Scheduled load balanced web application for heavy traffic sites using containerization

Aishwarya Shinde[1], Vaishnavi Kadam[2], Arun Bairagi[3], Vijay Annam[4]

[1,2,3,4] *Student, Dept. of Computer Engineering, JSPM's Imperial College of Engg. and Research, Pune, India*

***

Abstract - Load balancing technique is used to distribute the work equally into the number of nodes. This is used to allocate the workload to different server and provides fast service. The main goal is to balance and schedule the load equality among the nodes suit such that no nodes will be on or under the situation. Load balancing is used to do proper utilization of resources and better user satisfaction. For the heavy traffic on sites or server we have to use the containerization. Containerization method is an OS level virtualization used to display and run different application without initiating an entire virtual machine for each app. It means that when you want to run the some application on another machine then we have to install all the packages and libraries on that machine. To avoid this problem we can use containerization, In one container we can install all the packages and libraries which are needed for that application and build OS level virtualization using Docker by creating images of container. So we can use this image on any machine without any additional resources or equipment's. Containerization works with micro services and distributed application, as each container operates to identify other host and uses minimal resource from the host. This paper contains different methods and algorithm to resolve the complication of load balancing and task scheduling.

***Key Words*:** Containerization, Docker, Horizontal Scaling, Micro-Services.

1.INTRODUCTION

Load balancing is a computer networking method to balance the load on multiple computers or multiple server instances. Load balancing offers a way to make the right use of resources and better user satisfaction. The main objective is to balance and schedule the equality of load between the nodes that no nodes should be on or in the under situation.

Load balancing handles the incoming request from users for information and other services. Load balancing provides a way of achieving the proper utilization of resources and better user satisfaction.

To schedule and balance the load on server, when multiple request are arrives, the server is able to handle the multiple requests and provide same service to the all request. We can solve this problem to schedule the load on server, In that when we know there is load on server then we can up the instances of server which will help to provide the services to all incoming request.

The two main type of methods provided by load balancing are static and dynamic. Containerization is the key technology that allows the simultaneous execution of diverse tasks and micro services over a shared hardware platform.

"Containerization" is a portable operating system level virtualization method which runs and deploy without initiating an entire machine for each app.

It means that when we want to execute same project on difference OS then there is no need to install all package which are required for run the entire project. That's why we use containerization which contains all the required material in it. The most common tool for containerization is Docker, specifically it is the open source engine which is based on globally runtime environment.

2. Proposed System

Our system consist load balancer, multiple instances of server machine and client connected to the balancer, as shown in the fig. load balancer connects multiple instances and communicate with database. There are multiple requests from the multiple users to access the same site at a time then load of site is increased and particular server goes down it means the heavy traffic on the site. Most of time particular sites will be loaded due to multiple requests. So there is load on server and server unable to provide services to all users at a time. To avoid this problem we use load balancer to handle such load by using scaling algorithms, by creating the multiple instances of server machine using Docker and distribute the load on different instances. To distribute the work we use the weighted algorithm using NGINX. In general, all users get served faster due to scaling the instances horizontally for given particular time when there is no need of instances for balancing the load then the instances of server goes down statically. In this way we achieve the proper utilization of resources and better throughput with user satisfaction.

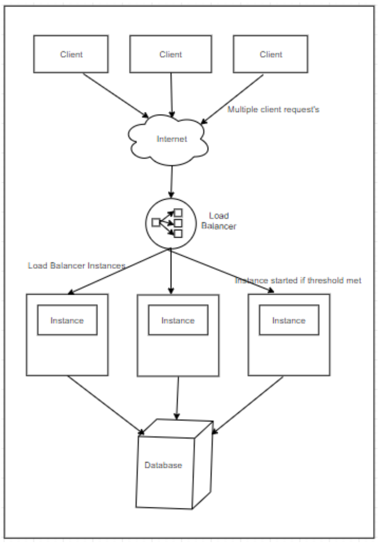


Fig -1: Architecture Diagram

Docker:

