distributed caches benefit from cooperation, as opposed to each cache fetching content individually from the origin, depends on the rate of requests and their distribution over caches. Section 3 evaluates the benefits of cooperative caching for real crowds.

Request Redirection and Elastic Provisioning. When unprecedented surges of traffic hit a service, it is often necessary to provision new resources or re-allocate existing ones to handle the requests. Services that operate on cloud infrastructure, for instance, might want to spin up additional virtualized instances to further spread load. Large content distributors with a fixed infrastructure may wish to allocate a greater portion or number of their caches to a particular customer [14]. Peer-to-peer systems may wish to build distribution trees to replicate content quickly [5, 10, 13]. In Section 4, we analyze how much time these parties might have to adapt their computing footprint in the face of flash crowds.

Source Attribution. Third-party portals such as Slashdot are often thought to be the primary drivers of flash crowds. These sites operate as centralized "bulletin boards," sharing a small set of links amongst large numbers of simultaneous viewers. In Section 5, we briefly consider how often these portals play a role in flash crowds and assess whether decentralized, peer-to-peer link sharing (such as Facebook or Twitter updates) creates equivalent request surges.

Before performing such analysis, we first describe our CoralCDN data set and provide the working definition of a flash crowd that we use throughout our analysis.

1.1 4 Years of Data from CoralCDN

CoralCDN is a open, cooperative content distribution network, designed to proxy content requests for heavily loaded services. It operates on a set of globally distributed machines that act as caching proxies. Users fall broadly into three categorical use cases. First, HTTP clients can explicitly access a piece of content through CoralCDN by "Coralizing" an existing URL (appending `nyud.net` to any URL's domain). This is a popular practice for links posted on major portals, as the Coralized link helps prevent the origin server from becoming overwhelmed by referred traffic. Second, some sites persistently use these URLs for their embedded content, so that static files are transparently requested via CoralCDN. Third, some sites use HTTP redirects to offload traffic to CoralCDN during periods of heavy load. When possible, we distinguish between these scenarios.

CoralCDN proxies persistently log data for each HTTP request they service. The logs include the URL being accessed, any HTTP referrers, the client IP address, and several other request attributes. The results in this paper are drawn from four years of aggregated CoralCDN logs, from 2007 to early 2011. These logs cover 33,354,760,349 individual HTTP requests and represent a complete (not sampled) trace of all CoralCDN traffic during this period. That said, a small number of logs are missing due to periodic failures of CoralCDN proxies and its logging subsystem [4].

CoralCDN sees a biased cut of HTTP traffic, and in some cases, this limits our exploration of the data. Since websites might use CoralCDN intermittently, for instance, we cannot readily perform longitudinal analysis across individual domains, *e.g.*, to characterize the frequency and scale of flash crowds that an average domain would experience over the course of a year. We leave such analysis to future work and carefully limit the scope of our inference.

2. DEFINING FLASH CROWDS

While the term "flash crowd" is in widespread use, prior work has generally not defined what constitutes such a crowd (systems proposals can be qualitative, and previous measurement studies [6] only had traces of one or two such crowds). To detect crowds in such a large dataset, we were pressed to craft a working mathematical definition of "flash crowd." We present this definition both to describe our own methodology and to standardize future research efforts.

2.1 Defining Flash Crowds

We define a *flash crowd* as a period over which request rates for a particular *fully-qualified domain name* are increasing exponentially. Call r_{t_i} the average per-minute request rate over a particular time period t_i. We say a domain is experiencing a flash crowd if

$$r_{t_i} > 2^i \cdot r_{t_0}\ , \quad \forall i \in [0, k]$$

and

$$\max_i r_{t_i} > m \quad \wedge \quad \max_i r_{t_i} > n \cdot r_{avg}$$

where constant m is a minimum per-minute request rate, and the constant n specifies how much more the maximum sustained rate must be than the service's average rate. Effectively, this requires that the crowd has occurred over a modest number of periods (at least k) and resulted in a larger terminal rate (m) that is many times the mean (n). Choices of k, m, and n are subjective, although in this study, we use $k = 3$, $m = 500$ reqs/min, and $n = 15$. The use of such constants may seem ad-hoc, but they are necessary in any derivative-based definition in order to filter out small fluctuations. We continue to add subsequent periods until we have seen τ periods ($\tau = 3$) for which

$$r_{t_j} \cdot 2^{-1} < r_{t_{j+1}} < r_{t_j} \cdot 2$$

Many unique URLs may belong to the same crowd, *e.g.*, they represent embedded content in a web page. However, because they may be served by different origin servers or CDNs, we conservatively consider requests to different fully-qualified domains (*e.g.*, a.example.com and b.example.com) as belonging to distinct crowds.

