CS787 Project Proposal

Brian Kroth

2013-03-04

1 Problem Description

The Computer Aided Engineering center at the College of Engineering of UW-Madison has a fairly complex virtual hosting infrastructure that it provides for the college and other university wide services. The ultimate issue we would like to be able to answer is how best to size our server infrastructure to suit the service's needs using some variant of bin packing, instead of trial and error. In order to explore how we might do this, we first describe the system as it is currently laid out as follows.

1.1 Background

There are roughly 800 individual production virtual hosts (vhosts) of various types served by roughly 25 backend virtual machine (VM) servers that are proxied by 4 frontend servers. Each vhost has certain properties associated with it in a database such as an editing group, the certificate domain it belongs in [1], and a type, as well as a number of others. In order to save on memory footprint required, but still retain certain security features such as running different websites under different users, the vhosts are combined into roughly 650 security domains by these three features, and a single standalone Apache web server instance is dedicated to running each of these, with an individual `VirtualHost` directive for each vhost that belongs to that security domain.

A vhost's type might be one of HTML, Mod PHP, Mod Perl, etc. Each backend VM implements one of these types and represents a standard hardware configuration (eg: 4 GB RAM, 2 vCPU, etc.) for that type. The type of a vhost determines which type of VM a vhost can be assigned to and controls some of the vhost's default configuration parameters including how much memory a barebones vhost of that type is expected to take up. We are generally more concerned with memory than CPU usage, though certainly each request occupies a certain amount of CPU time. However, individual vhost code and settings customizations can increase that requirement, and since it is not currently practical to load test every individual vhost accurately, currently we tend to fairly crudely estimate the size of a given vhost type by using what we expect to be typical values and then possibly adjusting it by a factor based upon the load its requests generate on the system.

1.2 Vhost Workers

Currently a rudimentary assignment routine is used to balance the number of the vhosts across the available number of backends for a given type by considering a `workers` property on each vhost, which is currently set manually, while avoiding moving the "heaviest" vhosts if possible to prevent service interruption. The `workers` property is an integer field which is (ab)used in the following ways:

1. It represents the number of spare/idle Apache child processes [2] we configure for the vhost's security domain [3] to keep around to process incoming requests. The tradeoff here is that since each worker is a full separate process keeping more spares around requires more memory, but lowers the expected response time latency since we need to also incur the cost of a fork.

2. The `workers` property is assumed to represent the relative importance of the vhost (ie: if we expect more requests, then more people must care about the resource it represents), so we configure our monitor systems to check it more frequently. This has the side effect of possibly keeping more

[1] We use wildcard certificates as much as possible to save on IPv4 address requirements on the frontends, though we make use of IPv6 addresses extensively as well.

[2] For most vhost types we use a prefork Apache MPM rather than a threaded one, so each worker is a separate full process since most user code, especially PHP, doesn't properly support threading.

[3] Currently the maximum value of all vhosts in a security domain is used for that Apache instance, though under the proposed scheme that might be reworked to be the sum of the security domain's vhosts' `workers` property

children active after they otherwise would have been reclaimed in the case that the vhost goes idle for a period of time.

Currently the `workers` property defaults to a value of 2 for all vhosts in order to try and keep memory requirements low, since most vhosts are not particularly active. If the vhost has more activity than that, then extra children are automatically spawned to handle the incoming requests, and are automatically reclaimed when they've been idle for a sufficient period of time.

On the other hand, certain vhosts get enough activity or have extra redundancy requirements that they are assigned multiple backends and the frontend proxy is configured to balance and failover among them. This is currently controlled through a `secondary backend instances` integer field, and is also controlled manually.

1.3 Vhost Size

If we had

1. a good value for the `workers` property for a vhost that accurately represents the expected number of requests we'll need to process in a given time window, and therefore the number of Apache child processes that will be running at a given time, and
2. a good estimate on the amount of memory required by each of those Apache child processes,

then, we should be able to obtain a reasonable estimate on the memory "size" of a vhost by simple multiplication. [4] If we additionally had an estimate on the average amount of time or CPU resources it took to process a request, we could also obtain an estimate on the CPU "size" of a vhost.

To address item 1 we could perform log analysis for each vhost (excluding monitoring requests) to determine for some time window (to be determined) the number of concurrent requests received. This should give us an estimate on the number of active children required in that window. We could then either take the average number of concurrent requests or some other distribution in order to obtain a good value for the `workers` property of a vhost.

One issue with this approach is that we do not know what the best time window is to use for evaluating "concurrent" requests. One possible approach is to look at the rate at which Apache children are reclaimed after being idle.

