An Auto-Scaling Approach to Load Balance Dynamic Workloads for Cloud Systems

Dr. Sharvani GS,
Associate Professor, Department of Computer Science and Engineering, RV College of Engineering-Bangalore, sharvanigs@rvce.edu.in
Resma KS,
Research Scholar, Department of Computer Science and Engineering, RV College of Engineering-Bangalore,(Research Center Affiliated to VTU-Belagavi), resma.vin@gmail.com

Article History: Received: 11 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 10 May 2021

Abstract
With the advances in modern technology, need for cloud services is ever increasing. In such case on public real-time cloud environments the workload on the respective server can vary depending on the user needs. Cloud Computing is the latest technology that provides on-demand availability of computer resources without direct intervention of the user. This technology combined with cloud load balancing enables enterprises to manage workload demands. The cloud load balancer, with auto scaling features, further increases the efficiency and optimized utilization of the computing resources. This paper discusses how Amazon Web Services as the cloud medium along with MobaXterm for remote control, and implement ARR configurations for the load balancer to route the requests. AWS gives us knowledge about various concepts such as auto-scaling groups, target groups, load balancers, etc.

Keywords–Cloud Computing, Workload, Availability, Load Balancer, Auto Scaling, Target Groups

I. INTRODUCTION

Purpose: With more and more advances being made in cloud computing and its increasing efficiency, companies have started using cloud as their underlying architecture for most of the important operations. The demand for resources is always increasing in these companies and with the help of cloud architecture, all the demand requirements are met easily. Cloud allows them to increase/decrease the load on servers according to their requirements as cloud provides the policy of pay-as-you-go which makes it a good option for the organizations.

Scope: Cloud computing is the on-demand delivery of IT resources via the internet, with pay-as-you-go pricing. Instead of buying, owning and maintaining physical data centers and servers, the companies can leverage this task to third party companies to do the same for them. These third-party companies then provide a virtual environment to the user/company which then the user can use without bothering about the details of the architecture/mechanism. Following are a few reasons why every company should migrate to cloud computing:

a. Cost saving - Companies no longer need to buy, own and maintain their own data centers and servers. They can use the cloud architecture on a pay-as-you-go pricing.

b. Focus on the business - A company does not need to bother about the architecture of the cloud storage or the technical issues related to the data stored in the cloud as it will be maintained by the cloud service provider.

c. Performance - Cloud services are efficient as it provides data to the user irrespective of the geographical location of the user. It also ensures automatic updates for the services and applications.

d. Security - Cloud computing ensures protection against data from any kind of theft possible. It restricts the access of data to only the authorized entities restricting the unauthorized entities to have any information.

e. Flexibility - There are cases when a part of the cloud might go under repair/maintenance or updates. in such situations the other resources still continue to function until the problem is solved.

Problem Statement:

The reliability and system performance in a cloud environment heavily depends on the load balancing strategies used. Usually, the incoming requests for the resources are heterogeneous and highly. This project aims at designing an optimized load balancer which is robust enough to handle such heterogeneous and varying request and provide a better reliability and performance.

Motivation:

Instead of buying, owning and maintaining physical data centers and servers, users can leverage the task of storing data to third party companies. User can access technology services such as computing power, storage and databases on an as-needed basis from a cloud provider such as AWS. Following are a few reasons that has motivated users to shift to cloud:

a. Cloud computing, has provided users with the functionality of Auto Scaling. It is a method where computational resources are scaled and measured in terms of number of servers used based on the load put onto that server. In big companies, most of the workload can't be predicted beforehand which causes a problem. Auto scaling helps in such situations as it allows the companies to scale up/down their resources automatically without prior knowledge of the workload.

b. A load balancer thus helps in scaling out by distributing the workload among an array of servers. It allows the user to add or remove the capacity automatically depending on the workload.

II. LITERATURE REVIEW

The topic of load balancing is vast and applicable to various sizes of networks. A lot of research has been made on this topic and several researchers have written scientific articles and research papers on load balancing. A conceptual model for the future of cloud computing using load balancing algorithms is discussed in [1]. The proposed model is used for efficient resource management, improving the reliability of cloud services and providing the different load balancing algorithms such as static load balancing algorithms (SLBA), dynamic load balancing algorithms (DLBA), and dynamic nature inspired load balancing algorithms (NDLBA). It is concluded that DLBA and NDLBA are more efficient than SLBA.

The analysis of load balancing algorithms using the simulation tool Cloud Analyst has been done in [7]. The researchers have analyzed the performance of round robin algorithm, Equally Spread Current Execution (ESCE) load algorithm and Throttled Load Balancing (TLB) algorithm. It is concluded that TLB performs the best among the three algorithms.

A meta-study has been conducted on load balancing and server consolidation in cloud computing environments in the literature of [3]. It is stated that load balancing with server consolidation enriches the exploitation of resource utilization and can enhance Quality of Service (QoS) metrics, since data centers and their applications are increasing exponentially. Their work attempts to present taxonomy with a new classification for load balancing and server consolidation, such as migration overhead, hardware threshold, network traffic, and reliability.

Furthermore, a detailed analysis of traffic management that occurs in the cloud when a security attack occurs is discussed in detail in [4]. Analytical modelling and evaluation are considered to get QoS measurements under security attacks. The traffic control management is also proposed and considered together with the analytical model. The resulting analytical model is solved using successive over-relaxation (SOR) approach to get QoS measurements.