2.2 Request Rate Epochs and Crowd Speed

Thus far, we have not specified the granularity at which we measure request rates. Adjusting the epoch period for request rate calculations affects the type of crowd we detect. At shorter epochs, such as 60 seconds, we detect crowds that increase very rapidly but have a limited duration. For instance, if r_{t_0} were to be just 1 req/min, a crowd lasting 30 minutes must have a terminal request rate $r_{t_{30}}$ of more than one billion requests per minute to satisfy this definition. On the other hand, crowd detection with longer epochs will present a less extreme growth rate.

In this study, we consider crowds using six different epoch periods: 1, 2, 5, 10, 30, and 60 minutes. (Crowds detected using shorter epochs provide less time for a service to react to its increased demand, a problem explored in Section 4.) Using these definitions, we identify 3,553 crowds in the CoralCDN trace. Of these, 1,052 crowds originate from fewer than 10 client IP addresses each; manual inspection of some of these traces uncovered various undesired uses, such as attempted password cracking. Crowds of this nature likely would not occur in a closed, commercial CDN, given their increased security protections. Consequently, with the exception of Figure 3, our subsequent analysis filters out these crowds and only considers the remaining 2,501.

3. FLASH CROWD CACHEABILITY

3.1 Caching Challenges

During flash crowds, a CDN often acts as an intermediary between a large distributed base of clients and a small number of origin servers; ensuring efficient communication with both of these parties poses distinct challenges. On the demand side, CDNs need to proportionately distribute thousands of simultaneous requests across a network of caches. Simultaneously, a CDN must judiciously fetch content from the origin server(s), being careful to avoid simply relaying the brunt of the load to the origin.

How large are the flash crowds that CoralCDN observes and how amenable are they to caching? At the minimum, each unique URL must be fetched at least once from the origin. Crowds with many unique URLs—for example, due to different URLs representing distinct dynamic content—can incur greater origin load. Figure 1 plots the flash crowds analyzed in this study, comparing the number of unique URLs per crowd (given on the x-axis) with their peak request rate to CoralCDN (given on the y-axis). If a crowd is identified under more than one epoch size (per Section 2.2), we record it multiple times in this analysis. The upper right quadrant of the figure represents crowds that are both massive in scale and distributed over large numbers of URLs.

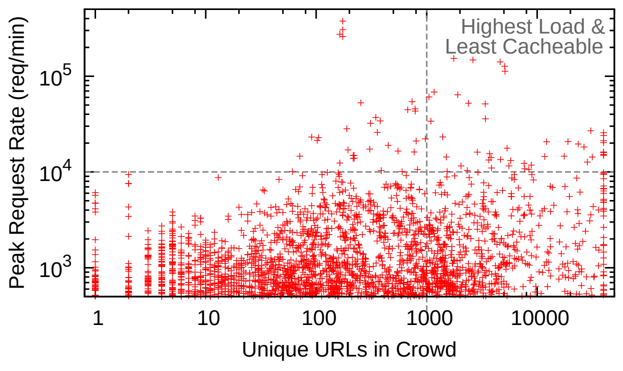

Figure 1: Size and cacheability of 2,501 flash crowds.

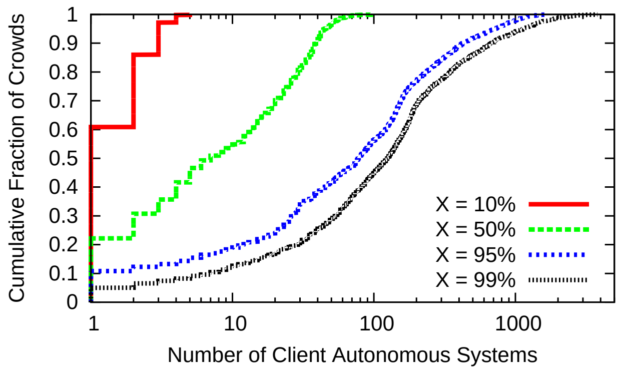

Figure 2: AS-level clustering of client requests. Each line plots a CDF of the number of ASes with clients that generate $X\%$ of crowds' requests.

