Using Mini-Flash Crowds to Infer Resource Constraints in Remote Web Servers

Pratap Ramamurthy[⋆], Vyas Sekar[†], Aditya Akella[⋆], Balachander Krishnamurthy[⋆⋆], Anees Shaikh[††]
[⋆]University of Wisconsin-Madison , [†]Carnegie Mellon University
[⋆⋆]AT & T Labs–Research , [††]IBM Research

ABSTRACT

Unexpected surges in request traffic (e.g., flash crowds) can exercise server-side resources such as access bandwidth, CPU, and disks in unanticipated ways. Administrators today do not have the requisite tools to fully understand the effect that flash crowds can have on server-side resources. As a result, most Web-servers today rely on significant over-provisioning, strict admission control, or alternatively use potentially expensive solutions like CDNs to provide high availability under load. A more fine-grained understanding of the performance of servers under emulated but controlled flash crowd like conditions can guide administrators to make more efficient provisioning and resource management decisions.

We present the initial design of Mini-Flash Crowds (MFC) – a light-weight wide-area profiling service that reveals resource bottlenecks in a Web-server infrastructure, including access bandwidth, processing resources, and back-end data management. The MFC approach is based on a set of controlled measurements in which an increasing number of distributed clients make synchronized requests to exercise specific resources of a remote server. Using a number of wide-area experiments and controlled lab tests, we show that our approach can faithfully track the impact of request loads on different server resources. Our approach is non-intrusive and thus we can use it to actively probe numerous live Web servers. We present the results from a preliminary measurement study of resource provisioning on public Web servers.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: [Measurement techniques, Modeling techniques, Reliability, Availability, and Serviceability]

General Terms

Management, Measurement, Performance

Keywords

Flash crowds, Web performance

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

INM'07, August 27–31, 2007, Kyoto, Japan.

Copyright 2007 ACM 978-1-59593-788-9/07/0008 ...$5.00.

1. INTRODUCTION

Providers of Web-based applications have to provision resources (processing and memory capacity, access bandwidth, database and storage) to ensure good response time. Providing this consistently under a broad range of operating conditions requires an ability to predict the volume and mix of requests. Large providers typically resort to over-provisioning, distributed content delivery, or dynamic resource provisioning. Smaller providers provision for the common case by trading off robustness to large variations in workload for a cheaper infrastructure. Yet, operators are left without a sense of how their application infrastructure will handle large increases in traffic due to planned events or unexpected surges due to attacks or flash crowds. Such events can lead to significant loss of revenue and dissatisfied users if the infrastructure is unable to maintain reasonably good service or at least degrade gracefully.

We introduce a profiling service to help operators better understand the ability of their Internet applications to withstand increased request load. We propose *Mini-Flash Crowds (MFC)* – a mechanism that reveals constraints in the application infrastructure by quantifying the number and type of simultaneous requests that are likely to tax specific resources and affect response time. Using MFC, an application provider can compare the impact of an increase in database-intensive requests versus an increase in bandwidth-intensive requests. Armed with other information (say, through controlled lab testing), the operator can make better decisions on provisioning additional resources or introducing request shaping.

A MFC is a phased set of controlled measurements in which an increasing number of distributed clients make synchronized requests to a remote application server. These requests attempt to exercise a particular part of the infrastructure such as network bandwidth, local disk, or back-end database. As the number of synchronized clients increases, one or more resources may become stressed, leading to a small but *discernible* rise in the response time. At this point, inferences can be made about the resource profile of the application. The number of clients making simultaneous requests is increased only up to a pre-set maximum – if there is no change in the response time, we label the application infrastructure as unconstrained. This somewhat conservative approach allows MFC to reveal resource limitations for many applications while limiting its intrusiveness on the tested sites.