III. PROPOSED METHODOLOGY

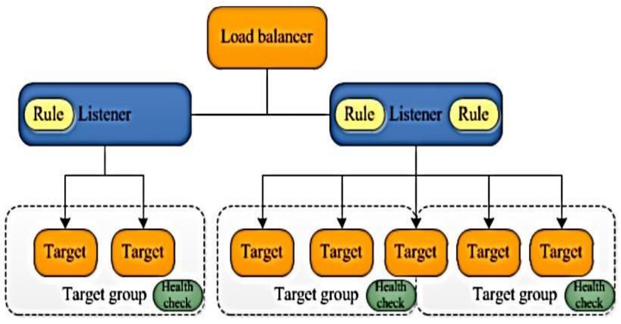


Figure 1: Typical Load Balancer

The following section describes the methodology to be followed. The session provides an overview of system, service or process. It also provides us with the concepts being used throughout the study. Some of the concepts being used will be discussed in the following sections followed by the system architecture and the flow diagram.

An EC2 instance is a virtual server in Amazon Web services which stands for Elastic Compute Cloud. Due to its functionality of on-demand scalable computing capacity, the AWS subscriber can request and provision for a compute server in the cloud. EC2 is the reason why companies can eliminate the need to buy, own and maintain physical data centers and servers. It allows to launch multiple web servers which can be scaled up or down as per use.

It is a template used by the EC2 Auto scaling group to launch EC2 instances. Some of the information specified are the Amazon Machine Image ID, type of instance, key-pair value, etc. This same information is specified in case the user has launched an EC2 instance before.

A target group is specified when creating a listener rule. The main task of the target group is to route the incoming traffic to the registered target. Traffic is only forwarded to the traffic group if a specific condition is satisfied. Multiple types of target groups can be created in order to support different types of incoming requests. For instance, one can create a target group to handle general incoming requests and a separate target group to handle the requests to the micro-services.

An Auto Scaling group maintains a collection of Amazons EC2 instances which are treated as a logical grouping for the purposes of automatic scaling and management. Various functionalities are implemented using auto-scaling groups. These functionalities include health check replacements and scaling policies. Size of the auto scaling group is decided by the number of desired EC2 instances set. Maintaining the number of instances is one of the main core functionalities along with automatic scaling. Using different scaling policies, the user can increase/decrease the number of instances in the Auto Scaling group.

A security group is responsible for controlling the traffic for instances by acting as a virtual firewall. More than one security group can be specified while launching the instance. If the user does not specify the security group, then the default security group is used. Rules can be added to each security group which will allow traffic in the associated instances. When the new rules are added or the old one gets modified, all the instances get automatically updated with the new/modified rules. The user needs to specify a security group while launching an instance in the VPC. The security group can later. Security groups associated with the primary network interface changes if the security group of an instance is changed as it is associated with network interfaces. Different types of load balancers which could be used are:

- Application Load Balancer - For flexible application management.
- Network Load Balancer - For extreme performance and static IP.

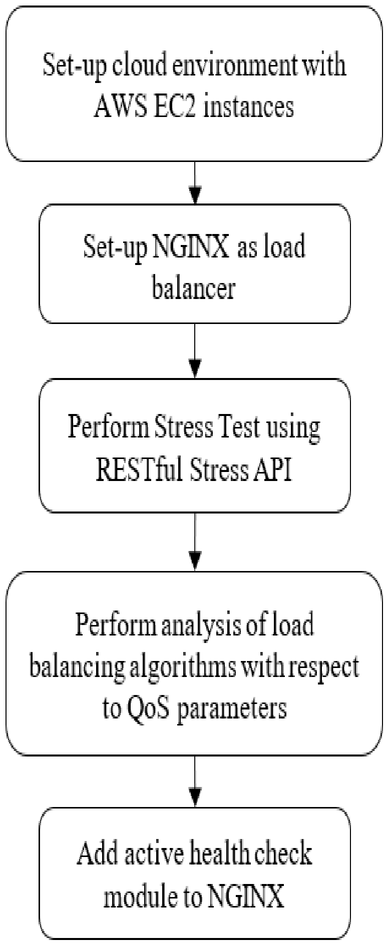
Classic Load Balancer [8] - For existing application that was built within the EC2 - classic network. The load balancer used in the study is the application load balancer (ALB), Figure 1. These load balancers support content-based routing which works well for serverless container-based applications. These load balancers are highly scalable. Figure 1 of Application Load Balancer illustrates the basic components. Each listener in the diagram contains a default rule, and one listener contains another rule which is responsible to route requests to a different target group. Same target group can be registered with multiple target groups as shown in the figure 1. The clients directly contact the application load balancer as it is the single point of contact for them. The incoming application traffic is then distributed across multiple targets. These targets can be multiple EC2 instances that are spread out in multiple availability zones. Adding more than one listener to the load balancer increases the application availability. Using the configured port, the listener keeps a check for any incoming client requests using the configured protocol. How the load balancer routes its requests to the registered targets is determined by the defined rules for the listener. Each rule consists of a priority, one or more actions, and one or more conditions. When the conditions for a rule are met, then the specified actions are performed. A default rule must be defined for each listener after which additional rules can be optionally defined. The requests are then routed to one or more targets registered with the target group using the port number and the specified protocols and port numbers. These targets can be EC2 instances in multiple availability zones. A target can be registered with multiple target groups. Health checks can be configured on a per target group basis and are then performed on all the registered targets.