3.2 Comparison of Caching Strategies

To minimize origin load and decrease client perceived latency, CDNs cache content widely across their networks. While they may employ a variety of caching strategies, one fundamental choice is the degree to which caches coordinate in fetching origin content. CoralCDN uses *cooperative caching*, where proxies discover and fetch content from other caching proxies, before requesting content from the origin. CoralCDN uses a distributed hash table for global content discovery, while earlier cooperative approaches were based on more static cache hierarchies [2]. In contrast, many commercial CDNs, including Akamai [7, 9], primarily use *non-cooperative caching*, where each remote proxy (or proxy cluster) independently fetches content from the origin site. A continuum of caching strategy exists with varying degrees of cache coordination. For clarity of comparison, however, we restrict our analysis to these two extremes.

Cooperative caching yields strictly fewer requests to the origin server than non-cooperative approaches, since all but the first request for each URL are served from cache. At the same time, cooperation increases system complexity and adds coordination overhead. In this section, we evaluate the performance benefit of cooperative caching in observed crowds. We also identify factors which influence how much benefit cooperative caching will provide to a particular crowd.

First, the client distribution over proxies plays an important role. When clients are concentrated at a small number of proxies (perhaps due to their locality), relatively few distinct caches fetch content from the origin, diminishing the benefits of cooperation. On the other hand, crowds distributed over large numbers of caches pose a higher risk of overloading the origin if caches do not cooperate.

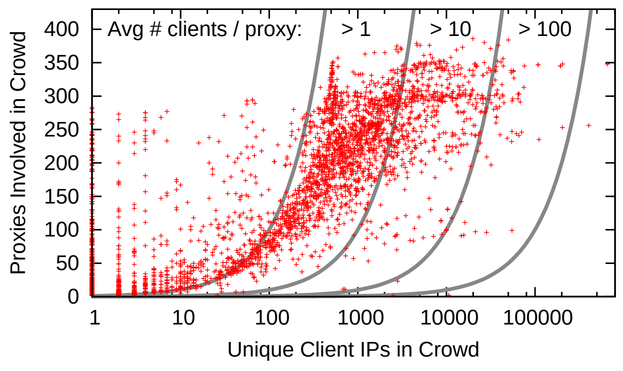

Figure 3: Number of proxies and clients involved in each flash crowd. Guide lines plot the curves for 1, 10, 100, and 1000 clients per proxy.

Our crowd traces support the latter scenario—clients are distributed across many network locations—suggesting that cooperation could be helpful. To illustrate, Figure 2 shows the clustering of clients based on network topology. While 20% of crowds have half of their requests originate from clients within a single Autonomous System (AS), crowds commonly involve hundreds of ASes. Similarly, many CoralCDN proxies are involved in serving a crowd, as shown in Figure 3, and requests are well distributed over them. This unfiltered figure shows all 3,553 crowds, including those with only a few clients, likely malicious, accessing large numbers of proxies.

The concentration of a crowd's URLs can also affect its cooperative cacheability. To characterize this less intuitive factor, we calculate the number of requests seen by each crowd's origin under both non-cooperative and ideal cooperative caching (under the latter, the origin sees a single hit per URL). In this analysis, we make the simplifying assumption that caches do not evict a crowds' content during its lifetime. CoralCDN typically operates at around 300 servers world-wide (spread over around 150 POPs), and the arrival of clients to each server was drawn directly from actual crowd traces. Since the distribution of requests amongst caches determines cache locality, using real request traces is necessary for correct evaluation.

