Brief Announcement: Building an Adaptive Distributed Web Server System on the Fly for Handling Web Hotspots

Weibin Zhao and Henning Schulzrinne
Department of Computer Science
Columbia University
New York, NY 10027

{zwb,hgs}@cs.columbia.edu

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Reliability, availability, and serviceability; C.5.5 [**Computer System Implementation**]: Servers; C.2.4 [**Computer-communication Networks**]: Distributed systems

General Terms

Performance, Design, Experimentation, Algorithms

Keywords

Web hotspots, overload control, adaptive distributed server systems, service discovery, scalability

As more web sites experience a request load that can no longer be handled by a single server, using multiple servers to serve a single site becomes a widespread approach. Traditionally, a distributed web server system has used a fixed number of dedicated servers based on capacity planning, which works well if the request load is relatively constant and matches the planned capacity. However, web requests could be very bursty. A well-identified problem web hotspots (a.k.a., flash crowds or the Slashdot effect) may trigger a large load increase but only last for a short time. For such situations, overprovisioning a web site is not only uneconomical but also difficult since the peak load is hard to predict.

To handle web hotspots effectively, we advocate dynamic allocation of server capacity from a server pool distributed globally because the access link of a local network could become a bottleneck. As an example of global server pools, content delivery networks (CDNs) have been used by large web sites, but small web sites often cannot afford the cost particularly since they may need these services very rarely. We seek a more cost-efficient mechanism. As different web sites (e.g., different types or in different locations) are less likely to experience their peak request loads at the same time, they could form a mutual-aid community, and use spare capacity in the community to relieve web hotspots at any individual site. Based on this observation, we designed *DotSlash*, which allows a web site to build an adaptive distributed web server system on the fly to expand its capacity by utilizing spare capacity at other sites. Using DotSlash, a web site not only has a fixed set of *origin servers*, but also has a changing set of *rescue servers* drafted from other sites. A web server allocates and releases rescue servers based on its load conditions. The rescue process is completely self-managing and transparent to clients. Note that DotSlash does not aim to support a request load that is persistently higher than a web site's planned capacity, but rather to complement the existing web server infrastructure to handle short-term load spikes effectively.

PODC'04, July 25–28, 2004, St. Johns, Newfoundland, Canada.
ACM 1-58113-802-4/04/0007.

DotSlash consists of service discovery, workload monitoring, request redirection, dynamic virtual hosting, and rescue control. Service discovery enables servers of different web sites to learn about each other dynamically and collaborate automatically without any administrator intervention. Workload monitoring allows a web server to react quickly to load changes. We focus on monitoring outbound HTTP traffic within a web server since network bandwidth is the most constrained resource for most web sites during hotspots. Request redirection allows an origin server to offload client requests to its rescue servers. We use two mechanisms for request redirections: DNS round robin at the first level for crude load distribution, and HTTP redirect at the second level for fine grained load balancing. Dynamic virtual hosting enables a rescue server to serve the content of its origin servers on the fly, without the need of any advance configuration. A rescue server works as a reverse caching proxy for its origin servers.

Rescue control allows a web server to tune its resource utilization by using rescue actions. To ensure that the network bandwidth utilization ρ_n remains within the desired load region $[\rho_n^l, \rho_n^u]$, overload control actions are triggered if $\rho_n > \rho_n^u$, and under-load control actions are triggered if $\rho_n < \rho_n^l$. A web server becomes an origin server if it has allocated rescue servers. An origin server uses the redirect probability P_r as the major control parameter: it increases P_r if $\rho_n > \rho_n^u$, and decreases P_r if $\rho_n < \rho_n^l$. An origin server allocates additional rescue servers if it has run out of the redirect capacity, and releases all rescue servers if it has not redirected requests to rescue servers for a configurable number of consecutive control intervals. In contrast, a web server becomes a rescue server if it has accepted rescue requests. A rescue server uses λ_{rd}^a, the allowed redirect data rate for its origin servers, as the major control parameter: it decreases λ_{rd}^a if $\rho_n > \rho_n^u$, and increases λ_{rd}^a if $\rho_n < \rho_n^l$. Note that a rescue server maintains a separate λ_{rd}^a for each of its origin servers. A rescue server accepts new rescue requests if $\rho_n < \rho_n^l$, and shutdowns a rescue relationship if $\rho_n > \rho_n^u$ and $\lambda_{rd}^a = 0$, or the rescue relationship has been idle for a configurable number of consecutive control intervals.

We have implemented a prototype of DotSlash on top of Apache, which consists of an Apache module that supports DotSlash functions related to client request processing such as accounting for each response, HTTP redirect, and dynamic virtual hosting, and a daemon that accomplishes other DotSlash functions such as service discovery, dynamic DNS updates, and rescue control. Experiments show that using DotSlash a web server can increase the request rate it supported and the data rate it delivered to clients by an order of magnitude, even if only HTTP redirect is used. Once DNS redirection is incorporated, a web site can further improve its performance and scalability.