IV. SYSTEM ARCHITECTURE

As the underlying architecture, the VPC and the region of US East (N. Virginia) is configured. There are 6 availability zones in this region. In every availability region there is 1 single public subnet. An auto scaling groups which will consist of the web server is configured. The auto-scaling group/web-servers is deployed across 2 availability zones. The Elastic Load Balancers, Figure 2 is deployed across the same 2 availability zones for higher availability. Regarding the security groups, the elastic load balancer will be exposed to the public, which means, the public traffic will be hitting the elastic load balancer and the only traffic that will be hitting the auto-scaling group will be on port 80 from the elastic load balancer. EC2 instances are launched and configured it as a web server after which, verify that it is working. After launching the instances create an image of the instance as an Amazon Machine Image (AMI). When it comes to the auto-scaling group, these machine images can be used to scale up the pool of servers. Launch Configuration will decide the extend of scaling

.

Initially, web server is launched in the AWS console, using Amazon Elastic Compute Cloud service. Then, Amazon Machine Image (AMI) is created for each EC2 web server. Figure 3, these AMI is used for creating a launch configuration, which in turn forms the basis for creating autoscaling group, for each EC2 instance, as needed. Parallelly, an application load balancer is created and configured to form the target group with all the required constraints.


Set-up cloud environment with AWS EC2 instances
Set-up NGINX as load balancer
Perform Stress Test using RESTful Stress API
Perform analysis of load balancing algorithms with respect to QoS parameters
Add active health check module to NGINX
Figure 2: Proposed Methodology

The launch configuration is further proceeded to form the auto scaling group. On the other hand, the load balancer along with Target group, in turn, is linked to the auto scaling using the configurations, automatically.

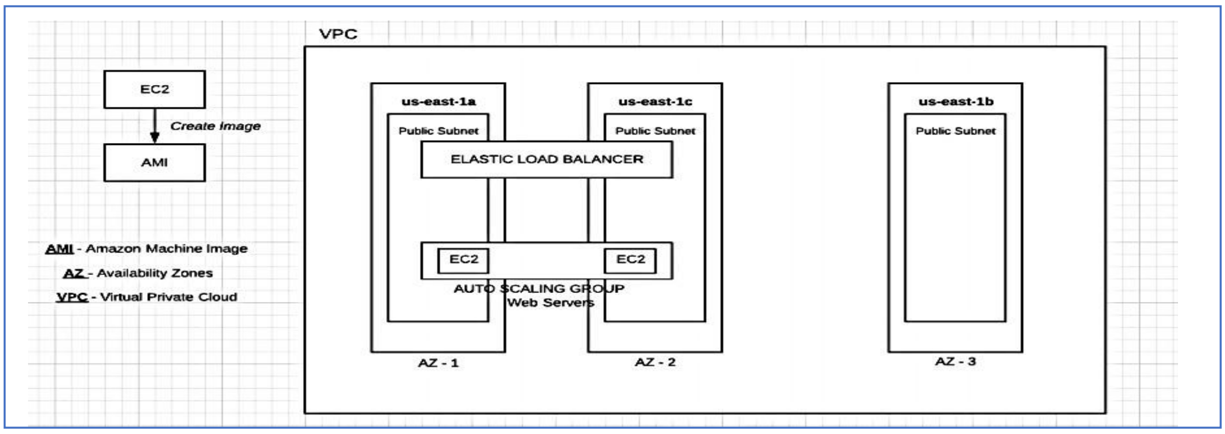


Figure 3: Typical EC2 Load Balancer

V. IMPLEMENTATION DETAILS

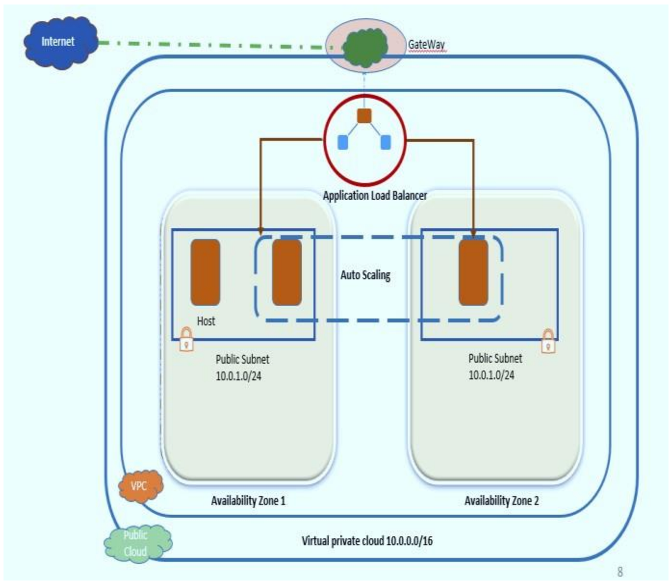


Figure 4: System Architecture of Proposed Load Balancer

The following section describes the implementation Figure 4, details about the analysis of load balancing algorithms as well as the active health check module. The analysis of the load balancing algorithms requires the necessary knowledge of the programming language used in the load balancer. Mobaxterm brings out almost all the essential UNIX commands to a windows environment.