Figure 4 demonstrates that content dynamism (the number of unique URLs) has a significant impact on the relative benefits of cache cooperation. The top graph plots the fraction of cache hits under cooperative and non-cooperative strategies, as a function of URL distribution. The lower graph plots the difference in these numbers, highlighting the normalized reduction in origin load under cooperation. Its shape reveals an interesting pattern: In crowds with a small number of unique URL's, cooperative caching provides relatively little incremental benefit, since each cache retrieves the critical set of URLs from the origin in a small number of hits. On the other extreme, crowds with mostly unique URLs—typically dynamic content differentiated for each client—receive no additional benefit from cooperative caching. Such crowds are fundamentally uncacheable, since no URL is ever repeated.

The lower graph in Figure 4 reveals a "sweet spot" for cooperative caching. In this region, cooperation has the potential to decrease origin load substantially, often by more than half the size of the crowd itself. These results suggest that the decision to introduce cooperation in CDNs should depend on the types of crowds that a CDN expects to handle. If it is somehow known *a priori* that crowds will be concentrated over just a small number of unique objects, then cooperation is likely unnecessary. Figure 5 illustrates these results in CDF form: about 30% of crowds see less than a 10% reduction

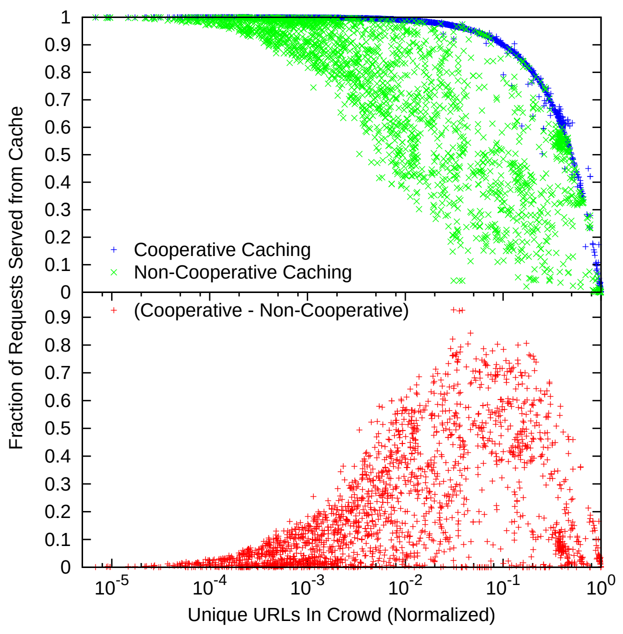

Figure 4: ***Top:*** **Percentage of request served from cache using cooperative (dark blue) and non-cooperative (light green) caching strategies.** ***Bottom:*** **Fractional increase in cache hits from cache cooperation.**

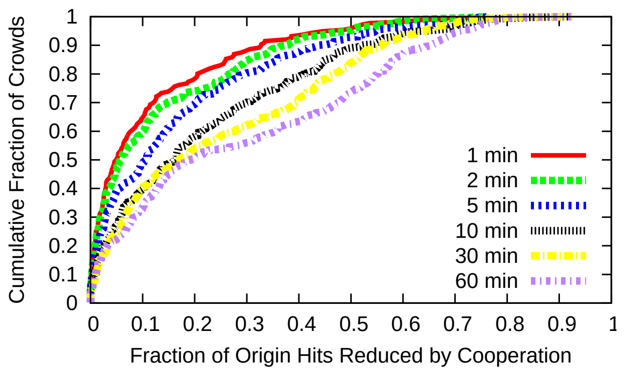

Figure 5: Origin load reduction from cooperation, as a fraction of total requests, for crowds defined by different epoch lengths.

in origin load, while 20% see between a 20% and 55% reduction in origin load. Further, slower-building flash crowds saw greater benefits from cooperative caching.

4. ELASTIC RESOURCE ALLOCATION

During flash crowds, services need to allocate additional resources to meet surging HTTP traffic. The nature of such allocation depends on whether the crowd is composed of static or dynamic content.

Crowds for a small number of unique URLs are easily served from cache. As a result, the primary resource allocation challenge falls to the CDN, which must direct clients to its wide-area proxies handling the requested domain (typically during DNS resolution). CDNs may also (re)allocate additional cache resources to particular crowds or popular content; to do so, they must observe that existing proxies are getting overloaded [14].

