CPE 400

Cloud Based Load Balancing

A Visual Representation and Simulation

Benjamin Nicholes
Martin Revilla
Meliton Padilla

Introduction

Load balancers in the cloud such as an Amazon Web Services Load Balancer exist to distribute incoming traffic on a network to servers supporting an application. The load balancer's job is to distribute client loads in such a way that servers do not experience degraded performance caused by load or other factors. A load balancer can use one of many algorithms to route client requests. This project will attempt to simulate and display how these algorithms work. Along with providing a result table that can help show the exact aspects that make certain algorithms better than others, in different use cases. Our team has also created our own load balancing algorithms which are Enhanced Enforced and Enhanced TDMA.

Load balancing

Each algorithm is used with random generators that would help simulate small to large client requests. The six load balancing algorithms chosen and the different parameters followed for each client request are listed below.

Load balancing algorithms

1. Weighted round robin
2. Lowest latency
3. Enhanced TDMA
4. Ratio
5. Least connections
6. Enhanced enforced

Algorithm parameters

- Latency
- CPU
- Region
- Current load
- Number of connections

Ideal use cases

In Round Robin it is assumed that all servers are equal in terms of power and speed, which is not efficient in the distribution of traffic. To effectively apply this to the real world other factors must be accounted for such as its processing power and handling capacity. Weighted round robin was chosen because applying different weights to each server can help distribute the traffic and optimize the efficiency the load balancing can achieve in the network. An example of real world implementation would be for networks set up with old and new servers that need to maximize its efficient workload distribution by taking advantage of those newer servers with more power.

Lowest latency is designed to provide the server with the fastest processing time, accounting for the current workload and number of connected users. Although load balancing requires many factors to be accounted for, lowest latency is able to focus on one factor that is able to improve the throughput of all its servers. An example of its potential usage would be for online game play that requires the fastest throughput, to provide real time gameplay.

Ratio/Weighted Round Robin is designed to load balance according to the weights put onto each server. A server with weight “1” would receive only 1x amount of data payloads as compared to a server with a weight “3” which would receive the 3x the amount of data payloads in a given timeframe. An example for ratio usage would be best for networks that have multiple types of servers with different power.

Least connections used is similar to a round robin distribution but will look at the amount of work being done. A server that is overloaded with jobs/connections will be ignored for the time being to efficiently distribute traffic accordingly. Jobs are ultimately distributed to the servers with the least amount of connections. Just as round robin it is an easy to use algorithm that is ideal for basic improvement in server utilization.

Enhanced TDMA is best to be ideally used by a company that hosts several international servers that are on different time zones. Server A and Server B might be used in Timeframe 1, and data payloads are exclusively sent to these servers on this time frame. Furthermore, data payloads are only sent to the server with lesser amount of CPU utilization for optimization of resources.

Enhanced Enforced can be best used for massive file transfers that are also time sensitive. A company might be data mining and they are sending this to their processing server in another region. Enhanced Enforced load balances the payload by directly sending it to the region if and only if the CPU utilization and latency of the intermediate server in the other region is within a certain latency “window” and cpu “window.”

Error Handling Scenario: If a server would go down, the algorithm would simply take out the server out of rotation, thus continuing as if it doesn’t exist. Different cases for each algorithm can be defined, for example how to utilize the remaining resources by changing the previous weights or adjusting the time frame for Enhanced TDMA.