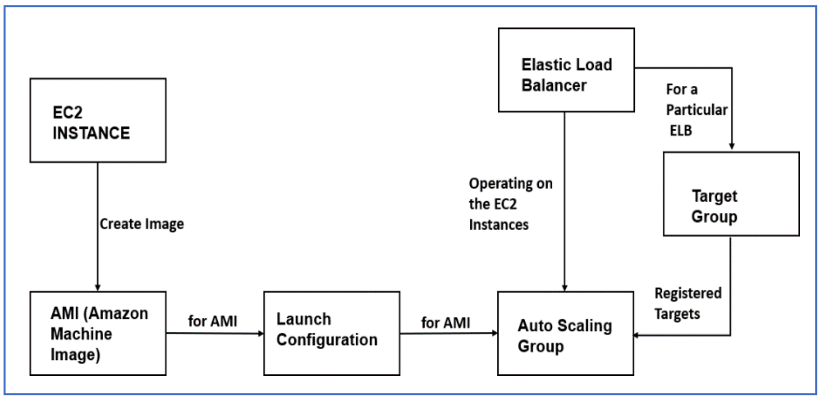
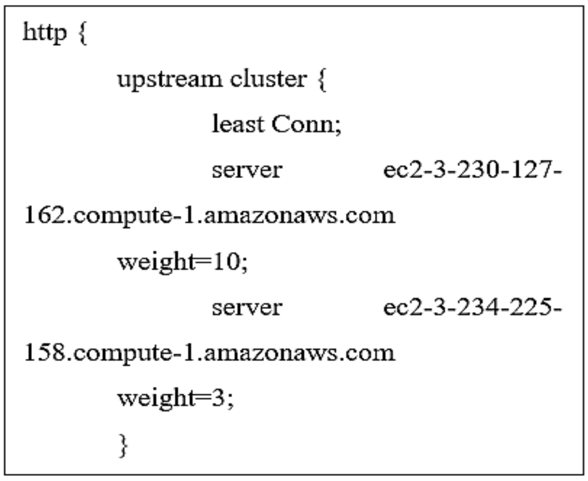


Figure 5: Data Flow

Least Outstanding Requests Algorithm for Load Balancing Requests. In this algorithm, the new incoming request is transferred to the target having the least outstanding requests. This load balancer will enable the load to spread evenly across the targets and ensure that the less capable targets that have low processing power are not burdened with more requests. This load balancer will further prevent long standing queues. It will enable the newer targets to take the load off from the targets that are already overloaded. The entire flow of the pseudocode for the Least Connections algorithm (Figure 6 and Figure 7) begins with the consideration of the total number of back-end web servers. Once the number is determined, the NGINX load balancer accepts incoming request to the web servers from the user

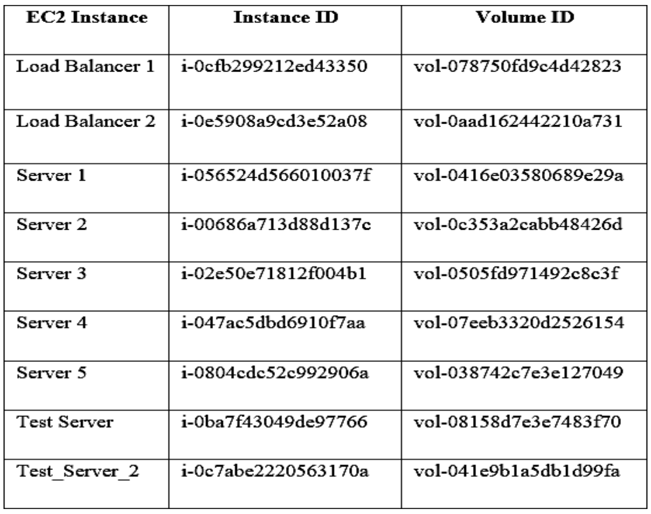
. The incoming requests are then served according to the Least Connections algorithm, in which the servers are checked for their weights and active connections. The one with the least active connections serves the first request, followed by the one with the second least active connections, and so on. The servers which have been configured to work with the Least Connections algorithm but are currently not working or “not alive,” are just skipped over by the algorithm when assigning incoming requests. Finally, the connection to the user is ended once all the requests are served by the algorithm in the load balancer. CURL Test Statements: CURL is a command line tool and a library which is used to receive and send data between a client and a server or any two machines connected over the internet. It supports a wide range of protocols like HTTP, FTP, IMAP, LDAP, POP3, SMTP and many more.

Figure 6: Pseudocode of Least Connections Algorithm



```
http {
        upstream cluster {
                least Conn;
                server          ec2-3-230-127-
162.compute-1.amazonaws.com
        weight=10;
                server          ec2-3-234-225-
158.compute-1.amazonaws.com
        weight=3;
        }
```

A. Implementation Workflow



EC2 Instance	Instance ID	Volume ID
Load Balancer 1	i-0cfb299212ed43350	vol-078750fd9c4d42823
Load Balancer 2	i-0e5908a9cd3e52a08	vol-0aad162442210a731
Server 1	i-056524d566010037f	vol-0416e03580689e29a
Server 2	i-00686a713d88d137c	vol-0c353a2cabb48426d
Server 3	i-02e50e71812f004b1	vol-0505fd971492c8c3f
Server 4	i-047ac5dbd6910f7aa	vol-07eeb3320d2526154
Server 5	i-0804cdc52c992906a	vol-038742c7e3e127049
Test Server	i-0ba7f43049de97766	vol-08158d7e3e7483f70
Test_Server_2	i-0c7abe2220563170a	vol-041e9b1a5db1d99fa

