The Robin and The Wasp

By Jordan Amos

For Tony White

Introduction

In todays modern web systems are becoming increasingly distributed. With the advent of AWS, Rackspace, Heroku, Google app engine and more, we run web servers on many machines and rely on a proxy service to route a web request to the appropriate machine. Because these load balancers are designed for previous paradigms where machines were purchased and owned, we would traditionally want to equally balance all requests across multiple machines. In the new age of distributed instances of computation, we pay by how many servers we use and thus want to make the most of the individual machines.

Background and Related Work

In the past, since the main goal of a load balancer was to equally distribute the computation, the industry standard became a round robin approach. All of the machines are stored in a FIFO queue. When a request comes in the first server is selected to handle the load and then moved to the end of the queue. This is a simple and effective way to ensure that the load will be equally distributed among all servers in the queue.

Caching is something that has made the web able to scale as well as it has. It is expensive for the server to have to recreate web pages from templating engines every time a request is made. Modern servers can cache that information into memory and send it to a requesting client more easily than if it had to make disk or database reads, and render the html.

Polish wasps are all capable of performing all of the same tasks from birth, but they will also tend to specialize in specific tasks as they perform them. They do this by having a resistance to perform any given task inherently. When a stimulus enters the environment they will perform the task using a bayesian probability based on the amount of stimulus and their resistance to performing the task. If they perform the task, then the resistance to that job will decrease. At every time, all resistance is increased. This is what will allow them to specialize.

Problem Statement

I believe that by using a polish wasp algorithm in a load balancer, we can leverage the benefits of server caching to improve performance of web servers, with emphasis on static sites. To test this hypothesis there are three layers that will need to work in unison.

The first layer is the static http file server. For this layer, the open-source project, node-static seemed to have everything needed. The next step to this layer is files to serve. I create a single simple html page with a body and slightly over 1000 lines of lorem ipsum text. From this I ran a script to copy the file, named page1.html to pageX.html where X is 2 to 25. With that I had a static web server that could serve 25 pages. I would need to launch 7 of these at once so I created a script to do that.

The second layer is the actual load balancer. Since most of the technology already exists in open-source projects, I decided to use one that had an easily accessible routing functionality. The project http-proxy seemed like a good fit and they even had a ready made example with the round robin functionality implemented. By using this project and the example I am able to run a round robin load balancer. I copied the example and implemented the wasp algorithm to achieve the desired results for the new Polish Wasp Load Balancer. I would now have two different load balancers that I could test, using the same code with the exception of the routing algorithm.

The third layer is the test layer. For the testing and benchmarking of the load balancer I decided to use wrk. This is another open-source project that allows for you to test web applications. It measures requests and size and performs nice summary statistics for every test. The issue with the program is that you can only test one page request at a time. To work around this I created a script that will launch a number of simultaneous tests up to a total number of tests for a randomly selected page. The other thing used to monitor processes is top. Unfortunately I had no way of storing this information, but it was interesting trends anecdotally.

Results, Analysis and Discussion

The test was run on a single machine, using multiple processes with all other programs except the ones needed closed, I tried to minimize external actions from the programs themselves, but there is some room for dispute in my methodology there. I would like to run further tests using multiple machines, vms and other constraints to ensure the integrity of the data. Because of the way that the *wrk* program runs, there is an issue that arises because only 8 different pages are being called at one time. This means that there's a better than normal chance that any given server has the information cached. The results of this test were largely inconclusive, though they provide some interesting insights. The results of the data can be found in the references section.

Anecdotally, when running the tests the input of the *top* program the round robin was very effective at distributing the load between all of the running processes. The polish wasp algorithm would tend to have a more concentrated distribution, and even have some processes essentially idle. There are unimplemented provisions to have the wasps die if they are inactive and start when active, and there's no reason this can't be done with this toolset. To optimize the number of running instances.

The first thing to notice is the requests per second. On average, there isn't a significant difference. The round robin averaged 38.0054 requests per second and the polish wasp averaged 38.8836 requests per second. What is more interesting to me though is the trend that appears to be happening when we graph the results (See Apendix 1). There appears to be the beginning of a trend, where the the polish wasp algorithm will outperform the round robin as the servers become more specialized. The average if we exclude the first 18 tests, the average becomes 39.3934375 for round robin and 41.210625 for the wasps. That is a 4.4 % improvement in terms of requests per second. There is also the issue that because it's repeating the same 8 requests at any given time that the advantage for the wasps only lasts a short period during the start of the individual tests. I believe that this bias means that the average should be higher. There is a definite need for improved testing methodology.

The next thing that we were measuring was latency. The results of the test break down as follows. I have taken the averages run across all of the tests for the individual resuts in the title.

	Average Latency (ms)	Std deviation (ms)	Max (ms)
Round Robin	856.0208	217.11	1430.908
Polish Wasp	933.2478	235.3142	1518.7996

It becomes evident that the wasp algorithm doesn't do as well in terms of latency. I would have to guess that this is mostly due to the stochastic nature of the algorithm. It has a probability that any request will be stuck forever inside a while loop. We use the law of large numbers to our advantage, but the cost of that shows up here. There might be a way to mitigate this through some clever tactic to avoid using a indefinite while loop. There's also the question of if this is important this is over requests per second. It's also possible that the change is not noticeable enough, that it may justify the boost in requests per second. Though if we are simply taking averages, it is worse by 9%.