The MFC technique can be thought of as a "black-box" approach for determining the resource limitations of an Internet application, or for uncovering performance glitches, vulnerabilities, and configuration errors that were not apparent from internal testing in a lab or data center. The key advantages of the MFC approach are: i) lightweight and non-intrusive measurements that have minimal impact on, and involvement from, production servers; ii) use of real, distributed clients that test the deployed application infrastructure, and accurately reflect the effects of wide-area network conditions; and iii) ability to work with a broad range of Web applications with little or no modification, while at the same time providing some tunability to run more application-specific tests.

We conducted many wide-area and controlled lab experiments using synthetic workloads to validate MFC's ability to track server response times and reveal resource constraints. We also conducted preliminary experiments over live Web servers and known Phishing servers. Early results indicate that the MFC approach is an important contribution to the set of tools and services for profiling the performance and availability of Web servers.

2. MINI-FLASH CROWD DESIGN

2.1 Solution Requirements

Our goal is to develop a mechanism that gives application providers useful information about resource limitations without imposing undue load on their infrastructure. From the operator's point of view, the mechanism should accurately reflect the application's performance under realistic load conditions; i.e., the information should be *representative*. Traditional benchmarking approaches in controlled LAN settings (e.g., [11]) do not account for wide-area conditions or characteristics of the actual Internet connectivity (e.g., speed, diversity, etc.). Thus, we need to stress the actual application infrastructure using clients distributed around the Internet. Secondly, the approach should be largely automatic, requiring minimal input from operators about the specifics of the application and infrastructure. Finally, allowing some tunability is desirable for testing specific applications, and for tailoring the measurement to different operational goals. Some applications may be tolerant to large increases in response time (e.g., software binary distribution services), while others may be more sensitive to increases in response time (e.g., it may be important to know that a 10% increase in request volume causes the response time to double).

2.2 MFC methodology

The above requirements raise several technical challenges. Developing a generic request workload that can exercise a specific combination of resources on a production server infrastructure is difficult. Given an observed increase in response time, it is nontrivial to attribute it to contributions from individual server resources. For example, it may be very difficult to attribute a 50ms increase in response time to a bandwidth bottleneck at the server versus a limitation of the server processing capacity. Another challenge is the problem of exercising tight control on the load imposed on the application infrastructure from a set of distributed clients whose requests may be affected unpredictably by wide-area conditions.

To exercise specific server resources, we issue concurrent requests from a number of clients for a particular type of content. For example, to exercise a server's network connection, we can make concurrent requests for large objects hosted at the server (e.g., binary executables, movie files). To minimize the need for server-side input and to maintain a level of generality in the approach, we survey the content hosted on the server and automatically classify it into different types using heuristics such as file name extensions and file sizes. We achieve tight control over the load on the server resources by scheduling client requests in a centralized manner using measurements of the network delay between each client and the target server.

Figure 1 shows a single *coordinator* orchestrating the MFC experiment on a *target server*. At the coordinator's command, a specified number of *participating clients* send synchronized requests to the target server. A set of *measurers* monitor the progress of the MFC by sampling the response time at the target and communicating with the coordinator. The coordinator uses the feedback from clients and measurers to determine how long to run the MFC. The overall experiment consists of a profiling step, followed by several measurement phases.

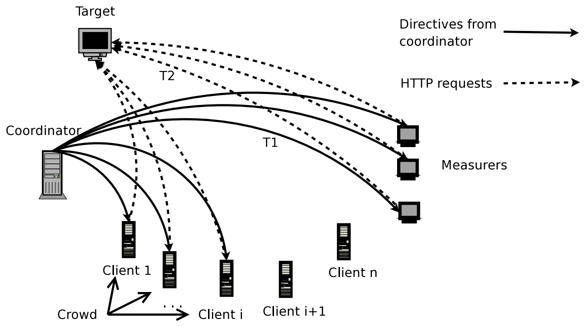


Figure 1: Structure of a Mini-Flash Crowd experiment.