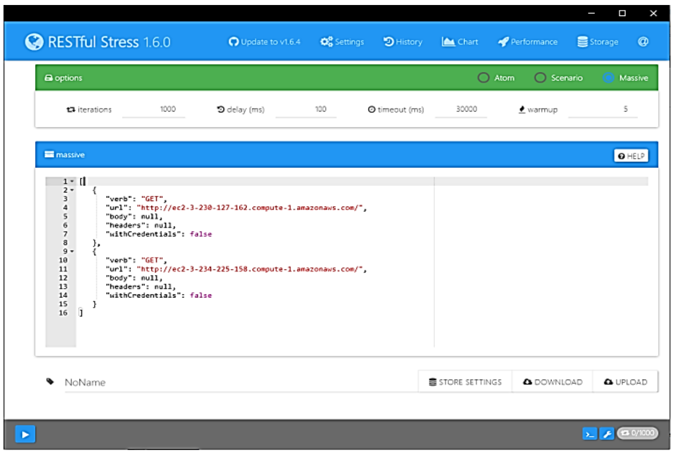
Figure 7: Syntax of configuration of Least Connections algorithm

Firstly, the environment was setup in the AWS management console by configuring a total of 7 EC2 instances, 2 of them being load balancers and 5 of them being the back-end web servers. Additionally, 2 EC2 instances were set up and configured to test the implementation of the active health check module. The instance IDs of the EC2 instances and their attached EBS volumes are displayed in Table 1. NGINX was then set up and configured to run on all the instances, such that the load balancers can redirect traffic to the web servers accordingly. The configuration of NGINX was easy as it is open-sourced and there is a lot of documentation available for the configuration process. The flow control is defined in figure 5.

This was followed by the stress tests that were done for both single load balancer setup as well as the double load balancer setup, with varying the number of servers. These stress tests were done using the RESTful Stress API, which provides various configurations that can be changed when performing the stress tests. Figure 10, depicts the configurations provided by the RESTful Stress API.

Observations were then taken down for each and every implementation scenario, in the form of AWS CloudWatch metrics. A total of 15 metrics has been taken into consideration for the analysis of the load balancing algorithms in the real-time cloud environment of AWS. Some metrics describe the EC2 instances themselves, whereas some metrics describe the attached EBS volumes. These metrics are given in Table 2. This was followed by the preparation of detailed analysis reports for each implementation scenario, with respect to the range of values observed for each CloudWatch metric and the stability of the trend of the graph obtained in the AWS console for each CloudWatch metric.

Comparative results were then tallied with regards to the best load balancing algorithm for a particular scenario. This is due to the fact that there can be a number of variations in terms of network conditions and back-end server configurations in a real-time cloud environment.



iterations 1000 delay (ms) 100 timeout (ms) 30000 warmup 5

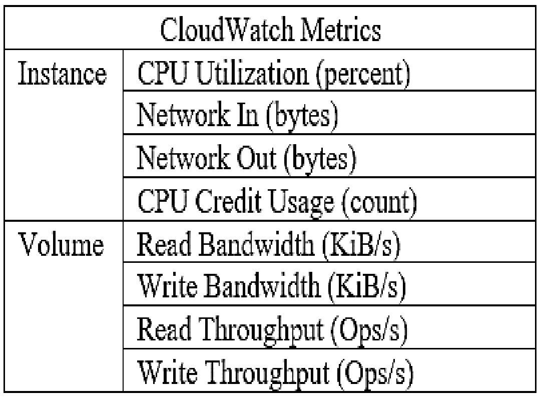
```
[
    {
        "verb": "GET",
        "url": "http://ec2-3-230-127-162.compute-1.amazonaws.com/",
        "body": null,
        "headers": null,
        "withCredentials": false
    },
    {
        "verb": "GET",
        "url": "http://ec2-3-234-225-158.compute-1.amazonaws.com/",
        "body": null,
        "headers": null,
        "withCredentials": false
    }
]
```

NoName

Figure 8: Sample configuration settings of RESTful Stress API

Finally, the active health check module has been written from scratch and implemented in the load balancers in order to continuously check the statuses of the back-end web servers attached to the particular load balancer. Screenshots of the outputs of the active health check module in action have also been taken for both cases of active servers as well as stopped servers.

Table 1: Parameters Taken for the Study



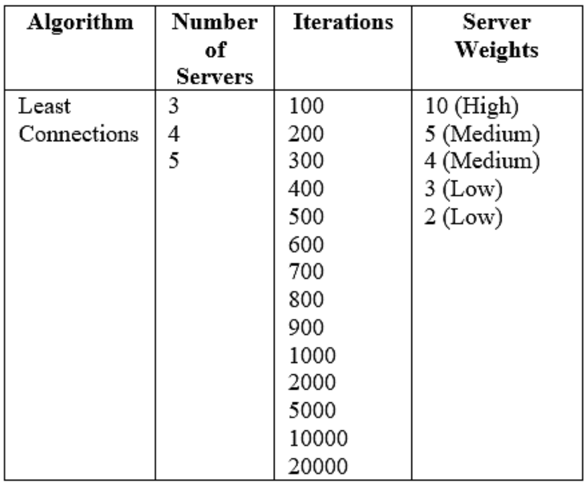
CloudWatch Metrics	
Instance	CPU Utilization (percent)
	Network In (bytes)
	Network Out (bytes)
	CPU Credit Usage (count)
Volume	Read Bandwidth (KiB/s)
	Write Bandwidth (KiB/s)
	Read Throughput (Ops/s)
	Write Throughput (Ops/s)

