end of the test. Thus, we re-implemented the cache monitor using C, the libmemcached library [7], the UriParser [9] library, and straightforward C data types. The C implementation, by comparison, requires only 2.5 seconds to process the trace, and peaks at 39MB RAM usage for the 2 weeks worth of web server logs. With this implementation, the overhead from the cache monitor is negligible. The crawler cache script is implemented in php, and since it is not in the critical path, it is not optimized for performance.

Our use of the memcached server allows both the C cache monitor and php crawler cache implementations to communicate with the same server with no language translations. Memcached is not a requirement, however, as the cache could easily be implemented using another web object cache with appropriate language bindings, or even shared memory. On the other hand, memcached is scalable to multiple servers, available on a variety of platforms, and supports multiple language interface libraries.

4.1 Space Requirements

The test site in our case study consists of about 153,100 accessible thread pages, ranging in size from 50K to well over 250K for the HTML alone. After gzip-compression, the average page size is around 14K. Most (if not all) high-load crawlers accept gzip encoding, and as such, the cached pages are all stored in a pre-compressed state, rather than wasting cycles by compressing pages at the time of the request. As an added advantage, we found that the compression enables storage for 5-6 times more content in the L1 cache, dramatically improving hit ratios.

As each php process consumes anywhere between 30MB and 300MB of RAM, we deemed 120, 250, and 500MB to be reasonable sizes for the memcached server. Most importantly, the cache size should be set so as not to cause memcached to swap, which can produce very undesirable results (one order of magnitude slower processing time). Our chosen cache size is 500MB for our production server. With 12GB of RAM available, this amount is only 4.1% of the total primary memory resource. Recall that crawlers account for 33% of all server processing time, and this number is easily justified.

To store all of the static pages in the L2 cache, it requires 2.2 GB of storage in the file system for our test site. This number is small considering our test site produces nearly 4GB worth of access logs per week, and serves 15GB of image attachments. In order to prevent running out of inodes, the numeric ID of each page is used to determine a file's location in the file system modulo some maximum number of files. For instance, an ID of 20593 (mod 250) = 93 can be stored in the file system under the directory `/cache/9/3/20593.gz`, saving inodes in the process and providing some degree of organization. Of course, the crawler cache can easily interface with a lightweight database if desired.

5. EVALUATION

We evaluated the effectiveness of our approach using a live, busy web server. Our test site is a very busy online community with over 65,000 members, running the popular vBulletin [18] forum software. The site receives on average of 2 million hits per day, with around 900 users online continuously. Like many dynamic websites, the server hardware can be excessively taxed when crawled by multiple search engines. For instance, in January 2011, overload conditions were introduced to the server when Microsoft Bing 2.0 crawling agents were released to crawl the site. Yahoo's Slurp has been routinely blocked around the Internet in previous years for similar reasons. Excessive slowdowns appeared sporadically for nearly three weeks until the crawl subsided to a more manageable rate.

The two primary benefits of our approach from the perspective of performance are reduced response time and increased throughput. Under extremely heavy crawler load, targeting crawlers directly with caching is very effective at staving off overload conditions, enabling uninterrupted service to human users.

5.1 Experimental Setup

We developed a parallel tool written in php to test our cache setup, which takes as input the web server traces gathered over a two week window. The script is parameterized by the number of child threads to use, with each child thread responsible for a portion of the trace, and allows a test with nearly 90% CPU utilization on all cores. A snapshot of the online community database is configured and used along with the traces pulled from the web server logs. As a result, we created a live and accurate environment for experimental tests. We limited our trace to only dynamic requests for page listings. The trace includes $2,208,145$ bot requests and $1,052,332$ human requests for the `page.php` script, which is the primary workhorse for our case study.