Profiling target content. Before the MFC experiment begins, the profiling step involves the coordinator crawling the target site. The coordinator classifies the objects discovered during the crawl into a number of categories based on content-type, for example, regular/text (for text and HTML files), binaries (e.g, .pdf, .exe, .tar.gz files), images (e.g., .gif, .jpg files), and queries (e.g., form elements). Objects are further classified into three categories based on the reported object sizes (issuing a HEAD request for files and GET request for queries) – *Small Objects*, *Large Objects* and *Small Queries*. The Small Objects and Large Objects groups contain regular files, binaries, and images whose sizes are less than 10KB or greater than 100KB, respectively. We also note the URLs which appear to generate dynamic responses (e.g., by executing scripts), and sort them by the size of the returned data. The Small Queries category is populated with the smallest 10th percentile from this set. Note that a small query is not necessarily a "form", but a predefined URL with a "?" which indicates a CGI script. The categories of object requests are selected for their expected impact on specific server resources (refer Section 3).

MFC Stages. After completing the profiling step, the MFC experiment proceeds in stages. In each stage, the MFC attempts to exercise a specific set of server resources by making a varying number of synchronized requests for objects from the same category.

In the *Small Object stage*, participating clients request *unique* small objects (if available). Assuming the same objects are not requested simultaneously by other clients (and hence cached in the server's file system), we expect this stage to primarily impact the disk sub-system. In the *Large Object stage*, participating clients request the *same* large object simultaneously, primarily exercising the server's network access bandwidth. Since we request the same object multiple times, the likely caching reduces risk of impacting of the disk. In the *Small Query stage*, clients make requests for unique dynamically generated data if available; else the same object is requested. Since such queries often require some interaction with a backend database, we expect that this stage will affect the back-end information system, irrespective of whether the clients request the same or different small queries[1]. Finally, in the *Base stage*, clients make a HEAD request for the base Web page hosted on the target servers (e.g. index.html or index.php). This stage provides an

[1] The load on the database backend is likely to be lighter in the latter case due to caching of results in the DB front-end.

estimate of the basic request processing time at the server.

Within each stage, we assume that the *load* on the server's resources is strictly monotonic in the number of simultaneous requests. However, there may not be a discernible load on server resources for a number of reasons: (1) the server caches objects and other clients (not part of the MFC) request the same object synchronously, or (2) the application infrastructure consists of multiple replicated servers, or (3) the server dynamically provisions additional resources to handle increases in request volume. Without knowledge of the server side configuration, it is difficult to determine if caching, load balancing, or dynamic provisioning affects a particular stage of the MFC experiment. Thus, if response time does not grow with the number of participating clients in a stage, we cannot infer the effect of the MFC on server resources. The lack of a perceptible degradation in performance does imply that the server is well-equipped to handle the requests. If the response time does increase monotonically, then we can assume that server-side resources are indeed being exercised.

Epochs. Each stage of a MFC experiment is composed of several *Epochs*. In epoch i, the coordinator directs N_i participating clients to issue concurrent requests of a given type to the target. Clients participating in epoch i constitute a *crowd*. The coordinator also determines the particular object that a client or measurer should request in an epoch.

During an epoch, the measurers closely monitor the target server to ascertain if the MFC is causing response times to increase appreciably. The measurers make concurrent requests for either the same type of object as the crowd or other types of content. The latter approach is useful for quantifying the correlation among resources on the target server (e.g., "How does a disk-intensive workload impact the response time of a dynamic or DB-intensive request?"). The measurers must access the target site at the same time as the crowd to track the impact of the MFC on the target accurately. At the end of the epochs, the measurers report the normalized response time (*response time for request − RTT between measurer and target*) to the coordinator.

Based on the measurers' response times for requests in epochs $1..i$, the coordinator either terminates the stage, or moves to epoch $i+1$. The coordinator uses the following simple algorithm to decide the next step:

Check: If the median normalized response time reported by the measurers in an epoch i ($i \geq 3$) is greater than a threshold θ, the MFC enters a "check" phase. The goal of this phase is to ascertain that the observed degradation in response time is due to overload on a server resource and not due to noise in measurement. The coordinator creates two additional epochs, one numbered "i-" with $N_i - \delta$ clients, and the other numbered "i+" with $N_i + \delta$ clients. If the measurers' median response time in these epochs is also above θ, the check succeeds and the coordinator **terminates** the MFC experiment. Otherwise, the check fails and the MFC **progress**es to epoch $i + 1$.