B. Restful Stress API And Testing Scenarios

The tool used for the stress tests (figure 8) of the load balancing algorithms in the real- time cloud environment of AWS is the RESTful Stress API. The version 1.6.0 was obtained from the Google Chrome Store and used in the stress tests. Stress tests have been performed on the back-end web servers for varying number of iterations, ranging from 100 to 20000. Each stress test took around 1.5 to 2 hours, due to the large number of iterations. The RESTful Stress API supports a variety of options for the massive stress test, including the number of iterations, delay between each iteration, timeout duration for the iterations and warm-up duration. The testing can also be done using a manual Python script using a finite loop, but there is no option to provide delays between each iteration and the subsequent testing would also take a lot longer, due to the large number of iterations.

A configuration was used for testing the load balancing algorithms using the RESTful Stress API, so that it would be easier for analysis. This configuration is given in Table 3. Delay and timeout were constant for all tests, namely 100ms and 30000ms.

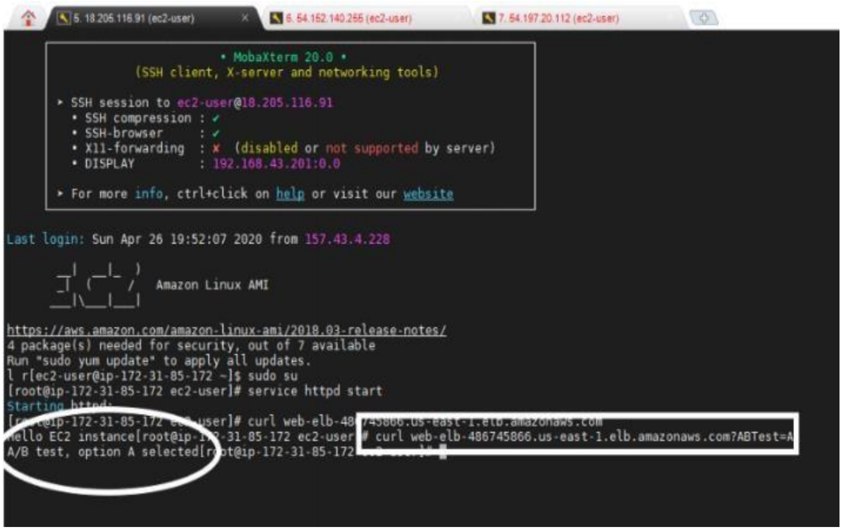
Table V.1: Test Configuration



Algorithm	Number of Servers	Iterations	Server Weights
Least Connections	3 4 5	100 200 300 400 500 600 700 800 900 1000 2000 5000 10000 20000	10 (High) 5 (Medium) 4 (Medium) 3 (Low) 2 (Low)

VI. RESULTS AND ANALYSIS

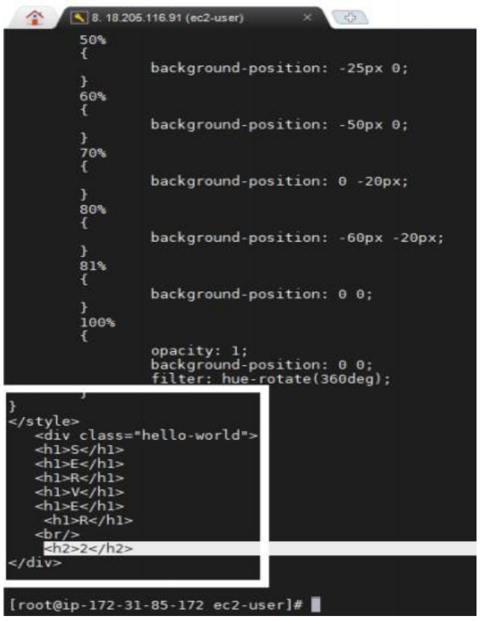
This section discusses all the results that were obtained during the implementation and stress testing phases.


5. 18.205.116.91 (ec2-user)
6. 54.152.140.255 (ec2-user)
7. 54.197.20.112 (ec2-user)

```
            • MobaXterm 20.0 •
  (SSH client, X-server and networking tools)

 ▸ SSH session to ec2-user@18.205.116.91
   • SSH compression : ✓
   • SSH-browser     : ✓
   • X11-forwarding  : ✗  (disabled or not supported by server)
   • DISPLAY         : 192.168.43.201:0.0

 ▸ For more info, ctrl+click on help or visit our website

Last login: Sun Apr 26 19:52:07 2020 from 157.43.4.228

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/
4 package(s) needed for security, out of 7 available
Run "sudo yum update" to apply all updates.
l r[ec2-user@ip-172-31-85-172 ~]$ sudo su
[root@ip-172-31-85-172 ec2-user]# service httpd start
Starting httpd:
[root@ip-172-31-85-172 ec2-user]# curl web-elb-486745866.us-east-1.elb.amazonaws.com
Hello EC2 instance[root@ip-172-31-85-172 ec2-user]# curl web-elb-486745866.us-east-1.elb.amazonaws.com?ABTest=A
A/B test, option A selected[root@ip-172-31-85-172 ec2-user]#
```

