Knapsacks, Shopping Carts and Application Workloads

Lori MacVittie, 2013-25-04

#geek Don't laugh or next week we'll talk fractals and tessellations and nature and tie that back to ERP. Somehow. Don't tempt me. I'm not kidding.

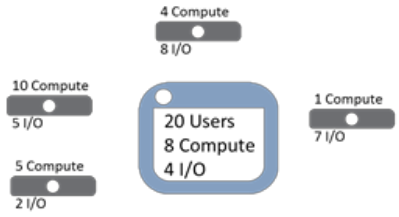
The other day I was shopping (on-line of course, because you know, Internets) as I often do: I was putting things into my shopping cart and occasionally pruning it back to fit within this imaginary total I was willing to spend. That's time consuming (albeit fun and sometimes agonizing) so I decided I should write an application that, when presented with X number of items and prices, could arrange those items in multiple carts, each under budget so I had all the possible combinations represented.

That, of course, immediately evoked thoughts of "How would you write that?" which immediately turned to the thought that it distilled down to a very basic knapsack problem.

Which in turn - you knew it was going there - led to the realization that really, capacity planning is as NP-complete as provisioning of compute in the cloud. It's as difficult a process as manually (or automatically) rearranging my shopping cart because applications are not workloads, and vice-versa. Not today's applications, anyway.

WORKLOAD != APPLICATION

An application today is actually comprised of multiple workload types, each of which exerts its own demands on compute and I/O that impact total capacity. We tend to base capacity planning around the notion of requests per second, or concurrent connections, or concurrent users, but those metrics are only vaguely related to actual compute and I/O consumption. The consumption profile of an application request that accesses a database is different than one performing analysis is different from one parsing and filtering data.



Example of a multiple-constraint knapsack problem: which servers should be chosen to maximize the number of requests while still keeping the overall capacity under or equal to 20 users requiring 8 compute and 4 I/O units?

These are the metrics upon which we would optimally base capacity, and yet we can't because they're all mixed up together in the same application and it is the application we are scaling, not individual workload types. The knapsack problem is NP-complete primarily because the objects being put into the knapsack are non-equivalent. They are all different, like the prices of clothing and various pairs of shoes I put in my shopping cart. And when you have a target you're trying to hit and using variably sized objects to hit it, you end up with a knapsack-like problem.

In an ideal, workload-driven architecture, we'd be able to recognize that X compute is needed for every request for a specific workload and thus if we want to support Y users or connections of that workload concurrently, we need Z compute. Determining how many "servers" is needed then becomes simple mathematics and it's no longer NP-Complete or NP-Hard or NP-anything.

If we could focus selecting a pool of resources based on one-constraint - I/O or compute - and match that to "users" needing that same constraint, we'd be making much more efficient use of resources all around. It would aid in sizing of "servers" because we match the RAM and CPU available to the demands placed on the hardware of the services being served.

There's a model that enables this, and it isn't the composite application model we're used to. Ultimately, the API model will provide the means by which we can more efficiently provision and scale "applications" by lessening the requirement to consider multiple variables.

Stay tuned for a more in-depth post on *that* topic....

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