Docker is an open platform for developing, transforming, and running applications. Docker enables us to split our applications from our framework so we can deliver software quickly. Docker provides the capability to package and run an application in a generally isolated surrounding called a container. The isolation and security allow run many containers simultaneously on a given host. Containers are lightweight because they don't need the extra load of a hypervisor, but run directly within the host machine's kernel. This means we can run more containers on a given hardware consolidation than if you were using virtual machines. Docker containers run within host machines that are genuinely virtual machines.
Docker provide security to application, security is implemented by means of isolation and encryption. The Docker Engine also control host firewall rules which restricts to access between different networks and which also manage ports for containers. Since all of this is maintained by the Docker Engine itself, it switches dynamically according to tasks, services and networks that are created inside of the cluster.
Docker's portability and lightweight nature also make it easy to dynamically manage workloads, scaling up or split down applications and services as business needs direction, in near real time

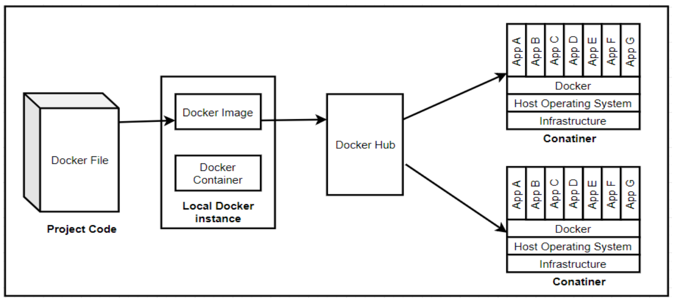


Fig -2: Docker and Containerization

Generally load balancing is used such algorithm to handle the load of the sites which is least connected, weighted load balancing, round robin algorithm.

Round robin algorithm

This algorithm is the scheduling load algorithm. The processor will be distributed into the equally for each process. It means that same time allocated to each process because of that no other process can wait for the processor to complete its execution. Processes are partitioned between all processor in such way that the work load between processer is distributed equally. This algorithm to provide the service at a compile time, each process will be executed with in a particular time slices.

Least connected load balancing

Another load balancing algorithm is least-connected. Least-connected allows controlling the load on application instances more fairly in a situation when some of the requests take longer time to complete. With the least-connected load balancing, NGINX will try not to overload a busy application server with excessive requests, distributing the new requests to a less busy server instead. Least-connected load balancing in NGINX is activated when the least-connected directive is used as part of the server group configuration:

```
upstream myapp1 {
least_conn;
server srv1.example.com;
server srv2.example.com;
server srv3.example.com;
}
```

Weighted load balancing

It is also possible to influence NGINX load balancing algorithms even further by using server weights. In the examples above, the server weights are not configured which means that all specified servers are treated as equally qualified for a particular load balancing method. With the round-robin in particular it also means a more or less equal distribution of requests across the servers are provided there are enough requests, and when the requests are processed in a uniform manner and completed fast enough. When the weight parameter is specified for a server, the weight is accounted as part of the load balancing decision.

```
upstream myapp1 {
server srv1.example.com weight=3;
server srv2.example.com;
server srv3.example.com;
}
```

With this above configuration, every 5 new requests will be distributed across the application instances as following: 3 request will be directed to srv1, one request will go to srv2, and another one to srv3. It is similarly possible to use weights with the least-connected and ip-hash load balancing is the recent version of NGINX.

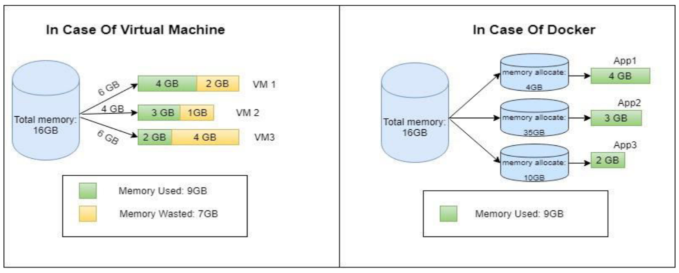


Fig -3: Difference between Virtual machine and Docker

Horizontal Scaling:

Horizontal scaling is to perform the operation horizontally in that we use NGINX, something that is parallel to the horizon. Horizontal scaling affords the ability to scale expanded to deal with traffic. It is the ability to connect multiple hardware or software entities, such as servers, so that they work as a single logical unit.

Horizontal scaling means scaling out. Horizontal scalability can be achieved with the help of clustering, distributed file system, load – balancing.