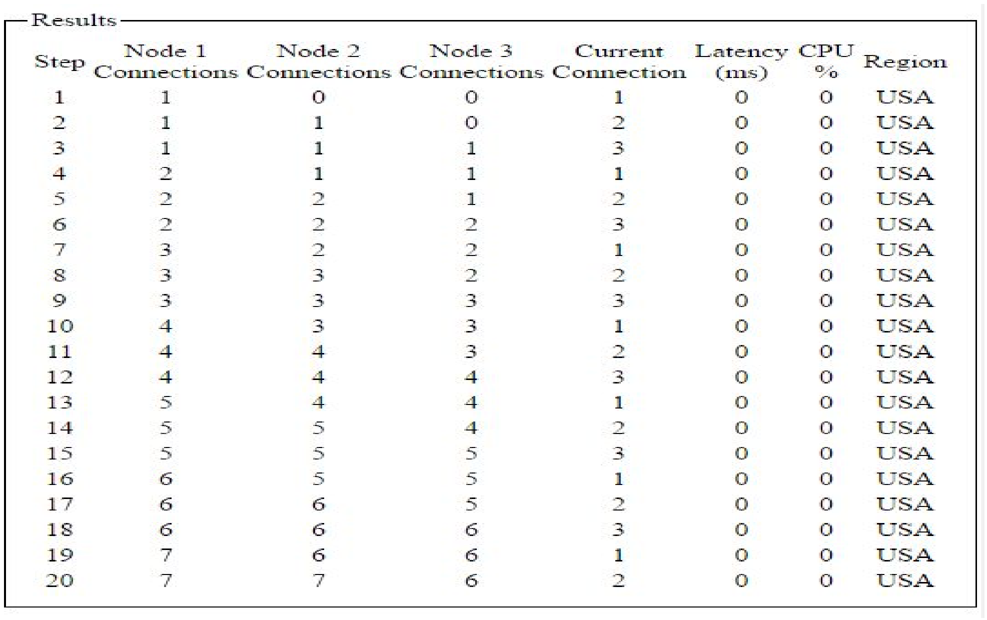
Contribution

A couple of new algorithms for load balancing were created as our novel contribution. Enhanced TDMA and Enhanced Enforced were developed to help demonstrate the difference in efficiency that load balancing can produce when considering different factors. Enhanced TDMA is able to account for the CPU usage each server is using based on comparing different regions on each time frame. Data payloads are only sent on the servers that are chosen to be used in a certain timeframe and are also only sent to the servers with the least CPU utilization. TDMA in practice gives each “sender” equal amounts of time to send, but with Enhanced TDMA, “senders” (servers in this case) are further subdivided, and transmission is only given to those with lowest CPU utilization.Enhanced Enforced takes the already existing “enforced” load balancing algorithm (which just restricts load balancing to a specific server) by enhancing it by accounting for CPU utilization, and latency. Load balancing of a data payload will only occur if the prefered server satisfies the latency window and CPU utilization window of their choice. In other words, a payload will only be sent if the current latency and current cpu utilization of the server is within a certain window, i.e: Within 50ms or within 50% of CPU use.

Results

Round Robin: As seen in the table 1, the load distribution is sent to each node incrementally. This is expected as all nodes take turns receiving the payloads. Even if by the end of the 20 iterations, node 1 would have received more data, the iterative behavior would continue past 20 iterations in a real scenario.

