The Infrastructure 2.0 Trifecta

Lori MacVittie, 2009-09-06

Balancing Cost, Performance, and Capacity in the Cloud

There is a huge difference between provisioning applications to support capacity and provisioning them to support performance requirements. That as capacity increases performance decreases is one of the truisms of scalability that is likely to be one of the first axioms of cloud computing that will bite us in the proverbial rear-end while simultaneously reaching for our wallets.

Alistair Croll of BitCurrent has a couple of great charts that illustrate this point perfectly. He then goes on to discuss how that affects cloud computing in "The cloud's most important equation":

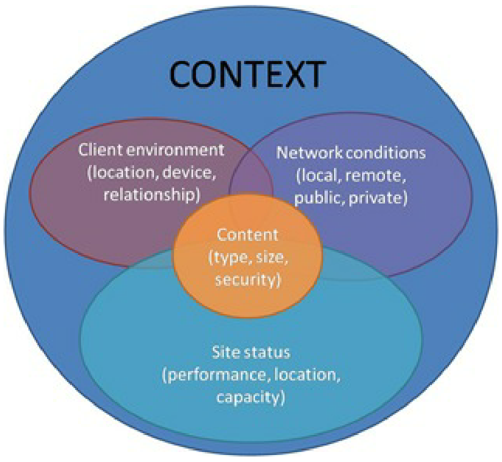
> *How will that tool or cloud know how to create new instances? Simple. It'll look at user experience. Sure, it may consider CPU load, or network usage, or free memory at first. But those are ultimately proxies for the only thing that matters: User experience. When the application's response time gets too slow, it's time to add capacity.*
>
> *…*
>
> *But in the past, you've been the one to decide what gets deployed in a data center. You controlled capacity; user experience was the variable. Now, the cloud controls capacity. The only knob you have is user experience.*
>
> *What this means is that autonomic provisioning and elastic computing systems will need to provide tools for their customers to adjust not only target customer experience, but also target spending. And because clouds are in the business of making money, they're unlikely to offer spending controls willingly to platform users.*

I won't try to replicate Alistair's math; suffice to say that it's nearly impossible to argue with the data and empirical evidence of thousands of customers: more users = more capacity used = degrading response time.

What's interesting about this discussion is there is a solution. The collaborative nature of Infrastructure 2.0 combined with the intelligence of a context-aware infrastructure mash-up to form just the solution we need to balance capacity-performance-cost ratios.

ASLEEP AT THE WHEEL

Load balancers that evolved into application delivery controllers spent some time under the name "traffic management." This implied control over the inbound and outbound flow of "traffic", or application data. The problem was – and still is in some cases – that folks ignore the advanced capabilities of an application delivery controller and treat it like a static, layer 4 load balancer. It still controls the flow of data, but it's essentially asleep at the wheel.


CONTEXT
Client environment (location, device, relationship)
Network conditions (local, remote, public, private)
Content (type, size, security)
Site status (performance, location, capacity)

Those advanced capabilities – application awareness, conditional execution of logic via network-side scripting, integration via standards-based APIs, and spanning both network and application layers of the stack – affords such a "traffic management" system with the ability to take on a unique role within emerging data center models that revolve around on-demand capacity. That role is not just traffic management, but *application management and delivery*.

See, once a system has arrived at contextual-awareness ***and*** it is capable of integration you suddenly have at your fingertips the ability to not only manage capacity as well as user-experience (response time) but also costs. The ability to perform autonomic provisioning exists and is utilized heavily already today in cloud environments. So managing capacity on-demand is not a problem at all.

Even managing capacity while managing costs by capping instances is not necessarily a problem. Putting limitations on the number of instances in total that can be "online" at any given time is not a technological issue. As Alistair points out it's a matter of whether the provider wants to allow that limitation as it necessarily keeps revenue growth slow.

What seems to be the problem is managing both capacity and costs while simultaneously maintaining an expected level of application performance, i.e. the end-user experience. But technically speaking ***that's*** not a problem either; it's simply a matter of ensuring that the right tools are used for the job. An application delivery controller, for example, is context-aware. This means that for any given application request it knows about everything from IP address to cookies to current network conditions to state of the connection to current application capacity and performance. It has all the information necessary to make decisions regarding ***where*** to direct requests in order to meet performance and capacity requirements, it just needs to factor in the capping of instances to control cost.

This is a perfect example of a complex problem that Infrastructure 2.0 can solve.

CODIFYING the EQUATION

In order to balance all three variables properly you need to be able to change the way in which application requests are directed when you reach "capacity", which we effectively redefine for this discussion to mean your ***financial*** capacity in terms of instances. So first you must be able to determine the number of instances that are currently active. If we've reached the maximum, as determined by you and your financial calculations, then we need to start worrying about balancing user-experience and capacity.