5.2 Reduced Response Time

We first run the trace through the web server to gather baseline measurements, and we found the mean page generation time to be $162,109\mu s$, on order with our production server. This is the average latency experienced for all human users, as well as crawlers without the cache. With the addition of the crawler cache, we gained a reduction in latency of three orders of magnitude for crawler requests, to $1,367\mu s$. As an additional benefit of the crawler cache, the user-perceived response time for human users is reduced by 31%, to $111,722\mu s$. These results are summarized in Figure 8.

5.3 Increased Throughput

Achieving the reduced latency, we also observed a considerable increase in server throughput, as shown in Figure 9. The throughput for the traces without the crawler cache is 117 satisfied requests per second. With the addition of the crawler cache, throughput nearly doubles to 215 requests per second. With a higher throughput, overall aggregate response time for the entire trace is cut in half. Under periods of heavy load, this added capacity would be enough to stave off overload conditions caused by benign-yet-heavy crawlers.

5.4 Cache Generation

Ideally, most sites dependent on human activities (blogs, online communities, social networks) will have considerable dips in server load, providing an opportunity for L2 cache generation. In our case, based on diurnal usage patterns, we chose 4am to generate or update any new cache entries, though this parameter will depend heavily on the site's usage characteristics. To fully generate our cache from scratch on our test machine, we spent 1 hour and 22 minutes in using a parallel php script with 8 threads. This initialization need only be performed when the cache is deployed.

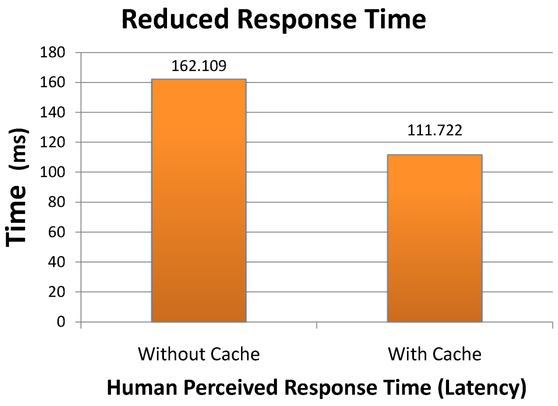

Figure 8: Crawler cache reduces server latency, improving user response time for human users.

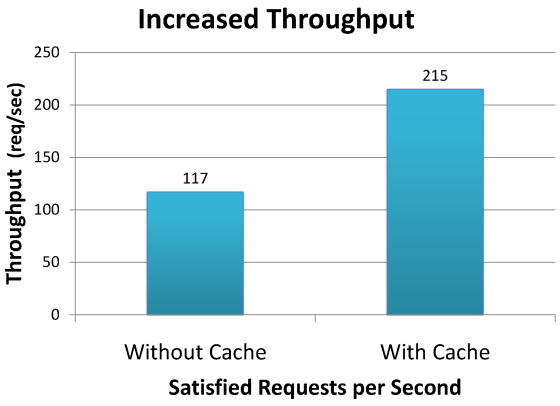

Figure 9: Crawler cache reduces server workload and as a result increases throughput.

Cache refreshing will also depend heavily on the site's database write characteristics. In our case, human users only modify around 200 unique threads per day on average (with multiple updates to those threads throughout the day). With only 200 modified pages to generate daily, this implies that the L2 cache can be refreshed in under 1 second. For sites like ours, the L2 cache can be updated multiple times per day, always providing fresh content to the crawlers as threads are updated.

5.5 Limits to Our Approach

For busy dynamic sites such as online communities, forums, and blogs, caching can be very beneficial to mitigate crawler overload risk. Unfortunately, for sites with an extremely high degree of transient data, such as event times and stock prices, static caching may not be the best approach. However, given the archival nature of the search engines, sites with a large amount of transient data are not well suited to crawling in general. These sites might better benefit from rewriting the URL to a static page explaining the possible benefits of visiting the page live.

In our live site, each dynamic request requires loading session and site configuration data, validating the security of the request, and making several trips to the database to assemble a complete page from various compound templates. This induces a considerable disparity between the time required to serve dynamic and static requests, between two and three orders of magnitude. Very simple, lightly-featured templated systems may have a smaller gap, and might not benefit as drastically from our approach. However, the current trend is toward richer, more complex, programmable content management systems.