Table 1: Round Robin results

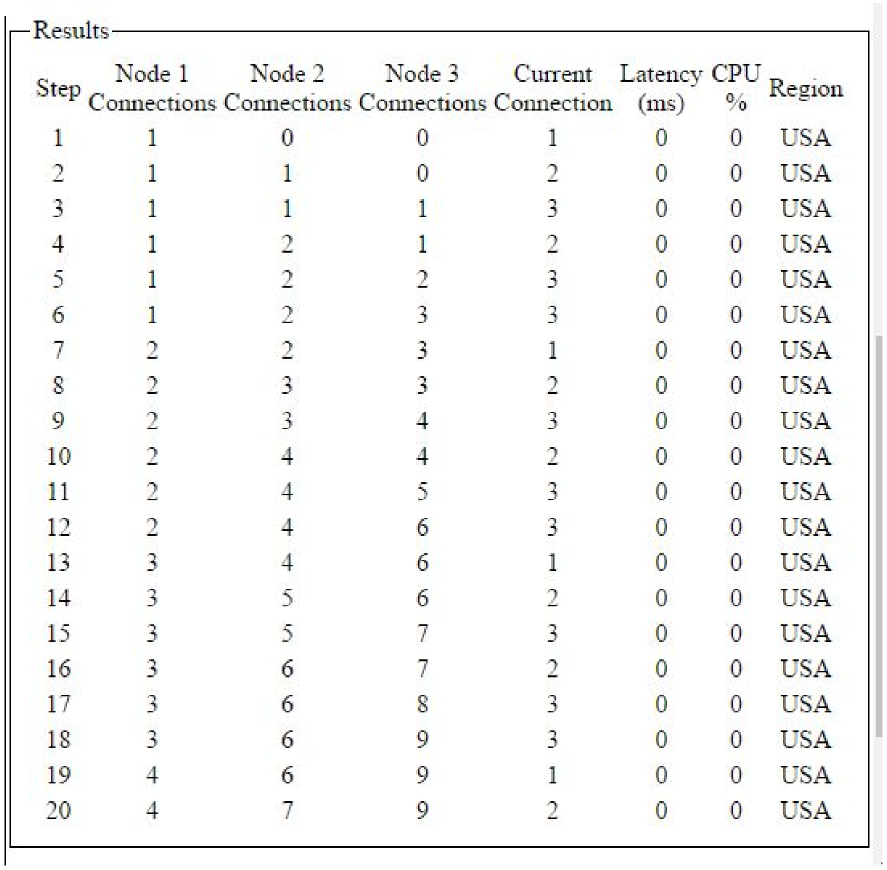


Results

Step	Node 1 Connections	Node 2 Connections	Node 3 Connections	Current Connection	Latency (ms)	CPU %	Region
1	1	0	0	1	0	0	USA
2	1	1	0	2	0	0	USA
3	1	1	1	3	0	0	USA
4	2	1	1	1	0	0	USA
5	2	2	1	2	0	0	USA
6	2	2	2	3	0	0	USA
7	3	2	2	1	0	0	USA
8	3	3	2	2	0	0	USA
9	3	3	3	3	0	0	USA
10	4	3	3	1	0	0	USA
11	4	4	3	2	0	0	USA
12	4	4	4	3	0	0	USA
13	5	4	4	1	0	0	USA
14	5	5	4	2	0	0	USA
15	5	5	5	3	0	0	USA
16	6	5	5	1	0	0	USA
17	6	6	5	2	0	0	USA
18	6	6	6	3	0	0	USA
19	7	6	6	1	0	0	USA
20	7	7	6	2	0	0	USA

Weighted Round Robin: In the simulation, by 20 iterations, it is shown at the end that node 1 received 4 connections, node 2 received 7, and node 3 received 9. This is expected since node 1 had a 1x weight, node 2 has a 2x weight, and node 3 had 3x weight. The goal of this algorithm is to have load distribution be different and dependant on the weight of the nodes.

Table 2: Weight Round Robin results

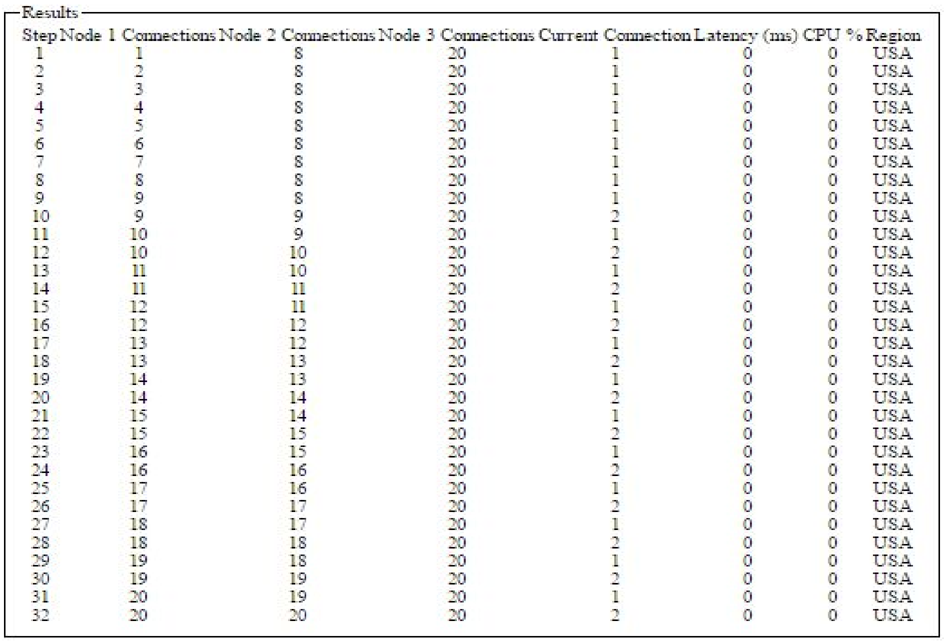


Results

Step	Node 1 Connections	Node 2 Connections	Node 3 Connections	Current Connection	Latency (ms)	CPU %	Region
1	1	0	0	1	0	0	USA
2	1	1	0	2	0	0	USA
3	1	1	1	3	0	0	USA
4	1	2	1	2	0	0	USA
5	1	2	2	3	0	0	USA
6	1	2	3	3	0	0	USA
7	2	2	3	1	0	0	USA
8	2	3	3	2	0	0	USA
9	2	3	4	3	0	0	USA
10	2	4	4	2	0	0	USA
11	2	4	5	3	0	0	USA
12	2	4	6	3	0	0	USA
13	3	4	6	1	0	0	USA
14	3	5	6	2	0	0	USA
15	3	5	7	3	0	0	USA
16	3	6	7	2	0	0	USA
17	3	6	8	3	0	0	USA
18	3	6	9	3	0	0	USA
19	4	6	9	1	0	0	USA
20	4	7	9	2	0	0	USA

Least Connections: At step 1, the initial simulation had node 1 with 1 connection, node 2 with 8 connections, and node 3 with 20 connections. The load balancing algorithm would then only send to node 1 since it had the least amount of connections at the start. It would do this until step 8 in which both node 1 and node 2 both have 8 connections. It would then become round robin in nature between node 1 and node 2 due to them having the same amount of connections. Node 3 hasn't been increased with connections because it is at 20. It would only receive more connections once node 1 and node 2 reach 20 connections as well. A round robin behavior was then seen and expected.

