SOA & Web 2.0: The Connection Management Challenge

Lori MacVittie, 2008-21-02

A Quick History Lesson

Back in the day, server-load balancing vendors figured out that connection management (the setup and teardown of TCP/IP connections) was actually quite a burden on servers. You see, the server not only had to spend time setting up and tearing down the connection, but it also had to keep track of those connections in something we like to call a "session state table".

The problem was, and still is, that a server has a limited amount of memory and can only manage X number of connections concurrently. This is primarily a matter of configuration of the web server and hardware resources. Once the server is no longer able to open another connection, attempts to connect begin to time out, leaving the user to suffer what is lovingly known as the /. or Fark effect.

This is, essentially, the original problem that load-balancing vendors sought to solve.

One of the reasons load-balancing intermediaries did so well at this task was because their session state tables could be much larger. After all, they weren't doing other things like running lots of daemons and worrying about synching up read/writes on hard drives. The intermediaries only had to worry about one thing: load balancing requests across a farm/pool/cluster of servers.

Around the year 2001 or so, load balancing became more than just a way to ensure availability of a site. It started to concern itself with optimization - not only of the application data it was delivering, but squeezing the most out of the resources it was tasked to manage: the servers. Thus was born TCP multiplexing; the ability of an intermediary to open - and keep open - a number of connections to the servers it manages and to reuse those connections as much as possible. Basically, it was HTTP 1.1 *on the back-end.*

This innovation in the industry resulted in even more capacity being squeezed out of servers because the servers had fewer connections to manage and therefore had more resources available to process requests - quickly. This improved application performance and continued to improve the overall value proposition of load balancers.

Enter SOA & Web 2.0

One of the primary culprits of poor performance of services and Web 2.0 applications - especially those based on AJAX - revolves around the increasing number of requests - and therefore connections - that are required. SOA applications comprise multiple services each of which necessarily require a TCP connection. Web 2.0 applications and sites comprise multiple components and widgets, each of which generally requires a TCP connection as well. Marrying the two, as has been suggested frequently of late, compounds the number of connections required faster than the interest on your credit card balances.

Not only do these additional connections inhibit performance due to increased overhead and latency, but they can also decrease the capacity of servers upon which components and services are deployed. A single user suddenly requires more resources than they have in the past, reducing the total number of users that can be serviced at any given time. Traditional formulas used to calculate capacity no longer seem to work because there are so many more connections involved in even a simple Web 2.0 or SOA application.

TCP multiplexing (F5 calls its implementation of this technology OneConnect) reduces the impact of SOA and Web 2.0 applications on the application infrastructure by reusing connections on the back-end, thus maintaining capacity of your existing servers. By using existing resources more efficiently, an application delivery controller like BIG-IP can alleviate the requirement for additional servers, which decreases the expense associated with deploying SOA and Web 2.0 applications. And anything that **decreases** IT spending has got to be a good thing, right?

This technology also improves performance by removing the overhead associated with opening and closing multiple TCP sessions (connections) between the client and the server. Sure, we're probably talking about milliseconds here, but that's milliseconds ***per request*** and that can all add up when you're considering a SOA or Web 2.0 application with 5,10, or more components/services per page. Especially given today's highly aggressive user demands when it comes to performance.

So if you're trying to figure out why your SOA or Web 2.0 application isn't performing so hot, consider its implementation and how many connections it may be using. Then consider introducing an application delivery controller into your architecture to address that connection management challenge.

Imbibing: Coffee

Technorati tags: MacVittie, F5, BIG-IP, application delivery controller, Web 2.0, SOA, load balancing

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com