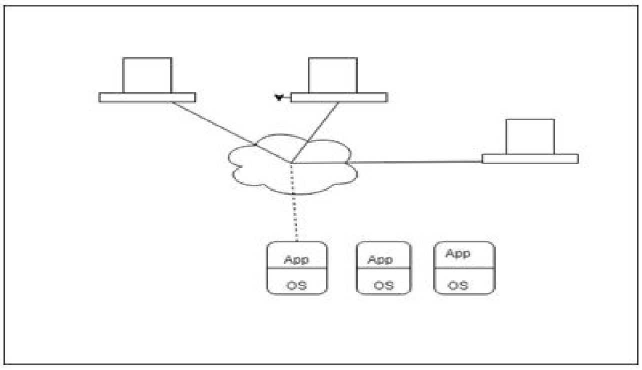


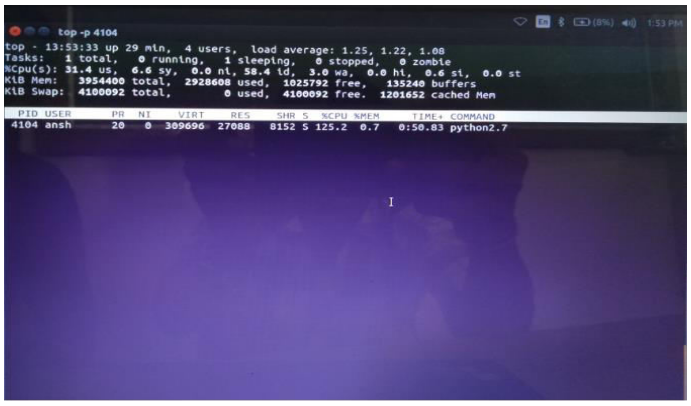
Fig -4: Horizontal scaling

Result:

Below table show the memory uses of existing system and proposed system. CPU is the most important factor to measure the server load. So we used CPU uses to represent the server load. CPU uses of this system measures when the requests are 1000 per second. For existing system and proposed system, CPU utilization of server is unequal. Since the proposed load balancing algorithm can make servers to balance the load almost even among the servers. This yields a better resource scheduling ability to avoid unbalanced load among the servers. Hence, overall resource utilization of the network is improved.

Table -1 Results

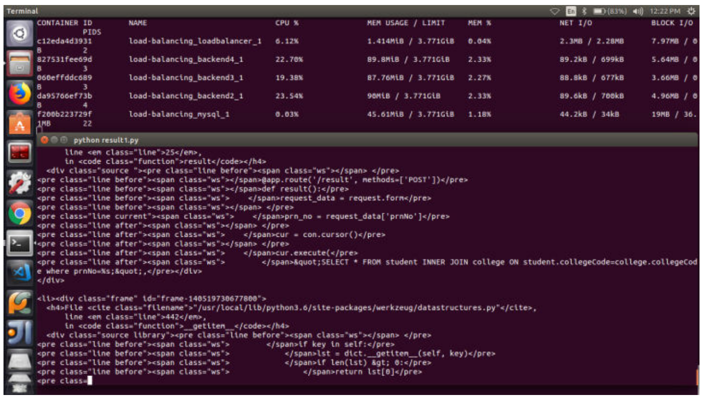
Content	Existing system	Proposed system
CPU uses	31.4%	22.20%
Memory uses	29MB	19MB



```
top -p 4104
top - 13:53:33 up 29 min,  4 users,  load average: 1.25, 1.22, 1.08
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s): 31.4 us,  6.6 sy,  0.0 ni, 58.4 id,  3.0 wa,  0.0 hi,  0.6 si,  0.0 st
KiB Mem:   3954400 total,  2928608 used,  1025792 free,   135240 buffers
KiB Swap:  4100092 total,        0 used,  4100092 free.  1201652 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 4104 ansh      20   0  309696  27088   8152 S 125.2  0.7   0:50.83 python2.7
```