Figure 9: Server Configuring on MobaXterm

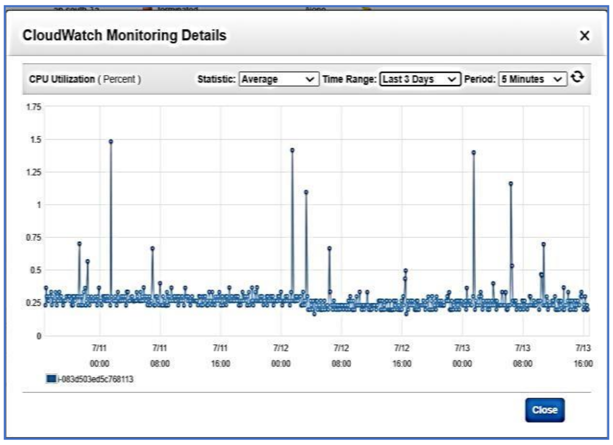
Figure 9 gives the snap shot of the server running on the load balancer and figure 11 gives the web server configuration on mobaxterm.



```
        50%
        {
                background-position: -25px 0;
        }
        60%
        {
                background-position: -50px 0;
        }
        70%
        {
                background-position: 0 -20px;
        }
        80%
        {
                background-position: -60px -20px;
        }
        81%
        {
                background-position: 0 0;
        }
        100%
        {
                opacity: 1;
                background-position: 0 0;
                filter: hue-rotate(360deg);
        }
}
</style>
   <div class="hello-world">
   <h1>S</h1>
   <h1>E</h1>
   <h1>R</h1>
   <h1>V</h1>
   <h1>E</h1>
    <h1>R</h1>
   <br/>
    <h2>2</h2>
</div>

[root@ip-172-31-85-172 ec2-user]#
```

Figure 10: Displaying Server running on the Load Balancer in action

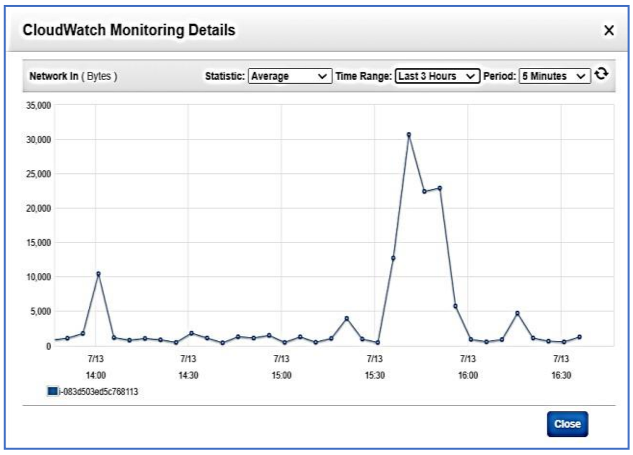
Figure 10 Gives the web server configuration on MobaXterm.


CloudWatch Monitoring Details

CPU Utilization (Percent) Statistic: Average Time Range: Last 3 Days Period: 5 Minutes

Figure 11: CPU Utilization of Designed LB

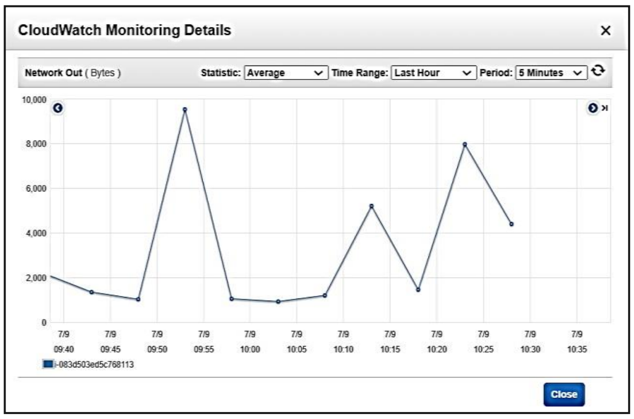
Figure 11 gives the CPU utilization obtained from the load balancer set up. which is better than the average performance given by the default systems. Those values enumerated in table 4 and table 5.


CloudWatch Monitoring Details

Network In (Bytes) Statistic: Average Time Range: Last 3 Hours Period: 5 Minutes

Figure 12: Network IN Designed Load Balancer

Network IN figure 12, is giving the bytes received to unit time. an improved network in means an optimized load balancer.


CloudWatch Monitoring Details

Network Out (Bytes) Statistic: Average Time Range: Last Hour Period: 5 Minutes

Figure 13: Network Out Proposed System

Network IN figure 13, is giving the bytes send out of the network to unit time. an improved network out means an optimized load balancer.