Table 3: Least Connections results

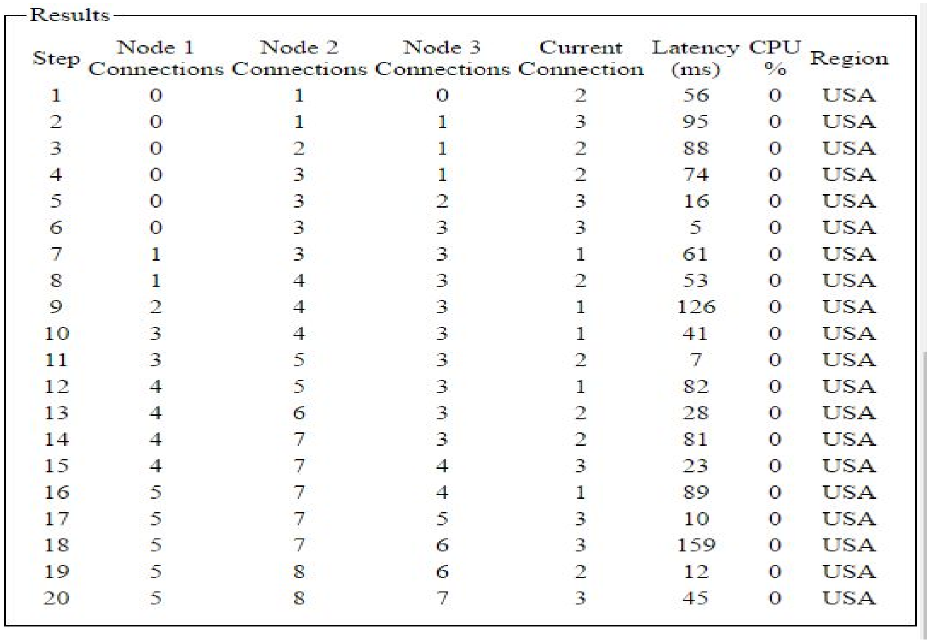


Results

Step	Node 1 Connections	Node 2 Connections	Node 3 Connections	Current Connection	Latency (ms)	CPU %	Region
1	1	8	20	1	0	0	USA
2	2	8	20	1	0	0	USA
3	3	8	20	1	0	0	USA
4	4	8	20	1	0	0	USA
5	5	8	20	1	0	0	USA
6	6	8	20	1	0	0	USA
7	7	8	20	1	0	0	USA
8	8	8	20	1	0	0	USA
9	9	8	20	1	0	0	USA
10	9	9	20	2	0	0	USA
11	10	9	20	1	0	0	USA
12	10	10	20	2	0	0	USA
13	11	10	20	1	0	0	USA
14	11	11	20	2	0	0	USA
15	12	11	20	1	0	0	USA
16	12	12	20	2	0	0	USA
17	13	12	20	1	0	0	USA
18	13	13	20	2	0	0	USA
19	14	13	20	1	0	0	USA
20	14	14	20	2	0	0	USA
21	15	14	20	1	0	0	USA
22	15	15	20	2	0	0	USA
23	16	15	20	1	0	0	USA
24	16	16	20	2	0	0	USA
25	17	16	20	1	0	0	USA
26	17	17	20	2	0	0	USA
27	18	17	20	1	0	0	USA
28	18	18	20	2	0	0	USA
29	19	18	20	1	0	0	USA
30	19	19	20	2	0	0	USA
31	20	19	20	1	0	0	USA
32	20	20	20	2	0	0	USA

Lowest Latency: The results of this simulation is more unpredictable since the latencies of each connection is randomized in each iteration. The node with the least latency is then chosen as that one that will gain a new connection. As a result, the connections of each node is less predictable. In a real life scenario these latencies would most likely be more consistent and would change due to high traffic, disconnections, or other errors.