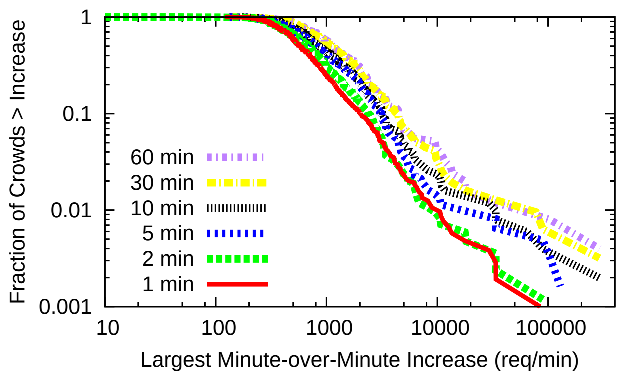

Figure 6: Complementary CDF of the largest request rate increase between successive minutes, for sets of crowds defined by different epoch lengths.

The situation is more complex when crowds are for dynamic or otherwise uncacheable content. In the face of flash crowds, poorly provisioned services can use fall-back mechanisms to reduce server functionality (for instance, disabling php execution on a web server). This approach reduces the computational load per request, but at the cost of losing personalization, analytics, or other beneficial services. It also fails to guarantee availability, since servers may be overwhelmed even under reduced functionality.

A second option is to push the problem of scaling computation (rather than the normal communication) to the CDN. This approach is supported by a set of "edge computing" APIs, which allow partial distribution of application logic to remote cache sites [3]. Unfortunately, not only does such a solution constrain application developers to a particular set of (non-standardized) service APIs, but it also fails to address the common situation where application logic requires back-end communication with a coordinating, more centralized database (which also must scale rapidly during crowds).

Only recently has a general-purpose solution arisen to the problem of scaling a dynamic application. The introduction of elastic computing platforms, such as Amazon's EC2, enables web services to spin-up new resources on demand in multiple global datacenters. Simultaneously, a litany of scalable back-end datastores—such as MongoDB, Cassandra, and HBase—scale quickly with the addition of new hardware resources (which also can be allocated in a virtualized environment).

Whether the resource allocation is delegated to a CDN or performed in-house using elastic-computing resources, the responsible service must have adequate time to react: Can such resource allocation be performed fast enough in the face of flash crowds? This section analyzes the rate of flash crowds and evaluates the efficacy of dynamic resource allocation.

4.1 Measuring Rate Increases

Figure 6 presents the complementary CDF of the maximal minute-over-minute increase in request rate observed during each crowd. A separate CCDF is plotted for each epoch time.

We observe that a handful of crowds experience extremely large surges in request rates, increasing by tens of thousand of requests in successive minutes. Further investigation determined that such volatility is caused by a particular usage scenario: such sites use HTTP 302 redirects to relay traffic to CoralCDN only during extremely loaded periods. Such elastic load shedding presents an

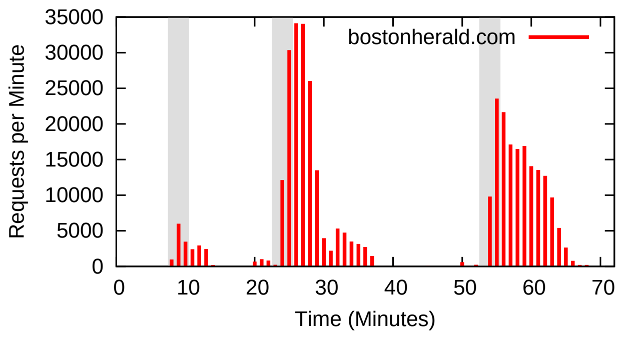

Figure 7: Surging request rates caused by intermittent 302 redirection from *The Boston Herald*.

attractive proposition for resource-constrained websites: the ability to use third-party content distribution only when local resources are oversubscribed. However, it also eliminates the traditional request build-up that can serve as an "early warning sign" for CDNs.

As an example of such rapid increases, Figure 7 plots the number of requests per minute to CoralCDN during a flash crowd experienced by *The Boston Herald* over roughly one hour. The three distinct traffic surges (grey background regions) occur just after the website switches on redirection, likely in response to a local load-shedding policy. As a result, CoralCDN sees thrashing and unpredictable request rates.