We note that the variation in the measurers' response time indicates the confidence we can attach to the measurement: the larger the variation, the higher the likelihood that the measurements were affected by noise.

Progress: If the measurers report no perceptible increase in the target's response time in a regular epoch, or the check phase fails, the coordinator progresses to the next epoch where a larger number of clients participate. To ensure that the target does not face sudden load surges, the coordinator increases the size of the crowd by a small, fixed value (we set this to 5 in our experiments).

Terminate: If the check phase succeeds, or the number of participating clients exceeds a certain threshold, the coordinator terminates the experiment and resets all clients and measurers.

Synchronization. In any given epoch, the load on the target server is proportional to the number of *concurrent* requests it is serving, which directly determines the server's response time. Thus, an important requirement is that when k clients participate in an epoch, the number of concurrent MFC requests at the server is $\approx k$. Implementing a distributed synchronization protocol among the clients introduces unwanted complexity; we rely on techniques that achieve reasonable synchronization in practice. Specifically, we exploit the centralized coordinator to schedule client requests in a synchronous way.

To ensure synchronization, the coordinator issues a command to the clients (and the measurers) at the beginning of the experiment to measure the round-trip latency to the target server[2]. Client i then reports the round-trip delay, T_i^{target}, back to the coordinator. The coordinator also measures the round-trip delay from itself to each client, T_i^{coord}.

Based on these measurements, the coordinator schedules client requests so that they all arrive at the server at roughly the same time T. Note that the actual HTTP request arrives at the server roughly at the same time as the *completion* of the 3-way SYN hand-shake. To synchronize client request arrivals, the coordinator issues a command to client i at time $T - 0.5 * T_i^{coord} - 1.5 * T_i^{target}$. Assuming that the network latency between the coordinator and the clients has not changed since the initial latency measurements, client i will receive this command at time $T - 1.5 * T_i^{target}$; at this time, client i issues the request specified in the command by initiating a TCP hand-shake with the server. Again, assuming client-target network latency does not change, the first byte of client i's HTTP request will arrive at the target at time T.

Since the time-span of a MFC experiment is less than a few minutes, we believe that the assumption of stationary network latencies is not unreasonable [12].

3. VALIDATION

We address the following questions through validation experiments performed in controlled laboratory settings: (1) Are requests from MFC clients adequately synchronous? (2) How well can MFC track response time of the server? (3) How effective are MFC requests at exercising the intended resources at the target server?

3.1 Validating Synchronization and Response Time Tracking

To answer the first two questions, we set up a server that does not host any real content and just returns a blank page. The server runs a lightweight HTTP server [1] on Linux 2.6.9 on a 3.2 GHz Pentium-4 machine with 1GB of RAM. The MFC software was deployed on 70 PlanetLab machines (both clients and measurers). Both the coordinator and the target web server are high-end machines within University of Wisconsin, with high-bandwidth connectivity to the Internet. The target server receives no other request traffic. We instrument the server to track request arrival times and to implement synthetic response time models.

Synchronization. To test the accuracy of the synchronization, we log the arrival times of each incoming HTTP request at the target server. Figure 2 shows the arrival time of each request with a crowd size of 45 clients. In these experiments, the coordinator commands the clients to make a HTTP request roughly $d = 15$s after taking the latency measurements. About 70% of the requests arrive within

[2] To do this, we use TCP pings, i.e., we measure the delay between the transmission of a SYN packet and the reception of a SYN+ACK.

5ms of each other (clients 7 through 40), and 90% of the requests arrive within 30ms of each other (clients 3 through 43), indicating that our synchronization algorithm works reasonably well.