Table 4: Lowest Latency results

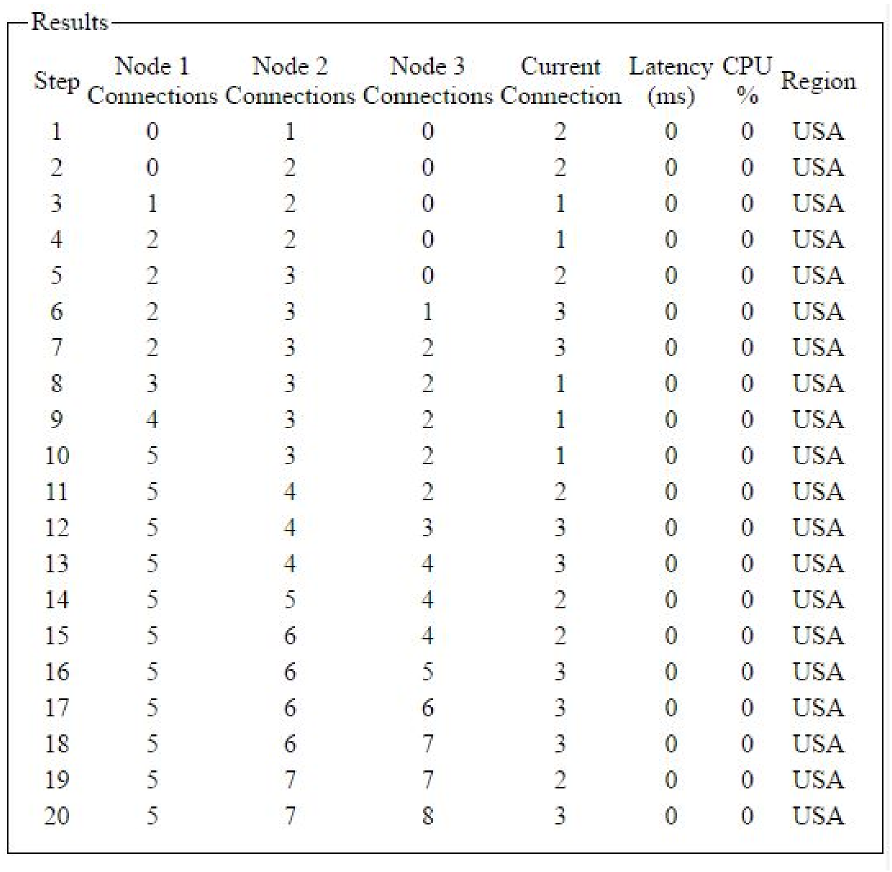


Results

Step	Node 1 Connections	Node 2 Connections	Node 3 Connections	Current Connection	Latency (ms)	CPU %	Region
1	0	1	0	2	56	0	USA
2	0	1	1	3	95	0	USA
3	0	2	1	2	88	0	USA
4	0	3	1	2	74	0	USA
5	0	3	2	3	16	0	USA
6	0	3	3	3	5	0	USA
7	1	3	3	1	61	0	USA
8	1	4	3	2	53	0	USA
9	2	4	3	1	126	0	USA
10	3	4	3	1	41	0	USA
11	3	5	3	2	7	0	USA
12	4	5	3	1	82	0	USA
13	4	6	3	2	28	0	USA
14	4	7	3	2	81	0	USA
15	4	7	4	3	23	0	USA
16	5	7	4	1	89	0	USA
17	5	7	5	3	10	0	USA
18	5	7	6	3	159	0	USA
19	5	8	6	2	12	0	USA
20	5	8	7	3	45	0	USA

Enhanced TDMA: The results of this simulation is dependant on the time frame which consists of 5 iterations. Each time frame randomly chooses two nodes in which loads are distributed. The node with the lesser amount of CPU utilization is chosen as the node to gain a connection. The results of this specific simulation is also dependent on randomization due to CPU utilization being also randomized on each iteration. On a real life scenario, these CPU utilizations would probably be more gradual in change, rather than completely being different on each iteration.