While instantaneous flash crowds to CoralCDN are still a small fraction of its total use, they still point to the important concept of resource elasticity. Under such scenarios, it can be difficult for CDNs and origins to respond quickly to dramatic load increases.

4.2 Effectiveness of Elastic Scaling

Crowds for static or dynamic content require distinct resource allocation mechanisms. Using cloud-computing platforms such as Amazon EC2, origin sites can "spin up" new virtual nodes to handle increased demand for dynamic or uncacheable content. Alternatively, CDNs can satisfy request spikes for static content by adapting the set of proxies handling a particular domain, possibly within seconds. In either case, the service must measure resource utilization and determine when to allocate new resources to address flash crowds.

To examine whether either environment can adapt rapidly enough, we simulated dynamic resource allocation strategies against the crowd traces. In our experiments, we assume that each proxy or compute node can handle a fixed request rate. We further consider a desired utilization ratio across all nodes. Production services often target an average utilization ratio that is well below 100% to ensure that spare capacity exists during unpredicted fluctuations in demand. Lower utilization provides a better hedge against rate increases, at the price of higher operational cost.

In our simulation, a service begins with a sufficient number of nodes to handle 5% of the crowd's peak load at the desired utilization. After each 10-second interval, during which time the request rate can increase, the service initializes zero or more new nodes to bring its utilization below the desired target. There is a delay of d seconds before a new node can service requests (d = 10, 60, 600). In this experiment, a service never decreases its number of active nodes.

Our main considerations are how often this simple strategy leaves the service oversubscribed (that is, more than 100% utilized) and how sensitive any oversubscription is to the speed of resource allocation.

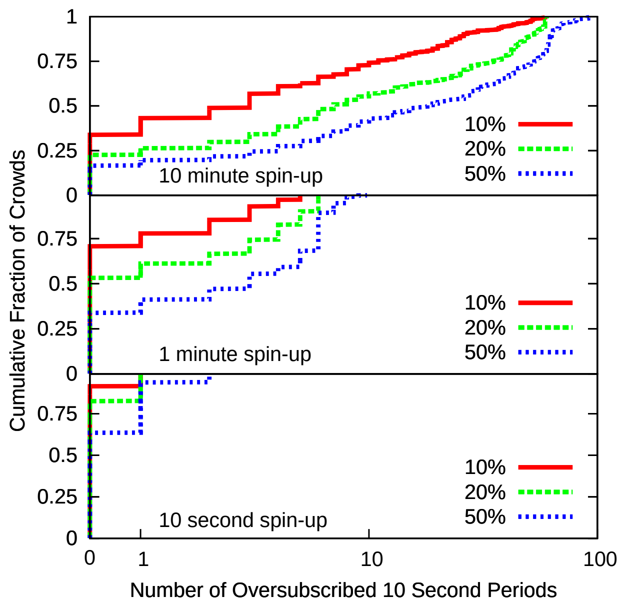

Figure 8: CDFs of the number of oversubscribed (>100% utilization) periods when resources are dynamically scaled during flash crowds. Plots differ by the spin-up delay *d* needed to bring resources online, each evaluating several target utilizations.

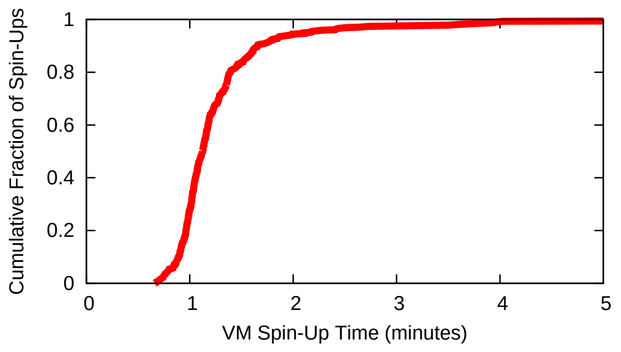

Figure 9: CDF of spin-up times for EC2 virtual instances.

Figure 8 plots a CDF of the number of oversubscribed 10-second intervals during each experiment (for each of the 2,501 crowds) for various target utilizations (10%, 20%, and 50%). We find that sub-minute resource allocation is required to prevent oversubscription in most flash crowds. For instance, if allocating new VMs requires ten minutes, 66% of all crowds experience at least one period of oversubscription, even under the most conservative target of 10% utilization. While this experiment is highly simplified, it motivates the need for further study on the effectiveness of resource allocation policies in the face of load volatility.