In our case (as in most dynamic communities), the static cache is not applicable to human users. We rely on the property that crawlers are fed "guest-level" content, which makes this segment of the population cacheable. For instance, each page loaded by a logged-in human user includes a check for private messages and online chat updates, as well as filtering out posts from "ignored" users, and an applied security model to control access to paid-subscription areas; this transient data and high degree of customization make the pages uncacheable for human users with these techniques.

6. RELATED WORK

Caching has long been studied and recognized as an effective way to improve performance in a variety of environments and at all levels of abstraction, including operating system kernels, file systems, memory subsystems, databases, interpreted programming languages, and server daemons. Caching in general is a method for transparently storing data such that future requests for the same data can be served faster. Our work, to our knowledge, is the first to methodically study crawlers in the context of caching to reduce server load, and to suggest how these crawler overload can be mitigated as a result of a few readily observable crawler properties. A distinguishing feature of our work is a uniform increase in throughput without resorting to caching for human users.

Caching techniques for static websites have been studied thoroughly [12, 21, 24]. Most of these techniques do not apply generally to dynamic websites, due to the inherent customization in dynamic sites. As a result, many different caching approaches for dynamic websites have been proposed. Research into caching for dynamic websites is usually implemented at various layers of abstraction. For instance, a dynamic website may include a half dozen cache mechanisms: at the database layer [11, 25], data interface layer [11], scripting layer, virtual file system, and the network proxy layer [14]. Several cache systems for dynamic websites attempt to map underlying queries to cache objects for intelligent invalidation [13, 20, 25]. The web application as a whole may also include several programmatic caches to cache repeated function results, web objects, and templates [3].

One phenomenon related to our work is the Flash Crowd; non-malicious, sudden onslaught of web traffic that can cripple server performance [19]. While burdensome to servers, high load crawlers are relatively uniform in their accesses and do not fall under the guise of flash crowds. Other recent works study the mitigation of flash crowds [15, 26], but these techniques rely on CDN's and additional servers to disperse load. Furthermore, our work targets crawlers specifically, which allows server throughput to increase uniformly while still providing dynamic content for human users.

Our work includes a measurement study of web crawler access characteristics on a busy dynamic website to motivate our two-level cache design. Previous work measures crawler activity [17] in detail, but do not study dynamic sites at our scale, and as a result, the crawlers behave differently. Our work shows that crawlers can consume a considerable percentage of overall server load, and hence, should be handled differently than human users. Other works include detection of crawlers through examination of access logs and probabilistic reasoning [22, 23]. Our requirements are more relaxed, only that we can detect high-load crawlers quickly and efficiently.

7. CONCLUSION

Search engines are essential for users to locate resources on the web, and for site administrators to have their sites discovered. Unfortunately, crawling agents can overburden servers, resulting in blank pages and crawler overload. Fortunately, high load crawlers are easy to identify using simple heuristics. We conducted a measurement study to show that crawlers exhibit very different usage patterns from human users, and thus can be treated differently than humans. By generating a static version of a dynamic website during off-peak hours, crawlers can be adequately served fresh content from the crawler's perspective, reducing load on the server from repeated dynamic page requests. Crawlers are archival in nature and do not require the same level of updates as human users, and this property should be taken advantage of by site administrators. Since static requests can be served two to three orders of magnitude faster than dynamic requests, overall server load can be practically reduced by serving crawlers using a static cache mechanism. We have developed a two-level cache system with an LRU policy, which is fast, straightforward to implement and can achieve a high cache hit ratio. Through a real website, we have demonstrated that our caching approach can effectively mitigate the overload risk imposed by crawlers, providing a practical strategy to survive the search engine overload.

8. REFERENCES

[1] Robots.txt - standard for robot exclusion. `http://www.robotstxt.org`, 1994.