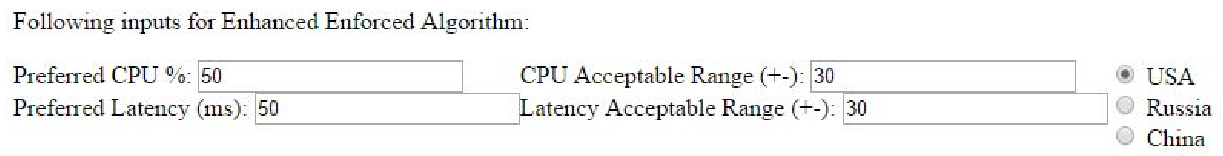
Table 5: Enhanced TDMA results



Results

Step	Node 1 Connections	Node 2 Connections	Node 3 Connections	Current Connection	Latency (ms)	CPU %	Region
1	0	1	0	2	0	0	USA
2	0	2	0	2	0	0	USA
3	1	2	0	1	0	0	USA
4	2	2	0	1	0	0	USA
5	2	3	0	2	0	0	USA
6	2	3	1	3	0	0	USA
7	2	3	2	3	0	0	USA
8	3	3	2	1	0	0	USA
9	4	3	2	1	0	0	USA
10	5	3	2	1	0	0	USA
11	5	4	2	2	0	0	USA
12	5	4	3	3	0	0	USA
13	5	4	4	3	0	0	USA
14	5	5	4	2	0	0	USA
15	5	6	4	2	0	0	USA
16	5	6	5	3	0	0	USA
17	5	6	6	3	0	0	USA
18	5	6	7	3	0	0	USA
19	5	7	7	2	0	0	USA
20	5	7	8	3	0	0	USA

Enhanced Enforced: In our simulation, the user has to input values for preferred CPU utilization, preferred latency, preferred region (USA, Russia, China), latency window and CPU window. Connections are then only increased in two nodes as a result, the preferred region, and a default node.The default node acts as the default destination of a data payload if the user's criteria are not met. As a result, it is expected for example, for Node 3/China to end up with 0 connections by 20 iterations, if the user chose Node 2/Russia as their preferred node, and as a result, Node 1/USA as the default node.



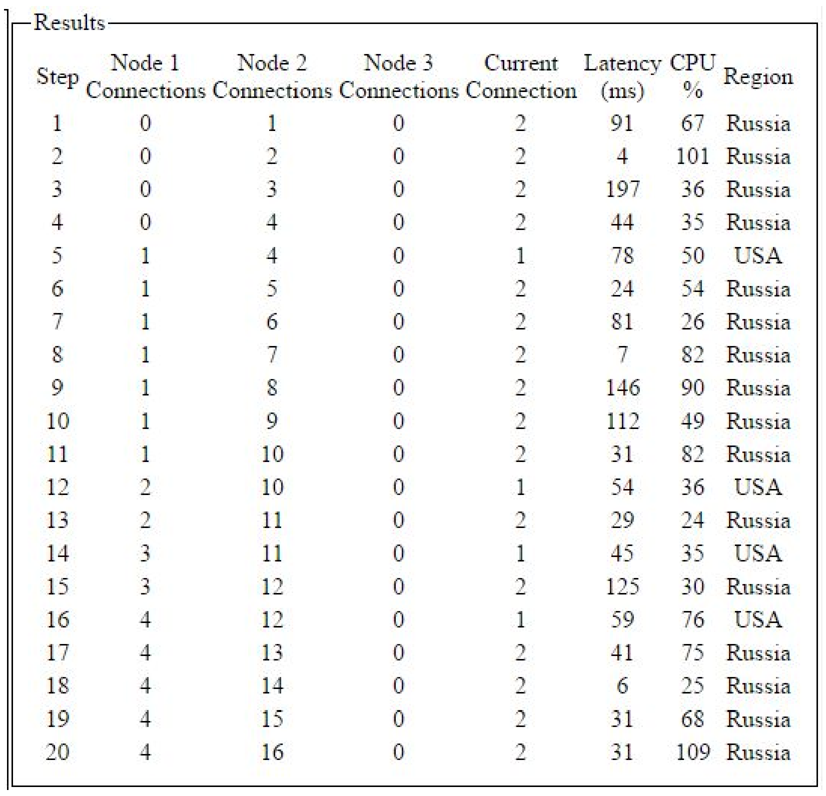
Following inputs for Enhanced Enforced Algorithm:

Preferred CPU %: 50
CPU Acceptable Range (+-): 30
Preferred Latency (ms): 50
Latency Acceptable Range (+-): 30

[✓ USA] Russia China

Fig 1: Enhanced Enforced user inputs

Table 6: Enhanced Enforced results



Results

Step	Node 1 Connections	Node 2 Connections	Node 3 Connections	Current Connection	Latency (ms)	CPU %	Region
1	0	1	0	2	91	67	Russia
2	0	2	0	2	4	101	Russia
3	0	3	0	2	197	36	Russia
4	0	4	0	2	44	35	Russia
5	1	4	0	1	78	50	USA
6	1	5	0	2	24	54	Russia
7	1	6	0	2	81	26	Russia
8	1	7	0	2	7	82	Russia
9	1	8	0	2	146	90	Russia
10	1	9	0	2	112	49	Russia
11	1	10	0	2	31	82	Russia
12	2	10	0	1	54	36	USA
13	2	11	0	2	29	24	Russia
14	3	11	0	1	45	35	USA
15	3	12	0	2	125	30	Russia
16	4	12	0	1	59	76	USA
17	4	13	0	2	41	75	Russia
18	4	14	0	2	6	25	Russia
19	4	15	0	2	31	68	Russia
20	4	16	0	2	31	109	Russia

Code Explanation
To effectively demonstrate the process of distributing workloads across multiple servers, Javascript, cytoscape.js, canvas and HTML are used to display these different algorithms. Two different versions were created that show the same algorithms and tables but present a different visualization. In order to run the simulation the user would only need to open up the index.html file and the simulation will open up. The live demo will be hosted on http://bennicholes.github.io/index.html, as an option to just load the entire project without accessing each file.

