Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web*

David Karger[1] Eric Lehman[1] Tom Leighton[1,2] Matthew Levine[1] Daniel Lewin[1]
Rina Panigrahy[1]

Abstract

We describe a family of caching protocols for distrib-uted networks that can be used to decrease or eliminate the occurrence of hot spots in the network. Our protocols are particularly designed for use with very large networks such as the Internet, where delays caused by hot spots can be severe, and where it is not feasible for every server to have complete information about the current state of the entire network. The protocols are easy to implement using existing network protocols such as TCP/IP, and require very little overhead. The protocols work with local control, make efficient use of existing resources, and scale gracefully as the network grows.

Our caching protocols are based on a special kind of hashing that we call *consistent hashing*. Roughly speaking, a consistent hash function is one which changes minimally as the range of the function changes. Through the development of good consistent hash functions, we are able to develop caching protocols which do not require users to have a current or even consistent view of the network. We believe that consistent hash functions may eventually prove to be useful in other applications such as distributed name servers and/or quorum systems.

1 Introduction

In this paper, we describe caching protocols for distributed networks that can be used to decrease or eliminate the occurrences of "hot spots". *Hot spots* occur any time a large number of clients wish to simultaneously access data from a single server. If the site is not provisioned to deal with all of these clients simultaneously, service may be degraded or lost.

Many of us have experienced the hot spot phenomenon in the context of the Web. A Web site can suddenly become extremely popular and receive far more requests in a relatively short time than it was originally configured to handle. In fact, a site may receive so many requests that it becomes "swamped," which typically renders it unusable. Besides making the one site inaccessible, heavy traffic destined to one location can congest the network near it, interfering with traffic at nearby sites.

As use of the Web has increased, so has the occurrence and impact of hot spots. Recent famous examples of hot spots on the Web include the JPL site after the Shoemaker-Levy 9 comet struck Jupiter, an IBM site during the Deep Blue-Kasparov chess tournament, and several political sites on the night of the election. In some of these cases, users were denied access to a site for hours or even days. Other examples include sites identified as "Web-site-of-the-day" and sites that provide new versions of popular software.

Our work was originally motivated by the problem of hot spots on the World Wide Web. We believe the tools we develop may be relevant to many client-server models, because centralized servers on the Internet such as Domain Name servers, Multicast servers, and Content Label servers are also susceptible to hot spots.

1.1 Past Work

Several approaches to overcoming the hot spots have been proposed. Most use some kind of replication strategy to store copies of hot pages throughout the Internet; this spreads the work of serving a hot page across several servers. In one approach, already in wide use, several clients share a *proxy cache*. All user requests are forwarded through the proxy, which tries to keep copies of frequently requested pages. It tries to satisfy requests with a cached copy; failing this, it forwards the request to the home server. The dilemma in this scheme is that there is more benefit if more users share the same cache, but then the cache itself is liable to get swamped.

Malpani et al. [6] work around this problem by making a group of caches function as one. A user's request for a page is directed to an arbitrary cache. If the page is stored there, it is returned to the user. Otherwise, the cache forwards the request to all other caches via a special protocol called "IP Multicast". If the page is cached nowhere, the request is forwarded to the home site of the page. The disadvantage of this technique is that as the number of participating caches grows, even with the use of multicast, the number of messages between caches can become unmanageable. A tool that we develop in this paper, *consistent hashing*, gives a way to implement such a distributed cache without requiring that the caches communicate all the time. We discuss this in Section 4.

Chankhunthod et al. [1] developed the Harvest Cache, a more scalable approach using a *tree* of caches. A user obtains a page by asking a nearby leaf cache. If neither this cache nor its siblings have the page, the request is forwarded to the cache's parent. If a page is stored by no cache in the tree, the request eventually reaches the root and is forwarded to the home site of the page. A cache

* This research was supported in part by DARPA contracts N00014-95-1-1246 and DABT63-95-C-0009, Army Contract DAAH04-95-1-0607, and NSF contract CCR-9624239

[1] Laboratory for Computer Science, MIT, Cambridge, MA 02139.
email: {karger,e_lehman,danl,ftl,mslevine,danl,rinap}@theory.lcs.mit.edu
A full version of this paper is availble at:
http://theory.lcs.mit.edu/~ {karger,e_lehman,ftl,mslevine,danl,rinap}
[2] Department of Mathematics, MIT, Cambridge, MA 02139