To characterize the delay for allocating new resources in the cloud, we profiled Amazon EC2. Our experiment measured the time required to spin-up a new VM, repeating the process in three different datacenter locations each hour over the course of one day (August 31, 2011). Figure 9 plots the CDF of spin-up times observed on EC2: The majority of instances became available within 1-2 minutes. Given these delays, avoiding dropped requests for the majority of observed crowds requires the most aggressive provisioning strategy.

Referer	# Crowds	Referer	# Crowds
digg.com	123	facebook.com	10
reddit.com	20	duggmirror.com	8
stumbleupon.com	15	duggback.com	5
google.com	11	twitter.com	4

Figure 10: Domains that are common referrers for crowds.

5. SOURCE ATTRIBUTION

While flash crowds may seem unpredictable and random to service providers, they are often explained by human factors. The "Slashdot Effect," for instance, refers to the spike in popularity witnessed by sites linked to by the Slashdot website. Increasingly, social networks are facilitating the *viral* sharing of links between acquaintances, which can lead to explosive growth in content popularity. This section qualifies the role of third-party portals in creating flash crowds.

5.1 Causal Inference for Source Attribution

We determine the cause of flash crowds through HTTP Referer headers. The Referer header, when present, indicates that a user has arrived at a particular URL by means of a link from another URL, or that the URL represents content embedded in a parent webpage. In many cases, a complex web of referrers arises when a user accesses a web page, fetches embedded content, and then further browses the trending website. We use the set of pair-wise referral links to reverse-engineer a user's HTTP session (with referral links that form a directed acyclic graph). This allows us to attribute their subsequent traffic to as small a set of originating referrers as possible.

5.2 Effects of Portals, Networks, and Tweets

Figure 10 lists referring domains that contributed heavily and frequently to flash crowds. We consider a domain as a heavy referrer if it originates more than a third of a crowd's requests. Well-known Internet portals, (in)famous for causing flash crowds, top this list. These findings confirm the perception that online portals are a common source for flash crowds. The term "Slashdot Effect" may be a slight misnomer, however: While Coralized URLs were posted on the Slashdot homepage several times during the trace, very few generated sufficient traffic to meet our crowd threshold. Still, these findings are predisposed by user behavior: CoralCDN appears to be more commonly used by digg readers than those of Slashdot.

Perhaps more surprising is the presence of crowds referred from Facebook and other social networking sites. Unlike centralized content portals, these sites disseminate links through social connections, spreading virally via pairwise, decentralized exchange. A link posted on Twitter is shared with direct followers, then re-tweeted and propagated through the social graph, for instance. Recent work [11] has considered the role of user location in predictively caching content for such crowds. We confirm that project's hypothesis that social-network activity can generate meaningful flash crowds.

A handful of crowds arose with even less explicit coordination: those whose primary referrer was search traffic from sites such as Google, Yahoo!, and Bing. These crowds relied on no central portal or coordinating website, instead growing through some type of out-of-band communication, *e.g.*, e-mail, word of mouth, etc. Still, through the "zeitgeist," search engines generated exponential traffic increases to particular websites, and ultimately to CoralCDN.

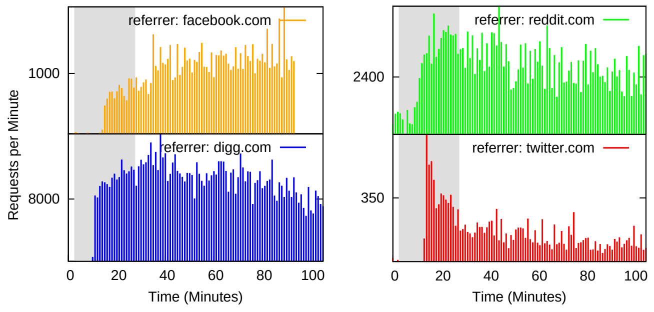

Figure 11: Referral traces from third parties. Each plots a distinct crowd, but timing of each surge (grey regions) are aligned.