Once the simulation is loaded the user will be defaulted to the canvas simulation and the two main buttons at the top will allow the transition between the canvas and cytoscape simulation. In the canvas version the user will have access to buttons displayed above the simulation screen that name all the load balancing algorithms. Once you select the algorithm to run it will simulate and produce the table needed. After the simulation is done, the user must hit reset in order to run another. Enhanced enforced is the only algorithm that accepts user input. The user input fields are displayed at the center and provide instructions on how to provide the input, changes in the input will reflect on the output table.

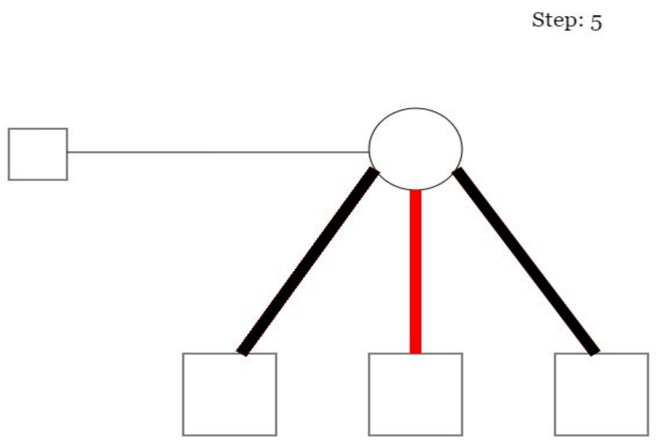


Fig 2: Canvas simulation graph

If the user switches over to the cytoscape simulation by clicking the cytoscape button under the title on the index page, the same steps apply: select the desired algorithm and click on the client request node to produce the table and run simulation. No reset is required, selecting a new algorithm will change the entire graph/table. The cytoscape graph is interactive and can zoom in or out as desired, if any issue with the display occurs simply refresh the page to realign the cytoscape graph.Both simulations have been tested in Chrome, Firefox and IE. If any issues arise for the browsers chrome has been the most tested browser and should be used for demonstrations.Cytoscape.js is an open source graph theory library written in JS and used on Node.js to create visual graphs. The Load Balancing node is the main source that will provide the other servers with the assigned client request. Each edge attached to the load balancing node will turn blue once a server is selected for that request.

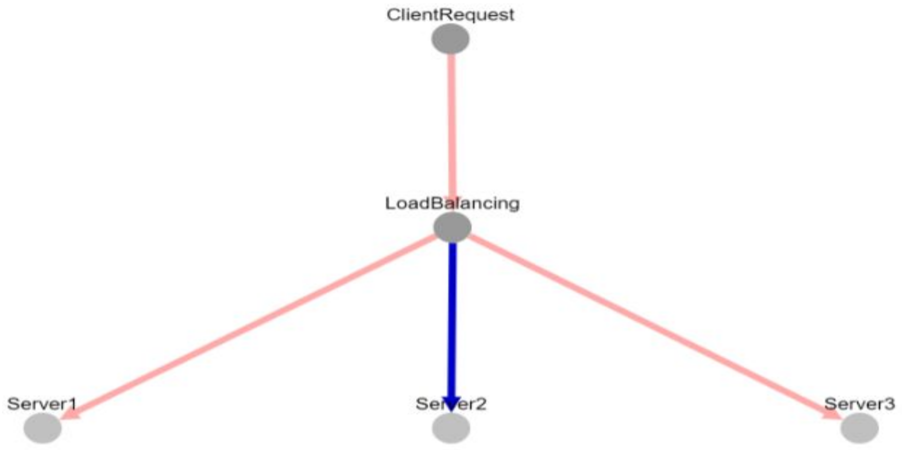
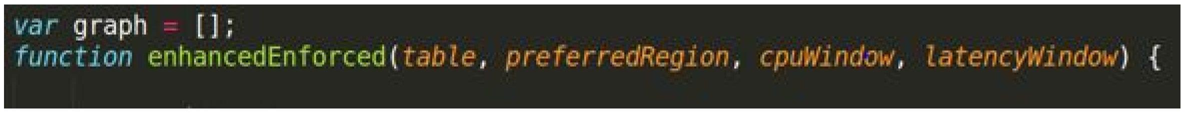


