KYOUNGSOO PARK AND VIVEK S. PAI

CoBlitz: a scalable large-file transfer service

KyoungSoo Park is a fourth year Ph.D. studentin computer science at Princeton University. His research focuses on the performance and security issues in large-scale distributed systems such as content distribution networks (CDNs) and the domain name system (DNS).

kyoungso@cs.princeton.edu

Vivek Pai is an Assistant Professor at Princeton University, where his research spans operating systems, networking, and high-performance applications. His research group is responsible for the CoDeeN system, described at http://codeen.cs.princeton.edu.

vivek@cs.princeton.edu

WITH THE RECENT EXPLOSIVE GROWTH in the scale and the size of Internet file downloads, we need techniques that provide high performance without burying servers under heavy loads. CoBlitz provides a timely solution to this, using unmodified Web browsers and servers as its clients and origin servers, a locality-aware front-end network, and a bunch of caching reverse proxies that have shown performance that meets or exceeds BitTorrent in most cases. And, best yet, CoBlitz is available for use today.

As bandwidths to the home increase, large file downloads are becoming increasingly popular on the Internet, with movies and software distributions commonly ranging from the hundreds of megabytes to several gigabytes. In its first five months of providing videos, Apple's iTune store provided over 15 million copies of TV shows and movie trailers, and Google Video now provides free downloading of thousands of video clips, from humorous home videos to professional music videos. By using small display sizes and high compression ratios, these files tend to be relatively small compared to HD-quality broadcast. However, as end-user bandwidths increase, we can imagine services providing much higher-quality downloads, with a corresponding increase in file sizes. New versions of Linux have long been distributed through the Internet, and their mirror sites get very busy shortly after every new release.

For sites that have a burst of traffic after every new release and whose users are sufficiently sophisticated, peer-to-peer protocols such as BitTorrent can be useful to offload traffic from the origin server and to leverage bandwidth available from the clients. For other scenarios, however, this option may not be as attractive, particularly if the content is not bursty or predictable, if the user population does not have browser plug-ins, or if the content consumers are other programs that only understand HTTP transfers. In these cases, it may be desirable to have a managed service that can offload the large file traffic from the origin, while still providing transfer via standard HTTP.

Although Content Distribution Networks (CDNs) have successfully provided this service commercially for standard Web content, very large files can pose some challenges for them. In particular, CDNs typically exploit main-memory caching of

Web objects, since they have a whole-file access model and a mean transfer size of around 10 kilobytes. Main-memory caching allows CDNs to reduce latency (since main-memory transfers can be hundreds of times faster than disk) and improve throughput, by avoiding disk seeks. If CDN providers start serving multi-gigabyte transfers from these same boxes, the competition for the machine's main memory can result in thousands of small files being evicted when a large file becomes popular. If several large files become popular simultaneously, then the main memory may not have enough room to cache them all and will thrash on disk accesses.

This is the scenario that we examine: how to provide a service that can *transparently* support the efficient transfer of very large files, using standard HTTP infrastructure (clients and servers). It should be capable of handling flash crowds as well as files that have a longer-lived demand, all without pre-positioning content or rehosting or reformatting the files.

Our Solution: CoBlitz

Our solution to this problem is a system called CoBlitz, which operates without any modification of the HTTP protocol, the servers, or the clients. CoBlitz evenly distributes the load of handling large-file requests to many participating nodes, to maximize resource utilization and reduce the origin server load dramatically.

The main idea of CoBlitz is to transform the large-file distribution problem into a regular small-file CDN scenario. CoBlitz transparently splits large-file requests into many requests for pieces of the file, called chunks, and has the chunks cached at multiple proxies in the content distribution network. Each chunk represents a certain range of a file, but with a little tweaking, the proxy handles it like a regular small file, thus benefiting from whatever caching strategy the proxy uses. To reduce the memory consumption on each node, CoBlitz arranges downloads so that each node is only responsible for caching specific ranges of the file. The server sees a reduction in traffic once CoBlitz caches the file to be served. Clients also see downloading speed improvement, because CoBlitz takes advantage of parallel chunk downloads while delivering to its client.

Another benefit lies in the easy deployment: CoBlitz looks like any other Web CDN, and using it is as simple as rewriting the links to be served to point to CoBlitz instead of directly referring to the origin server. The CoBlitz service just looks like a regular Web server to the client, so existing browsers, download tools such as wget and curl, or even Web services agents will all operate normally.