To illustrate the traffic patterns from a few example crowds, Figure 11 plots the requests rates from a variety of referring websites, including social media sites and content portals.

6. RELATED WORK

Several research projects have focused on mitigating the effects of flash crowds. A number of papers explore the use of peer-to-peer protocols to cooperatively address traffic surges [5, 10, 12, 13]. Work on request redirection in traditional CDNs has also considered flash crowds, reallocating demand among back-end proxies during periods of high load [14]. Still other work has focused on detecting and pruning flash crowds within local networks, before traffic ever reaches a bombarded service [1, 8]. None of these studies consider traces or offer analysis of real flash crowds observed across the wide area. A notable exception is Jung *et al.* [6], which draws conclusions from a small number of flash crowds against a single web server. That study focuses on distinguishing flash crowd traffic from denial-of-service attacks, a topic we only briefly address in this paper. Very recent work has considered using social network information, such as user location, to determine cache policy during flash crowds [11]. That work restricts its analysis to user-level observations of twitter crowds, however, and it lacks information about the true network location of users and caches. Our analysis is complementary, profiling the actual distribution of users amongst caches in an operational CDN and characterizing cache locality during real crowds.

7. CONCLUSIONS

This paper finds that flash crowds, even when conservatively defined, are quite commonplace in a popular, open CDN. We initiate the study of flash-crowd dynamics using logs of several thousand such crowds. Our measurements show that crowds vary in their amenability to effective caching, and that cooperation between caching proxies minimizes origin load for some crowds, but not all. Further, we find that some crowds can grow too quickly for dynamic resource allocation to keep up with demand; cloud-based approaches of spinning up new VM instances can sometimes fare poorly in preventing oversubscription. Finally, we confirm anecdotal evidence that third-party portals contribute to substantial numbers of crowds, and document that the sharing of links through social media and the uncoordinated use of search engines lead to flash crowd formation as well.

Acknowledgments. We thank Stefan Saroiu and Kay Ousterhout for their helpful comments about this work. This work was supported by the National Science Foundation under Award CNS-0904860.

REFERENCES

[1] P. Barford and D. Plonka. Characteristics of network traffic flow anomalies. In *Proc. Internet Measurement Workshop (IMW)*, Nov. 2001.

[2] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell. A hierarchical Internet object cache. In *Proc. USENIX Annual Technical Conference*, Jan. 1996.

[3] A. Davis, J. Parikh, and W. E. Weihl. EdgeComputing: Extending enterprise applications to the edge of the Internet. In *Proc. World Wide Web Conference (WWW)*, May 2004.

[4] M. J. Freedman. Experiences with CoralCDN: A five-year operational view. In *Proc. Networked Systems Design and Implementation (NSDI)*, Apr. 2010.

[5] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing content publication with Coral. In *Proc. Networked Systems Design and Implementation (NSDI)*, Mar. 2004.

[6] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and Denial of Service attacks: Characterization and implications for CDNs and web sites. In *Proc. World Wide Web Conference (WWW)*, May 2002.

[7] B. Maggs. Personal communication, 2009.

[8] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *Computer Communications Review*, 32(3), July 2002.

[9] E. Nygren, R. K. Sitaraman, and J. Sun. The Akamai network: A platform for high-performance Internet applications. *SIGOPS Operating Systems Review*, 44, Aug. 2010.

[10] V. N. Padmanabhan and K. Sripanidkulchai. The case for cooperative networking. In *Proc. Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, Mar. 2002.

[11] S. Scellato, C. Mascolo, M. Musolesi, and J. Crowcroft. Track globally, deliver locally: Improving content delivery networks by tracking geographic social cascades. In *Proc. World Wide Web Conference (WWW)*, Mar. 2011.

[12] S. Shakkottai and R. Johari. Demand-aware content distribution on the Internet. *IEEE/ACM Trans. Networking*, 18(2), Apr. 2010.

[13] T. Stading, P. Maniatis, and M. Baker. Peer-to-peer caching schemes to address flash crowds. In *Proc. Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, Mar. 2002.

[14] L. Wang, V. Pai, and L. Peterson. The effectiveness of request redirection on CDN robustness. *SIGOPS Operating Systems Review*, 36, Dec. 2002.