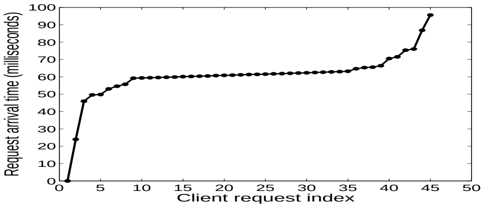
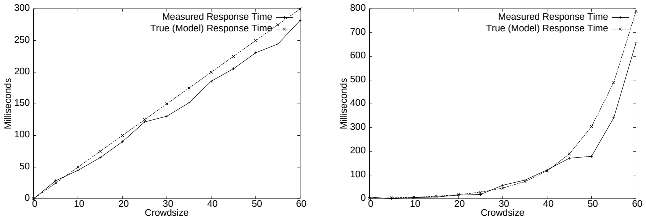


Figure 2: The arrival times at the target server for 45 participating clients



(a) Linear (b) Exponential

Figure 3: Emulation of response time functions.

Accurate Emulation of Response Time. Next, we incorporate artificial response time models into the validation server. Each model defines the average response time per incoming request as a function of the number of simultaneous requests at the server. The artificial response times were strictly non-decreasing functions of the pending request queue size.

We show the response times estimated by the measurers for two models: Linear (Figure 3(a)) and Exponential (Figure 3(b)). In both cases, we see that the measurers are able to faithfully track the server response time function.

3.2 Understanding Resource Constraints

We now examine the effectiveness of MFC in exercising specific resources at the target server. We set up a Apache 2.2 Web server (with the worker multi-processing module) on a 3 GHz Pentium-4 machine with 1GB RAM running Ubuntu Edgy (2.6.17.10 kernel)[3]. We emulate a MFC on this target server with clients located on the same LAN as the server.

For each experiment we measure (1) the response times seen by the measurers and MFC clients, and (2) server-side resource utilization using system diagnostic utilities (we use `atop` to monitor the CPU, resident memory, disk access, and network usage at the server). Unless otherwise specified we use crowd sizes in the range of 5-50 clients, in increments of 5.

Small object workload. In the small object workload, every client requests a 20 byte object from the target validation server. We consider two variants of this experiment: *unique* small objects and *same* small objects to distinguish caching effects at the server. Figure 4 shows the results of the same small object experiment. There is little perceivable load on the server and correspondingly no substantial increase in the observed response time. The absence of disk reads for the duration of the experiment indicates server-side caching. The CPU, network, and memory utilization increase very slightly with crowd size (not shown).

Next, we populate the server with a large number of unique 20 byte objects. Figure 5 shows that the observed response time when each client requests a unique object in each epoch. We see a 20× increase when compared to the same small object experiment for the corresponding crowd size. The network and memory utilization are comparable to the same small object experiment. Unlike the same objects experiment, the number of disk reads grows linearly with crowd size, accounting for the large increase in response time.

Large objects. In the large object workload, each MFC client requests the same 100KB object from the server. Figure 6 shows a significant increase in the median response time observed by the clients due to the network load on the server. CPU, memory, and disk utilization remain negligible during the experiment – the network bandwidth constraint is primarily responsible for the increase in response time.

Small dynamic object workload. For emulating a dynamic object or database query workload we set up a backend database (database details are not relevant for this validation experiment). The query of interest causes the server to retrieve 50000 entries in one of the database tables and return their mean and standard deviation. Note that the query workload is not network intensive as the responses are less than 100 bytes. The backend database is a MySQL server with the query cache size set to 16MB. Many database implementations proactively cache query responses to avoid additional query overhead. We experimented with two server-side software interfaces for the database backend: the `FastCGI` [8] module and `Mongrel` [4], a lightweight server explicitly designed for handling dynamic objects. FastCGI's inefficient implementation[4] caused memory usage on the server increased dramatically with crowd size; thus we focus on the results using Mongrel. For crowd sizes up to 50, we see that the response time stays within 10ms (not shown). The actual database backend appears to be efficient in caching query responses; we observed little effect on disk, CPU, and memory use.

We also examine a *unique* small dynamic object workload to identify resource bottlenecks in the absence of database query caching effects. Figure 7 shows our validation measurements; the most interesting feature is the median response time increasing due to the increased CPU utilization.