The Gory Details

How is this possible? Figure 1 shows the basic operation of CoBlitz. To incrementally build on an existing CDN, all of the "smarts" of CoBlitz are contained in a daemon that runs as a separate process on each CDN node. This agent looks like a standard Web server from the outside, but internally it splits the large-file request into many chunk requests, sends them to the CDN, merges the responses on the fly, and delivers them to the client in order. To improve the end-user throughput by multi-path parallel chunk downloads, it keeps a TCP-like sliding window of "chunks" and dynamically adjusts the window size to prevent the origin server or the infrastructure itself from getting overloaded.

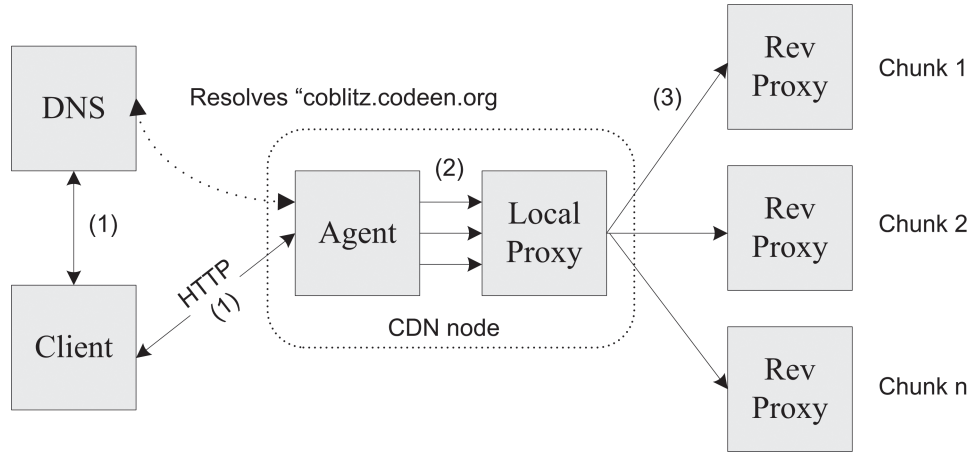

FIGURE 1: STEP-BY-STEP COBLITZ OPERATION.

Here is how CoBlitz works step by step:

1. A browser (or any HTTP-aware client) asks for a CoBlitzed URL. The format of a CoBlitzed URL is http://coblitz.codeen.org:3125/URL, where the URL is the original link before CoBlitz. Then the browser resolves the name "coblitz.codeen.org," which finds the closest CDN node to the client, and sends the request to it. The agent running on the CDN node is listening on port 3125 and accepts the request from the client.
2. Upon receiving the large-file request, the agent splits it into chunk requests and hands them to its local proxy. The local proxy runs a deterministic hashing algorithm (called Highest Random Weight [5]) to map each chunk request to a reverse proxy in its peer set. Selecting peers and maintaining the peer set are being done at the CDN level; this topic will be discussed later in the article.
3. The selected reverse proxy receives and serves the chunk request. If the chunk request is a cache hit, it is served from its cache right away, but in case of a cache miss, the reverse proxy fetches it from the origin server. The reverse proxy uses an HTTP/1.1 byte-range query to fetch the chunk rather than the whole file from the origin server. After fetching the chunk, the reverse proxy caches it as a small file rather than a range of a file.

As mentioned, the agent keeps a sliding window of chunks, and it retries any slow chunk via a different replicated reverse proxy. The retry timeout is calculated by a combination of the exponentially weighted moving average and standard deviation for recent chunks. For each retry, the timeout exponentially backs off to avoid getting overly aggressive. We allow up to two parallel downloads per chunk and let them compete with each other in case of retry. In practice, we see that about 10–20% of the chunks are retried.

The size of the chunk window gets adjusted as the transfer progresses. Whenever a retry kills off the head chunk in the window, we decrease the window size by one chunk. So when there is a problem, we can shut down the whole window within one maximum round-trip time (RTT). We increase the window size by $1/\log(x)$ chunks, where x is the current window size (i.e., we use 1 when $x = 1$), when a chunk finishes in less than the average chunk downloading time. We increase a bit aggressively when the window size is small as with the "slow start" phase in TCP, but when the window size converges, the window growth slows down.