Another issue is that it's not clear on what historical time scale we should be processing logs. For instance, processing all logs in the past year would given us better stability when considering periodic spikes, but would not react as quickly to a recent general uptake in activity for a vhost. Furthermore, it would be expensive to compute without way to track a sort of moving average since there are roughly 2 GB of logs per day and they would need to be decompressed first. [5] On the other hand, a single day's worth of logs might be to fine to get a good understanding of the long term behavior of a vhost.

To address item 2 accurately, we would really need to perform load testing on each vhost independently. Since this is not practical, we could possibly obtain an upper bound by loading a vhost with as many extensions and plugins as possible (ie: turn all the available settings on) and load test it to get a reasonable guess at how much memory a single worker could possibly use. Or, we could estimate some middle ground between the defaults (minimum) and the maximum. In either case it is important to accurately measure the actual amount of resident unshared memory for the process.

For vhost request CPU resource estimates we can again use log analysis of the average request response times. However, this number unfortunately encompasses extra overhead such as external I/O calls during which a worker may be preempted so it does not accurately reflect actual CPU resource demands.

1.4 Vhost to VM Assignment and Infrastructure Sizing

Assuming we had a reasonable estimate on the "size" of each vhost, then we should be able to answer the question of how best to size the number of backend VMs for a particular type in our infrastructure by applying a bin packing algorithm [6] to each vhost type, where the vhosts are the balls and the VMs are the fixed sized bins, the number of which we need to use we want to minimize.

We would like our solution to be done in a stable and well balanced way so that we do not disrupt service too much by unnecessarily moving vhosts to different

[4] At least, that mechanism works for prefork Apaches. For threaded Apaches we also need to divide by a factor of 25 first, since in that case a new full process is not forked until all of its threads have been claimed.

[5] *What is a good mechanism for that?*

[6] https://en.wikipedia.org/wiki/Bin_packing_problem

VMs even when their "sizes" might change slightly overtime and so that the heaviest vhosts are not generally pitted together as that might impact our ability to handle of requests to a particular vhost gracefully.

Additionally, a vhost with a "size" larger than the capacity (ie: RAM, CPU) of a single backend of that type would be a good indication of when we need to split that vhost across multiple secondary backends. This could be equivalent, for instance, to replacing a ball of size n with 2 of size $\frac{n}{2}$, and then running our bin packing algorithm.

Finally, although rebalancing is usually done off peak request hours, adding new vhosts can happen at any time during the day and they are expected to be provisioned and made available within a very short period of time. Therefore, our algorithm should be able to handle both cases well.

2 Deliverables

For this project, we'd like to start with a survey of single dimensional bin packing algorithms, such as the first fit approximation algorithm, since in our case we're principally concerned with memory at the moment, and in particular online versions, in order to handle the arrival of new vhosts (balls) to our VMs (bins).

We'd like to be able to present a possibly adapted algorithm as a solution that meets the criteria presented in the previous section. We'd also like to understand, perhaps through some bounds, how balanced and how stable the different solutions provided are.

In order to be able to analyze that we could use a measure of the number of vhosts assigned to a particular VM and the deviation from that average as the balance. Stability, on the other hand, could be defined as the number of vhosts that need to change to a different VM when a new vhost is added to the system.

To start with this research, we've obtained or placed holds on the material cited in the References section.

If time allows, understanding how to extend this to a two dimensional bin packing scheme (ie: RAM and CPU) would be interesting.

Ultimately, we would like to be able to implement this, but again only if sufficient time allows. In that case, an empirical comparison of the current balancing scheme vs. the bin packing one would be desirable. Some resulting system metrics to consider could possibly be the total RAM requirements (ie: sum of the bin capacities) and any measurable differences in vhost response time.

References

[1] S. Albers and M. Mitzenmacher. Average-case analyses of first fit and random fit bin packing. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 290–299. Society for Industrial and Applied Mathematics, 1998.

[2] A. Borodin. *Online computation and competitive analysis*. Cambridge University Press, Cambridge, U.K. New York, 1998.

[3] E. Coffman Jr, C. Courcoubetis, M. Garey, D. Johnson, P. Shor, R. Weber, and M. Yannakakis. Bin packing with discrete item sizes, part i: Perfect packing theorems and the average case behavior of optimal packings. *SIAM Journal on Discrete Mathematics*, 13(3):384–402, 2000.

[4] C. Courcoubetis and R. Weber. Stability of online bin packing with random arrivals and long-run-average constraints. *Probability in the Engineering and Informational Sciences*, 4(4):447–460, 1990.

[5] A. Federgruen and G. van Ryzin. Probabilistic analysis of a generalized bin packing problem and applications. *Operations research*, 45(4):596–609, 1997.

[6] C. Lee and D.-T. Lee. A simple on-line bin-packing algorithm. *Journal of the ACM (JACM)*, 32(3):562–572, 1985.

[7] S. Martello. *Knapsack problems : algorithms and computer implementations*. J. Wiley & Sons, Chichester New York, 1990.