Fig 3: Cytoscape simulation graph

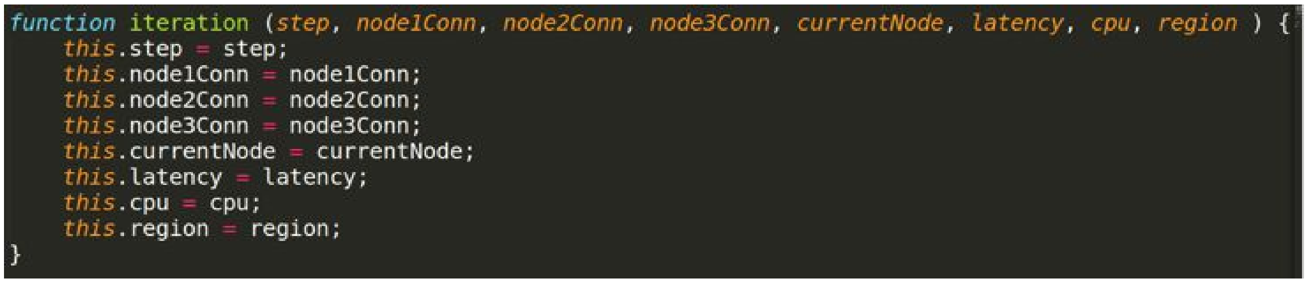
Simulation.js contains all of the load balancing algorithms to run the simulation. Each load balancing algorithm shown in the simulation has its own function with the same name. Each of these functions return a table array which contains “Iteration” objects. Iteration objects describe data that are occurring in each step that generate the table seen in the simulations.



```
var graph = [];
function enhancedEnforced(table, preferredRegion, cpuWindow, latencyWindow) {
```

For the function enhancedEnforced, four parameters are used.These four parameters are needed because the Enhanced Enforced simulation needs direct user input. preferredRegion is the variable that contains either USA, Russia, China represented as numbers. preferredCPU and preferredLatency contains the values which are compared to the latencyWindow and cpuWindow variables. If the “preferred” values are within the “window” of the window variables, the node is chosen. This is accomplished by following the simple formula: absolute value(preferred - actual) < window

An iteration object has the following properties:



```
function iteration (step, node1Conn, node2Conn, node3Conn, currentNode, latency, cpu, region ) {
    this.step = step;
    this.node1Conn = node1Conn;
    this.node2Conn = node2Conn;
    this.node3Conn = node3Conn;
    this.currentNode = currentNode;
    this.latency = latency;
    this.cpu = cpu;
    this.region = region;
}
```

Explanation of the properties:

step - describes the current iteration step.

node1Conn - describes the current amount of connections node1 has at the current iteration step.

node2Conn - describes the current amount of connections node 2 has at the current iteration step.

node3Conn - describes the current amount of connections node 3 has at the current iteration.

currentNode - describes the node which received the payload at that iteration.

latency - describes the simulated latency between the load balancing computer/switch and the node that is chosen to receive the payload.

cpu - describes the current cpu utilization of the node chosen to receive the payload.

region - describes the location of the node chosen to receive payload.

Conclusion

The main goal of this project was to be able to visualize and implement different types of load balancing algorithms that are for different use cases. Often it is hard to just understand an algorithm by just reading about it. From the simulation/result table provided it will help users understand what different aspects each algorithm is focused on. The simulation also allows the user to model certain components for the enhanced enforced algorithm, that are both based on preference and requirements. Presenting the load balancing concept in a visual way helped define why just one algorithm is not good for all network setups. In the future further algorithms and constraints could be implemented, such as DNS failover or reading output logs from servers under load. This is all important as understanding how load balancing works can help network administrators decide what load balancing technique would best solve their problem. Simulating and visualizing load balancing techniques for the cloud is necessary to fully understand how servers and networks can handle heavy loads.

References

https://www.nginx.com/resources/glossary/round-robin-load-balancing/
http://www.jscape.com/blog/load-balancing-algorithms
https://aws.amazon.com/elasticloadbalancing/
http://www.peplink.com/technology/load-balancing-algorithms/
https://support.f5.com/csp/#/article/K6406
https://en.wikipedia.org/wiki/Time_division_multiple_access
Libraries Used
http://js.cytoscape.org/
https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API