[2] Apache ttfb module. `http://code.google.com/p/mod-log-firstbyte`, 2008.

[3] Memcached - open source distributed memory object caching system. `http://memcached.org`, 2009.

[4] Google official blog: Using site speed in web search ranking. `http://googlewebmastercentral.blogspot.com/2010/04/using-site-speed-in-web-search-ranking.html`, 2010.

[5] Apache httpd server. `http://httpd.apache.org`, 2011.

[6] Controlling crawling and indexing with robots.txt. `http://code.google.com/web/controlcrawlindex/docs/robots_txt.html`, 2011.

[7] libmemcached client library for the memcached server. `http://libmemcached.org`, 2011.

[8] Php: Hypertext preprocessor. `http://www.php.net`, 2011.

[9] Uriparser, rfc 3986 compliant uri parsing library. `http://uriparser.sourceforge.net`, 2011.

[10] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *ACM SIGMETRICS'98*, pages 151–160, Madison, WI, 1998.

[11] S. Bouchenak, A. Cox, S. Dropsho, S. Mittal, and W. Zwaenepoel. Caching dynamic web content: Designing and analysing an aspect-oriented solution. In *Middleware'06*, Melbourne, Australia, 2006.

[12] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *IEEE INFOCOM'99*, New York City, NY, 1999.

[13] K. S. Candan, W.-S. Li, Q. Luo, W.-P. Hsiung, and D. Agrawal. Enabling dynamic content caching for database-driven web sites. In *ACM SIGMOD'01*, pages 532–543, Santa Barbara, CA, 2001.

[14] P. Cao, J. Zhang, and K. Beach. Active cache: caching dynamic contents on the web. In *Middleware'98*, London, UK, 1998.

[15] C.-H. Chi, S. Xu, F. Li, and K.-Y. Lam. Selection policy of rescue servers based on workload characterization of flash crowd. In *Sixth International Conference on Semantics Knowledge and Grid*, pages 293–296, Ningbo, China, 2010.

[16] E. Courtwright, C. Yue, and H. Wang. Efficient Resource Management on Template-based Web Servers. In *IEEE DSN'09*, Lisbon, Portugal, 2009.

[17] M. D. Dikaiakos, A. Stassopoulou, and L. Papageorgiou. An investigation of web crawler behavior: characterization and metrics. In *Computer Communications*, 28:8, 80–897, Elsevier, May 2005.

[18] JelSoft, Inc. vBulletin Forum Software. `www.vbulletin.com`.

[19] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: characterization and implications for cdns and web sites. In *WWW'02*, pages 293–304, Honolulu, HI, 2002.

[20] Q. Luo, J. Naughton, and W. Xue. Form-based proxy caching for database-backed web sites: keywords and functions. *The VLDB Journal*, 17:489–513, 2008.

[21] P. Rodriguez, C. Spanner, and E. Biersack. Analysis of web caching architectures: hierarchical and distributed caching. In *IEEE/ACM Transactions on Networking*, 9(4):404–418, 2001.

[22] A. Stassopoulou and M. D. Dikaiakos. Crawler detection: A bayesian approach. In *ICISP'06*, Cap Esterel, France, 2006.

[23] A. Stassopoulou and M. D. Dikaiakos. Web robot detection: A probabilistic reasoning approach. In *Computer Networks*, 53:265–278, 2009.

[24] J. Wang. A survey of web caching schemes for the internet. In *ACM Computer Communication Review*, 29:5, 36–46, 1999.

[25] I.-W. T. Yeim-Kuan Chang and Y.-R. Lin. Caching personalized and database-related dynamic web pages. In *International Journal of High Performance Computing and Networking*, 6(3/4), 2010.

[26] K. Yokota, T. Asaka, and T. Takahashi. A load reduction system to mitigate flash crowds on web server. In *International Symposium on Autonomous Decentralized Systems '11*, Kobe, Japan, 2011.