Implications of Validation Experiments. We see that by choosing appropriate workloads it is possible to narrow down the impact on specific server resources. This validates the MFC premise that it is feasible to provide useful inferences on resource constraints using a lightweight measurement infrastructure. We also came across two other interesting effects which helped us put the applicability of the MFC approach in perspective: i) impact on serial vs. parallel accessed resources, and ii) granularity of MFC inferences.

An increase in the observed response time in the MFC experiment under a particular type of request can be attributed to two possibilities. One is an increase in the load on a specific server resource, where each additional request consumes a proportional fraction of the resource. The other is an increase due to server-

[3] For the synchronization and response time evaluation we chose a simple server software that is easy to instrument with the functionality we desired. For these experiments we use a more realistic configuration

[4] FastCGI forks a new process for each request. As the number of requests increases, each of the forked process independently inherits the memory image of the parent process leading to very high memory usage during the experiment.

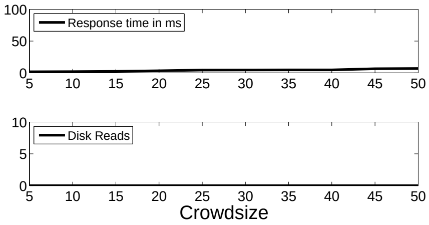


Figure 4: Same 20 byte small object

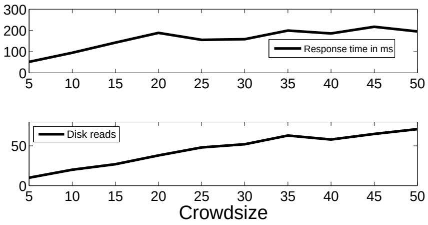


Figure 5: Unique 20 byte small object

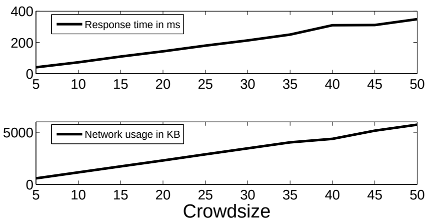


Figure 6: Same 100KB large object

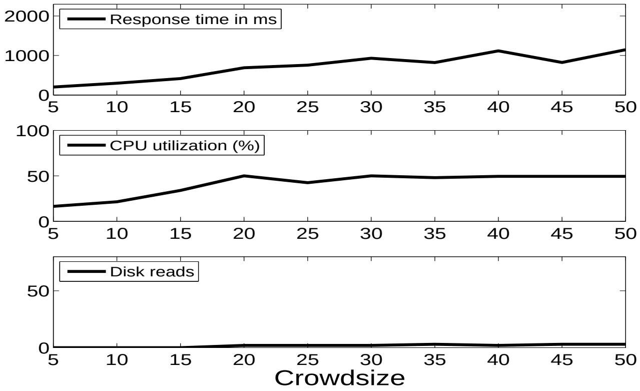


Figure 7: Unique small dynamic object request with Mongrel

side scheduling and resource serialization constraints, where additional requests do not impose any additional load on a resource, but merely create larger-sized queues of requests waiting for the resource (e.g., serialized access to a single disk). Thus, serialization bottlenecks can impact our ability to detect resource constraints.

Server throughput is determined by a number of factors, including hardware performance, software throughput, and the server-side components used for handling requests. Our experiments show that while we may be able to isolate resource constraints at a "subsystem" level (e.g., disk subsystem, database access subsystem, network subsystem), providing information at a finer granularity to precisely pinpoint if the constraint is a hardware or software inefficiency is difficult. Even with full access to the server and the ability to fully instrument it, performing root-cause analysis of performance degradations is still non-trivial given the complexity of modern day servers and applications. Naturally, inferences from remote measurements alone are more challenging.

Ultimately any fine-grained analysis of resource constraints are best understood by the individual administrators managing the target server or applications being tested. We sought to indicate coarse-grained resource constraints as a guideline for better server provisioning. Our validation results demonstrate the promise of the MFC approach to achieve this goal.