Challenges

Although the basic algorithm is relatively simple, the real challenge is to run it in a real distributed environment. With CoDeeN as its base delivery CDN [6], CoBlitz has been operational on PlanetLab [4] (on 600+ nodes, at 300+ sites, and in 30+ countries) for over two years. With the feedback of its real users, CoBlitz has fixed a number of problems in operation and evolved its peering algorithm.

CoBlitz adopts *unilaternal, asynchronous peering* as its peer-selection policy. Each node is independent in choosing its own peers (reverse proxies) and does not depend on any synchronous group membership maintenance algorithm, which can incur prohibitive delays in practice. The rationale behind this decision is to favor simplicity and robustness and to survive partial network connectivity [2] problems with minimal effort. As a result, scalability is easily achieved, because one can simply add more nodes as they are available, without changing or reconfiguring the peering structure.

However, unilateral peering does not guarantee perfect clustering (a clique in which each member knows which nodes other members are peered with) by design and can produce many different target reverse proxies for the same chunk, owing to differences in the peering sets. This behavior can be undesirable, because it can overload the origin server in case of cache misses, reducing resource utilization. To address the problem, we introduce *proximity-based multihop routing*, which routes the request to the best peer in the local neighborhood. Instead of going to the origin server from the first hop, each hop reruns the HRW algorithm with its own peer set to see whether any better node exists, and it reforwards the request to the better node if it exists. Each hop repeats this process until there is no better node and only the last node, a local optimum node, sends the request to the origin server. This algorithm creates an implicit overlay tree for each chunk, and nodes in the path to the origin cache the chunk while delivering it to their descendants. In this way, if other nodes near the intermediate nodes in the tree look for the chunk, the request is served without getting to the best node, distributing the load. In practice, 3–15% of chunks require an extra hop, and less than 1% of chunks are forwarded more than once.

Performance

To get some sense of the relative performance of this system, in Figure 2 we compare CoBlitz with BitTorrent and direct downloading. We use 400 PlanetLab nodes around the world as clients to simultaneously fetch a 50MB file from a single Web server at Princeton. Although CoBlitz is not intended to replace BitTorrent, this test gives some sense of CoBlitz as a reasonable choice for similar scenarios. This particular test is designed to resemble a flash crowd, and more tests can be found in our paper [3]. We tune BitTorrent for performance, allowing the origin and all peers to act as seeds for the duration of the test. BitTorrent clients take a variable amount of time to find their peers, and for the sake of a fair comparison, we have the CoBlitz clients delay by the same amounts. We show four measurements: direct downloading from the origin, BitTorrent, CoBlitz (in order), and out-of-order CoBlitz. Whereas CoBlitz normally operates by delivering all data in order to the client, we can configure it to deliver chunks as they are ready, and let the client assemble them. (Although this breaks our goal of using unmodified clients, it also allows us to see what price we pay for HTTP compatibility.) Other tests are shown in our paper [3].

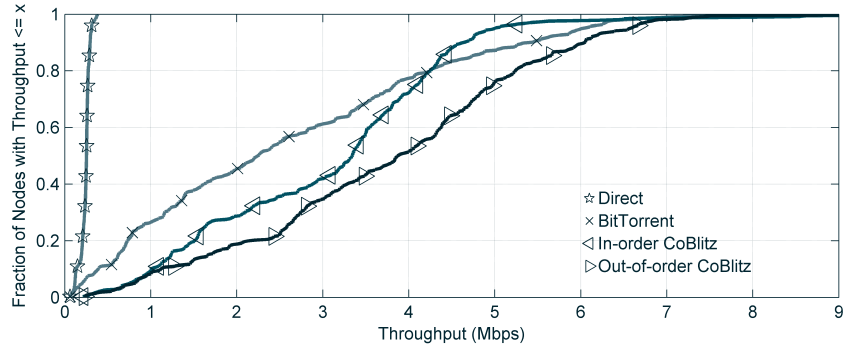

FIGURE 2: THROUGHPUT DISTRIBUTION FOR ALL LIVE PLANET-LAB NODES.

The most obvious lesson from this test is that both CoBlitz and BitTorrent are greatly preferable to having a large number of clients download directly from a single server. Even a well-connected campus such as Princeton is able to achieve only 250 Kbps on average to our worldwide clients. BitTorrent does much better, achieving 2.52 Mbps. CoBlitz outperforms BitTorrent at 79% of the clients, and it achieves an average download rate of 2.99 Mbps. The out-of-order CoBlitz shows the absolutely best performance at 3.68 Mbps, beating BitTorrent across the spectrum by 55–86%. The higher performance comes at the cost of incompatibility and would require our own browser plug-in, so we do not deploy this option.