Fig -5: Result with existing system



```
CONTAINER ID        NAME                          CPU %      MEM USAGE / LIMIT      MEM %        NET I/O             BLOCK I/O
          PIDS
c12eda4d3931        load-balancing_loadbalancer_1   6.12%      1.414MiB / 3.771GiB    0.04%        2.3MB / 2.28MB      7.97MB / 0
B         2
827531fee69d        load-balancing_backend4_1       22.70%     89.8MiB / 3.771GiB     2.33%        89.2kB / 699kB      5.64MB / 0
B         3
060effddc689        load-balancing_backend3_1       19.38%     87.76MiB / 3.771GiB    2.27%        88.8kB / 677kB      3.66MB / 0
B         3
da95766ef73b        load-balancing_backend2_1       23.54%     90MiB / 3.771GiB       2.33%        89.6kB / 700kB      4.96MB / 0
B         4
f200b223729f        load-balancing_mysql_1          0.03%      45.61MiB / 3.771GiB    1.18%        44.2kB / 34kB       19MB / 36.
1MB       22
```

```
python result1.py
      line <em class="line">25</em>,
      in <code class="function">result</code></h4>
  <div class="source "><pre class="line before"><span class="ws"></span> </pre>
<pre class="line before"><span class="ws"></span>@app.route('/result', methods=['POST'])</pre>
<pre class="line before"><span class="ws"></span>def result():</pre>
<pre class="line before"><span class="ws">    </span>request_data = request.form</pre>
<pre class="line before"><span class="ws"></span> </pre>
<pre class="line current"><span class="ws">    </span>prn_no = request_data['prnNo']</pre>
<pre class="line after"><span class="ws"></span> </pre>
<pre class="line after"><span class="ws">    </span>cur = con.cursor()</pre>
<pre class="line after"><span class="ws"></span> </pre>
<pre class="line after"><span class="ws">    </span>cur.execute(</pre>
<pre class="line after"><span class="ws">        </span>&quot;SELECT * FROM student INNER JOIN college ON student.collegeCode=college.collegeCod
e where prnNo=%s;&quot;,</pre></div>
</div>

<li><div class="frame" id="frame-140519730677800">
  <h4>File <cite class="filename">"/usr/local/lib/python3.6/site-packages/werkzeug/datastructures.py"</cite>,
      line <em class="line">442</em>,
      in <code class="function">__getitem__</code></h4>
  <div class="source library"><pre class="line before"><span class="ws"></span> </pre>
<pre class="line before"><span class="ws">        </span>if key in self:</pre>
<pre class="line before"><span class="ws">            </span>lst = dict.__getitem__(self, key)</pre>
<pre class="line before"><span class="ws">            </span>if len(lst) &gt; 0:</pre>
<pre class="line before"><span class="ws">                </span>return lst[0]</pre>
<pre class=
```

Fig -6: Result with proposed system

Conclusion:

Load balancing not only handles the load on the server but also provides concurrent access to multiple request and also increase the performance of server. In this paper we proposed a system for balancing and schedule the load equality among the nodes by using Docker and containerization. When there is load on the particular site then to avoid this problem the load is distributed among the multiple instances of the server by using weighted algorithm with the help of NGINX and improve the performance within the time period. If one server fails others are available to ensure. By using Docker we can utilize the CPU and resources.

REFERENCES

1. *S. Wilson Prakash "Server based dynamic load balancing "International Conference on NETACT |20-22 July 2017*
2. Jaimeel M Shah "Load balancing in cloud computing methodological survey on different types of algorithm" ICEI 2017
3. Abhinav Hans, Sheetal Kalra, "Comparative Study of Different Cloud Computing Load Balancing Techniques" IEEE 2014.
4. Reena Panwar, and Bhawna Mallick "Load Balancing in Cloud Computing Using Dynamic Load Management Algorithm" IEEE 2015. P. Sinha, Electronic Health Record. IEEE Press Wiley.
5. J.Prasanna, Ajit jadhav, and V.Neel Narayan "Towards an Analysis of Load Balancing Algorithms to Enhance Efficient Management of Cloud Data Centres," Springer, 2016.
6. Aarti Singha, Dimple Junejab, "Manisha Malhotra "Autonomous Agent Based Load Balancing Algorithm in Cloud Computing" Elsevier, 2015.
7. Tushar Desai, Jignesh Prajapati "A Survey Of Various Load Balancing Techniques And Challenges In Cloud Computing" IJSTR, 2013.