4. PRELIMINARY MEASUREMENTS

We present two sets of preliminary wide-area measurements using the MFC approach. Over the course of one week, we ran the "Base" stage of the MFC (HTTP HEAD requests to index.html) against 200 live Web servers and 44 known phishing servers, using 65 Planetlab machines as clients. We use a threshold $\theta = 100$ms, i.e., if the difference between the observed response time (median across measurers[5]) and the base response time (corresponding to a crowd size of 5) is more than 100ms we *stop* the MFC experiment.

Generic Web servers: For purposes of illustration, we group different Websites into four non-overlapping categories based on the Alexa "reach per million" metric [2][6]: 50 Web sites each with reach in the range 1-10, 10-100, 100-1000, and 1000-10000. We expect that Websites with small reach would be qualitatively similar to one another in terms of provisioning, and different from Websites with much larger reach.

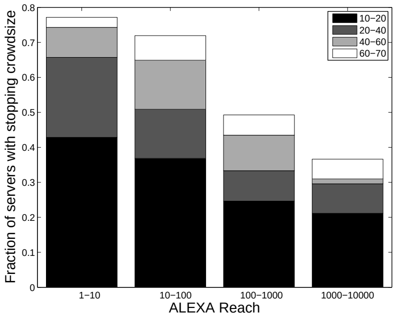


Figure 8: Breakdown of stopping crowd sizes for various Alexa reach ranges with a HEAD request

Figure 8 shows a summary of the crowd size at which different Web servers show a degradation of more than 100ms in response time. For each reach category, we break down the crowd size values into sub-ranges as shown. We observe that, as expected, the fraction of Webservers that show a 100ms degradation in response time decreases significantly as we move to larger reach categories (95% for the 1-10 category vs 35% for 1000-10000). However, we find it surprising that more than 30% of the Websites even in the 1000-10000 category show visible degradation in response times even with fewer than 65 simultaneous requests.

Phishing Sites: We also conducted a preliminary measurement study of a number of phishing sites obtained from Phish-tank [5]. Our goal was to understand how these these servers compare against popular or low-end Web servers in terms of their provisioning. We tested 44 different phishing sites, of which 40 showed a 100ms response time increase with a crowd size less than 45. Figure 9 shows the breakdown of the crowd sizes for the 40 clients that showed such an increase. More than 60% of the phishing sites showed such an increase with a crowd size less than 15. If we compare this to the Alexa 1-10 category Websites from Figure 8, we find that the

[5] In our experiments, we treat each client as a measurer as well, i.e., we do not use a separate set of measurers.

[6] If a Web site has a reach per million of 1000, it means that the Web site was visited by 1000 people in Alexa's sample of 1 million users.

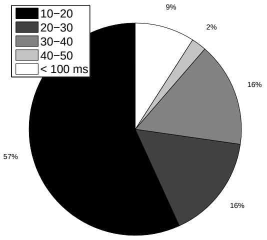


Figure 9: Breakdown of stopping crowd sizes with HEAD request for phishing sites

corresponding fraction of sites was 40%, suggesting, again as expected, that most of the phishing sites are hosted on fairly low-end servers similar to the lower-reach Websites.

5. DISCUSSION

Differences with real flash-crowds. One of the key emphasis of our work is that we use a *mini*- flash crowd that differs significantly from a real flash crowd. The key difference is that we run the measurements under a controlled setup. Thus, the requests from the MFC experiment appear like normal requests, except that they have ability to measure performance tipping points. We explicitly ensure that the target does not face a sudden surge in load unlike a real flash crowd. This implies that the target Website does not enter a panic mode of operation and thus skew the experiment results.

Multi-server Websites. Our current infrastructure assumes that a single IP address corresponds to a single physical host and the response times are measured under this assumption. This assumption does not hold for Websites that use certain load balancing techniques (e.g., using DNS redirects or IP anycast) in the server backend, or use a distributed deployment of servers (e.g., CDNs). The MFC approach can still be used to identify how such servers react to realistic flash-crowd scenarios; however identifying specific hardware bottlenecks becomes harder when the Websites use reactive load balancing techniques.