In addition to better performance, CoBlitz better utilizes the contents fetched from the origin server as well. By fetching one copy from the origin server, CoBlitz serves 43–55 other nodes, whereas BitTorrent serves about 35 nodes. CoBlitz achieves about a 98% cache hit rate in this test, even when the document has never been seen before, dramatically reducing traffic to the origin.

Can I Use It Now?

CoBlitz was designed and developed to easily provide public access and is being used as Fedora Core mirror in six different locations worldwide, as well as a document-caching server for the CiteSeer Digital Library [1], providing over 50,000 papers through CoBlitz. Figure 3 shows the aggregate throughput of the CoBlitz Fedora mirror when Fedora Core 5 was released, on March 20, 2006. We have only a single origin server to serve Fedora Core, but with the help of CoBlitz, it achieved a peak delivery throughput of 700 Mbps and sustained over 400 Mbps for several days. This traffic was client-limited; even at these rates, we had extra capacity in our system, since our other tests have shown aggregate throughputs as high as 3 Gbps.

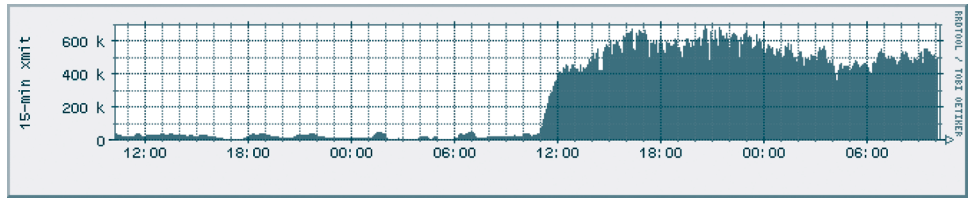

FIGURE 3: COBLITZ TRAFFIC IN MBPS (LABELED AS "K") ON RELEASE OF FEDORA CORE 5, AVERAGED OVER 15-MINUTE INTERVALS.

In running CoBlitz as a public service, we have to balance simplicity with operational overhead. As mentioned earlier, using CoBlitz simply involves adding the prefix "coblitz.codeen.org:3125" to any URL. For example, if you want to serve http://www.example.com/big-file.zip through CoBlitz, you simply need to convert it to http://coblitz.codeen.org:3125/www.example.com/big-file.zip and make it a link on any page you want. However, allowing this to happen to any URL would open us to unlimited bandwidth-shifting and possibly other forms of abuse, such as transferring copyrighted material without permission. To keep this service running for the technical community, we have restrictions in place that disallow public use of CoBlitz for entertainment media formats (e.g., images, audio, or video) but allow downloads of software, PDF documents, etc. Full details are at our Web site, http://codeen.cs.princeton.edu/coblitz/. Universities can use it virtually without restriction, as can other sites we whitelist. If you have a technical or nonprofit site and would like to try CoBlitz for your media transfers, please send email to KyoungSoo (kyoungso@cs.princeton.edu) to inquire about getting added to the whitelist.

REFERENCES

[1] CiteSeer Scientific Literature Digital Library: http://citeseer.ist.psu.edu/.

[2] M. J. Freedman, K. Lakshminarayanan, S. Rhea, and I. Stoica, "Non-transitive Connectivity and DHTs," in *Proceedings of the 2nd Workshop on Real, Large Distributed Systems (WORLDS '05)* (Berkeley, CA: USENIX, 2005).

[3] K. Park and V. S. Pai, "Scale and Performance in the CoBlitz Large-file Distribution Service," in *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI '06)* (Berkeley, CA: USENIX, 2006).

[4] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet," in *Proceedings of the First ACM Workshop on Hot Topics in Networks (HotNets-I)* (Princeton, NJ, 2002).

[5] D. G. Thaler and C. V. Ravishankar, "Using Name-based Mappings to Increase Hit Rates," *IEEE/ACM Transactions on Networking,* 6(1) (Feb. 1998): 1–14.

[6] L. Wang, K. Park, R. Pang, V. S. Pai, and L. Peterson, "Reliability and Security in the CoDeeN Content Distribution Network," in *Proceedings of the 2004 USENIX Annual Technical Conference* (Berkeley, CA: USENIX, 2004).