The last thing that we are looking at is the average transfer rate per second. The average was 15.6534 MB/s for round robin abd 16.0214 MB/s for the wasp algorithm. The results are better with by using the polish wasp algorithm by 2.2%. I didn't expect this in my results, but I'm guessing that it correlates with the requests per second.

Conclusions

While the results aren't conclusive there is definitely enough to conclude that the performance detriment of the more complex polish wasp algorithm, may be made up through specialization. This is inspiring, as the algorithm itself hasn't been implemented in it's entirety.

The main conclusion is that the initial results warrant further, more rigorous testing. There are several changes I would make to the tests for better results. The first thing I would change is the hosting. I ran my tests with 7 servers on one machine. I would like to have

these run independently on distributed vms on a reliable cloud service. I would also like to move the load balancer to a similar machine. I would then work to improve the testing layer. The wrk program doesn't work well for this because it can only access one page. What I would need is a custom solution that can make requests to a randomly selected page. I could either create my own, modify wrk, or find another testing program. I think that modifying wrk would be the best approach. Whatever the testing solution, it should also be run on a vm to ensure consistency between the tests. I would also like to run the wasp server and the robin server with tests running before collecting the data. I believe that we will see more accurate results if we don't measure the training stage of the wasp server.

The final takeaway from this experiment is that even with a rudimentary implementation of the algorithm we are beginning to see improvements from specialization. With a fully featured implementation, with better measurement of stimuli rather than a static unit size, optimized learning and forgetting rates, and the ability to kill and start individual servers we should be able to see better performance gains in terms of request/second without affecting the the latency. Even a 4.4% improvement would be a huge savings to the industry. Google spends 5B dollars per quarter on computation, so that would save them 220 million dollars per quarter. I suspect that this may not translate to sites that are highly dynamic and based on user data. It's also possible that this technology would work well to load balance database requests, especially with caching rdbms solutions. I also see this working well in a hierarchy, even in combination with round robin clusters. The method to scale this is a current unknown.

References

Node static web server:

https://www.npmjs.com/package/node-static
https://github.com/cloudhead/node-static
http://www.sitepoint.com/serving-static-files-with-node-js/

Load Balancer:

https://github.com/nodejitsu/node-http-proxy/blob/master/examples/balancer/simple-balancer.js
https://github.com/nodejitsu/node-http-proxy
https://www.npmjs.com/package/http-proxy
http://goldfirestudios.com/blog/136/Horizontally%C2%ADScaling%C2%ADNode.js%C2%ADand%C2%ADWebSockets%C2%ADwith%C2%ADRedis
http://blog.keithcirkel.co.uk/load-balancing-node-js/

Polish Wasp:

http://sikaman.dyndns.org:8888/courses/4107/handouts/07-DivisionOfLabour.pdf

Testing (wrk) :

https://github.com/wg/wrk
https://github.com/wg/wrk/wiki/Installing-Wrk-on-Linux
http://blog.keithcirkel.co.uk/load-balancing-node-js/

Generating Lorem ipsum:

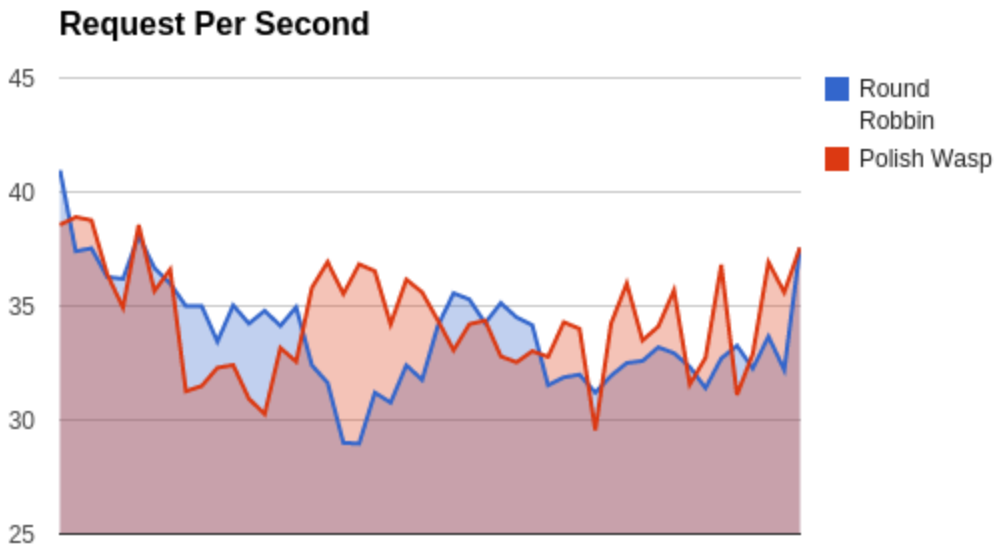
http://www.lipsum.com/

Test Results:

https://docs.google.com/spreadsheets/d/1ac6ZrozDvCPfui9KmqrYnXK75tgRolCNoSADA3JJ7rE/edit?usp=sharing

APENDIX

1.



Individual Tests