Intro Load Balancing for Developers &ndash; the Architect&rsquo;s View

Don MacVittie, 2009-11-03

Okay, there are a zillion bits about load balancing including introductions and articles for developers, but I'm throwing this out in a blog format so we can be more chatty and less "instructional". To that end, I'll be a *lot* more laid back than what you're used to reading on the topic, but the whole point of this (and any follow-on) blogs is to get you to *understand* rather than just *know* load balancing. If you already understand, join me, there's a comment form below, tell the other readers what I've missed or understated.

So you've built your web app ZapNGo, and it rocks. You deployed it on a big machine, and before you knew it, the machine was overloaded and users were getting timeouts. This wasn't meant to be so busy, so your first step is to move the database off to another machine, freeing up a ton of memory, disk I/O and CPU time, and costing you a (relatively) modest amount of network time. Sure, some of you – at least in your day jobs – put the database on a separate machine to begin with, but the app that takes off is inevitably the one that was developed on the side, so humor me.

Things run fine for a little while, but eventually, you max out your machine again. If your overload is CPU or RAM, the first place you look is at the app design itself. Maybe you improve it, maybe you don't – depends upon your organization, available resources, the app design, and overall complexity of the problem it solves. Let us assume you found some problems (did you really have to create a local copy of that variable-sized array in your recursive routine? An array built from the result of a query? ;-)).

All is peachy again, and then you get slash-dotted – or the new equivalent, twitter-feeded – and things go crazy. Now you've already optimized the app, you've moved the DB off, and with web applications (which we're discussing, because slash-dotting doesn't overload other apps) you can't generally move the web server and its bazillion associated bits off the machine, so you're stuck, right? Your application is now ZapNGone?

Nope, just need a little more work and a load balancer – or these days possibly an Application Delivery Controller (ADC). Suffice it to say that high-end load balancing is generally one feature of an ADC, so if you own one, you own a load balancer. We'll delve into the other bits of ADCs later, or you can read up on some of the stuff Lori has written in the last year or so (before she went surfing on the cloud ;-)).

If you've been programming for long, you likely know the vague bits of load balancing – Round Robin isn't a new term to you, and you know it helps expand applications – but how do you know you need one, what are your options, and how do you get started?

Well, you've got the "How do I know I need one" bit down from the above – if you've moved off what you can, and you've optimized the application, and it's still overloading the server it is hosted on, you can continue to throw hardware at it – by replacing the server, adding memory, adding network cards, whatever – or you can throw in a load balancer and another server. If your application is that busy, ***or if it isn't, and it is mission critical***, then you want the load balancer option. Seriously. You know that unless you're on some pretty darned high-end servers there is a limit to how far you can grow the webserver box. There isn't a limit to how many boxes you can throw behind a good load balancer, and there ***are*** limits to how many you can put behind a poor load balancer, but it's more than one, so that might be an option also. The reason mission critical is mentioned is because a load-balanced solution can keep going if a server drops. No matter how large you grow your server, if it goes, you're down. With a load balancer that's not true, if one box drops the others can pick up the slack and keep your site live. I don't know all of the load balancers on the market, so check with your chosen vendor. Our ADC can absolutely shift work off of dropped servers so your website stays live (this is one of the "so buy ours" parts, didn't want you to miss it ;-)).

Once you know you need one, let's talk a bit about what one. There are software load balancers – both Open Source and proprietary – there are hardware load-balancers, and there are hardware ADCs. There is also the cloud. Yeah, I said that, just don't tell Lori.

If you've got one app that's having performance problems (which aren't really problems - "too many users" shouldn't be considered a problem), and don't foresee adding any more, it is well worth checking out the cloud. Seriously, it's infrastructure on demand, and many cloud providers use or offer BIG-IP, giving you load balancing ***and*** all the other ADC functionality like security and web app acceleration. So price it out. Some are prohibitively expensive, all (that I know of) charge you by how much you use, much like CDNs do. But it might fit your model, so it's worth checking on.

If you're going to have more than the one application (ZapNComeBack is under development?), then the cost of the cloud will increase, and it's worthwhile to consider putting in something you can use for more than just load-balancing your application. That's where ADCs come in. I won't delve into all of the benefits at this time, but they're pretty huge – go read some of the excellent articles in the DevCentral Docs section or check out the DevCentral Codeshare (login required) to get an idea of what an ADC is capable of. While of course I think our primary competitors are less functional than we are, check their sites out too – and no, I'm not going to link you to our competitors, sheesh. If you're seriously on a budget, the software load balancers are worth checking out, and if you essentially have no budget, the Open Source load balancer is worth checking out – though remember that you still have to deploy it.

So how do you get started? Well the simplest option is to put two servers running the same software behind the load balancer… But it's rarely that simple. If they're both hitting the same DB, you can imagine there is potential for conflict, if the application has variable execution times, you are going to want to use something other than round robin load balancing (which sends requests sequentially to each server, then starts over)… The list goes on. There are some coding gotchas to using any load-balancing algorithm also, but I'm looking at how long this post is, and I'll put off mentioning them until a follow-up. I have a collection of links, but there is at least one link not in there I want to gather up before posting them.

So there we have it, the idea – how to figure out if you need one and what your options are… Next we'll talk about how to account for one in your code, complete with a list of links to explain issues and resolutions.

Don.

PS: I'm running a Storage series, I guess I'll turn this into a (much shorter) series too, since we're cut off before the intro is done, and I'm already thinking about the next step after this. So Wednesdays must be "Load Balancing Intro for Developers" day. Maybe when I'm done with them all, we'll bundle them as Tech Tips.

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