Security Implications. The presence of a public MFC infrastructure raises a concern that the approach can be exploited to launch targeted attacks with maliciously crafted request patterns to servers. However, we note that the MFC service will be applied in a restricted manner with access controls and the experimental parameters chosen to ensure the non-intrusiveness of the experiments.

Implementation inefficiencies vs. Performance prediction. At a high-level, we would like the MFC setup to achieve two goals. First, we want to *identify* (to some approximation) implementation inefficiencies and resource bottlenecks in a Web server infrastructure. Second, we want to provide a framework for Web site administrators to *predict* the performance under heavy load. While our initial design, measurement and validation experiments confirm the former goal, the latter goal requires that the setup be able to provide a full load-response curve. We plan to explore performance prediction in future work.

Implications for Administrators. Ultimately, the MFC measurements have to be translated into meaningful suggestions for network operators. For example, if a few small queries cause problems then they need to fix query lookups; if large objects are causing early keel-over they need to either break them up or pre-cache etc. We are currently working on providing a systematic framework for operators to debug Website performance.

6. RELATED WORK

Web server benchmarking tools [6, 11, 3, 10] range in sophistication. Some emulate multiple user sessions, create client requests for dynamic content and emulate several "standardized" workloads for e-commerce and regular Web browsing scenarios. Controlled emulation is done in a laboratory setting using average inter-arrival times, number of clients, and rate of requests derived from measurements taken at real Web servers. Benchmarks are used to stress test a server by running multiple clients in parallel where both clients and server are on the same LAN. MFCs use clients across the wide-area Internet generating more representative results. MFC's targeted requests exercise specific server resources in a controlled way yielding detailed and fine-grained observations.

Early work on flash crowds [9] proposed server-side heuristics to distinguish flash crowds and DDoS attacks. Chen et. al identify flash crowds by examining performance degradation in responses from a Website [7]. These techniques are useful to identify when a server is experiencing extreme load but MFCs accurately pin-point the request volume at which the server's response time begins to show perceptible degradation.

7. SUMMARY

We presented the initial design of MFC – a light-weight wide-area profiling service that reveals resource bottlenecks in a Web-server infrastructure. Using controlled measurements with an increasing number of distributed clients making synchronized requests to exercise specific resources of a remote server, we are able to faithfully track the impact on different server resources. Through a number of wide-area experiments and controlled lab tests, we show that our approach is non-intrusive and usable for actively probing live servers. We presented the results from a preliminary measurement study of resource provisioning on live public Web servers and known phishing servers. We are currently running extensive experiments with other MFC stages over a much larger set of Web servers. We are also running more targeted measurements at interesting special categories of Websites such as phishers, blogs, typosquatters, and startup company Websites.

8. REFERENCES

[1] Anti-Web HTTPD Homepage. http://www.hcsw.org/awhttpd/.

[2] Alexa. http://www.alexa.com.

[3] Mindcraft Benchmarks: WebStone. http://www.mindcraft.com/webstone/.

[4] Mongrel: V2. http://www.rubyonrails.org.

[5] Phishtank. http://phishtank.org.

[6] SPECweb2005 Benchmark. http://www.spec.org/web2005, 1999.

[7] X. Chen and J. Heidemann. Flash Crowd Mitigation via Adaptive Admission Control Based on Application Level Observations. *Transactions on Information Technology*, 5(3):532–569, 2005.

[8] P. Heinlein. Fastcgi. *Linux J.*, 1998(55es):1, 1998.

[9] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: Characterization, and implications for CDNs and web sites. In *World Wide Web*, May 2002.

[10] D. Mosberger and T. Jin. httperf: a tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev.*, 26(3):31–37, 1998.

[11] SPECweb99 Benchmark. http://www.spec.org/osg/web99, 1999.

[12] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker. On the Constancy of Internet Path Properties. In *First Internet Measurement Workshop*, Nov. 2001.