Most application delivery controllers are capable of load balancing traffic across your instances using the "fastest response time" algorithm. How response time is determined is variable, but in general it attempts to calculate accurately which instance/server is responding fastest. This is a good choice for maintaining good application response time. But once you've reached capacity (financial, remember, not technical) then you may need to adjust that algorithm to better distribute requests based on number of connections in order to use every last drop of capacity you can. You might want to switch to using what the industry calls "least connections" instead. The reason for switching to "least connections" is that you'd want to maximize the resources you have without oversubscribing the instances/servers. A faster response time could indicate more resources, but it could also simply indicate fewer dynamic requests or smaller content requests and doesn't always mean resources are available for more connections. At this point you're battling costs just to maintain availability and a ***least connections*** algorithm should provide a measure of assurance that requests ***can*** be served.

This is where the ability to essentially "redefine" the load balancing decisions made by an application delivery controller comes in handy. On some application delivery controllers you can change the load balancing method used to direct traffic based on, well, anything you want. By using a simple network-side scripting command you change the way in which load balancing is managed and change it back at will.

Voila. Load balancing method changed. Now requests are being distributed to the server with the least number of connections. This will slightly improve degrading application performance by ensuring the server with the "most" resources available is chosen instead of just choosing the "fastest" one ***and*** ensures you don't suddenly oversubscribe one server and completely destroy response time. And of course the logic can be written such that when specified conditions are met, the load balancing method is returned to "fastest response time".

TURN-KEY versus FLEXIBILITY

Don't get me wrong, this is not a panacea for the problem described by Alistair in his blog. It will not maintain ***optimal*** application performance levels and it might not even necessarily maintain ***acceptable*** application performance levels when capacity is constrained by financial considerations. A load balancing solution can optimize the distribution of requests to available resources but if you're out of resources, you're out of resources. Period. There are additional solutions an application delivery controller can provide that ***can*** assist in maintaining acceptable performance levels (application acceleration, for example, and both client and server-side caching) but those must be deployed and offered by the cloud provider before they can be applied to a cloud-based application, and it's almost certainly going to be the case that you're going to end up paying for that as well, which may very well make it a non-option.

But this about ***balancing*** the three variables, not necessarily ensuring all three are maximized. ***That*** may be impossible regardless of how flexible and willing a provider is to allow access to the knobs necessary to implement such controls. After all, as Alistair pointed out, in the "old days" you added more capacity – and it cost you. You weren't really balancing the three internally, you were adjusting as necessary the variables to arrive at the desired result. You had to make choices then, too. Adding more servers/instances addresses performance and capacity, but increases costs. Not adding more instances/servers addresses cost but to the detriment of performance and capacity. Prioritization has always been a part of IT and will continue to be whether applications reside locally or in the cloud.

I'm sure at this point ***someone*** is going to tell me how wrong the above solution is and that it won't work that way or that there's a better algorithm to choose from and I'll grant that from their perspective ***they're probably right.*** No two applications are the same, no two environments are the same, and there are a lot of variables that go into optimizing traffic management. The point is not necessarily that you should dynamically switch from "fastest response time" to "least connections" or that you should even be starting with "fastest response time" as a load balancing algorithm. The point is that Infrastructure 2.0 solutions are ***flexible*** enough to support that kind of modification in real time based on a variety of variables – ones you're going to have to define. You've got a better solution? You can implement it. ***That's*** the point.

This is exactly the kind of scenario in which a flexible, dynamic infrastructure really ***can*** offer a solution that's just not possible without the capabilities that make infrastructure dynamic. It may not be turn-key, but you can't have it both ways. You can't have flexibility and extensibility in a completely turn-key solution. Infrastructure 2.0 isn't going to, at least not for a long time, be turn-key, it's going to be flexible and able to integrate with the rest of the infrastructure – like the virtual instance management system – but unless you want vendors to dictate your architecture it's going to require some amount of effort on the part of IT or the provider.

In the end, though, Alistair is absolutely right when he says "because clouds are in the business of making money, they're unlikely to offer spending controls willingly to platform users." This isn't a technological problem, it's a business problem. And while technology – Infrastructure 2.0 - can certainly address the technical aspects of such a solution, it can't force a business – cloud provider or not - to implement it.

- All your data center are belong to … you
- The Context-Aware Cloud
- Infrastructure 2.0: As a matter of fact that's not what it means
- Control, choice, and cost: The conflict in the cloud
- The cloud's most important equation

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