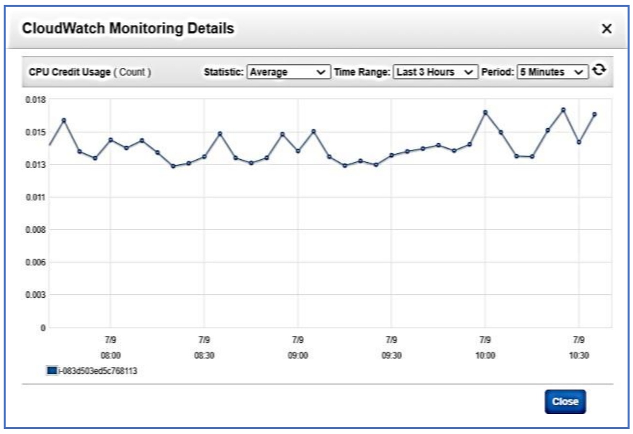
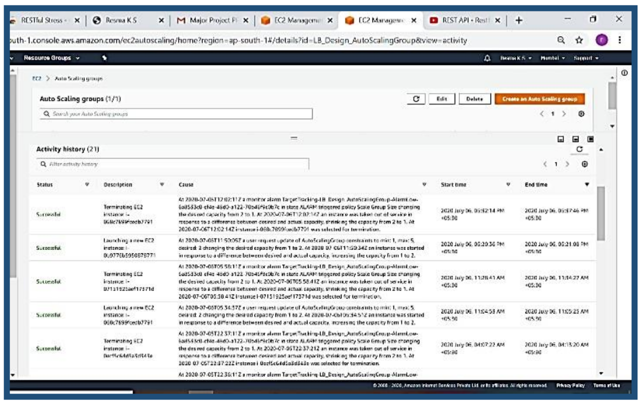


Figure 14: CPU Credit Usage Proposed System

cloud being a pay and use service, figure 14, cpu credit usage is a very important metrics to be studied. this measure directly corelated to the billing for the tenant usage. this metrics is showing a reduced value to average value because of the implementation of the auto scaling features.
Figure 15 gives the demonstration of the auto scaling features of the designed load balancer. as and


RESTful Stress
Resma K S
Major Project Pl
EC2 Managem
EC2 Managem
REST API - Rest
south-1.console.aws.amazon.com/ec2autoscaling/home?region=ap-south-1#/details?id=LB_Design_AutoScalingGroup&view=activity
Resource Groups
Resma K S
Mumbai
Support
EC2 > Auto Scaling groups
Auto Scaling groups (1/1)
Edit
Delete
Create an Auto Scaling group
Search your Auto Scaling groups
Activity history (21)
Filter activity history
© 2008 - 2020, Amazon Internet Services Private Ltd. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

Status	Description	Cause	Start time	End time
Successful	Terminating EC2 instance: i-068c7699fcedb7791	At 2020-07-06T12:02:11Z a monitor alarm TargetTracking-LB_Design_AutoScalingGroup-AlarmLow-6ad553c0-cf4e-48d0-a122-70b4f9c0b7c in state ALARM triggered policy Scale Group Size changing the desired capacity from 2 to 1. At 2020-07-06T12:02:14Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 2 to 1. At 2020-07-06T12:02:14Z instance i-068c7699fcedb7791 was selected for termination.	2020 July 06, 05:52:14 PM +05:30	2020 July 06, 05:57:46 PM +05:30
Successful	Launching a new EC2 instance: i-0b0776b5950878771	At 2020-07-06T11:50:05Z a user request update of AutoScalingGroup constraints to min: 1, max: 5, desired: 2 changing the desired capacity from 1 to 2. At 2020-07-06T11:50:34Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 1 to 2.	2020 July 06, 05:20:36 PM +05:30	2020 July 06, 05:21:08 PM +05:30
Successful	Terminating EC2 instance: i-07151925aef17371d	At 2020-07-06T05:58:11Z a monitor alarm TargetTracking-LB_Design_AutoScalingGroup-AlarmLow-6ad553c0-cf4e-48d0-a122-70b4f9c0b7c in state ALARM triggered policy Scale Group Size changing the desired capacity from 2 to 1. At 2020-07-06T05:58:41Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 2 to 1. At 2020-07-06T05:58:41Z instance i-07151925aef17371d was selected for termination.	2020 July 06, 11:28:41 AM +05:30	2020 July 06, 11:34:27 AM +05:30
Successful	Launching a new EC2 instance: i-068c7699fcedb7791	At 2020-07-06T05:34:37Z a user request update of AutoScalingGroup constraints to min: 1, max: 5, desired: 2 changing the desired capacity from 1 to 2. At 2020-07-06T05:34:51Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 1 to 2.	2020 July 06, 11:04:53 AM +05:30	2020 July 06, 11:05:25 AM +05:30
Successful	Terminating EC2 instance: i-0acf5c64d5a5d543e	At 2020-07-05T22:37:11Z a monitor alarm TargetTracking-LB_Design_AutoScalingGroup-AlarmLow-6ad553c0-cf4e-48d0-a122-70b4f9c0b7c in state ALARM triggered policy Scale Group Size changing the desired capacity from 2 to 1. At 2020-07-05T22:37:21Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 2 to 1. At 2020-07-05T22:37:22Z instance i-0acf5c64d5a5d543e was selected for termination.	2020 July 06, 04:07:22 AM +05:30	2020 July 06, 04:13:20 AM +05:30
		At 2020-07-05T22:36:11Z a monitor alarm TargetTracking-LB_Design_AutoScalingGroup-AlarmLow-		

figure 15: servers scaled up automatically with increased load.

when the load on the system increases, the web servers are launched automatically according to the rule set for the autoscaling. in the experiment conducted the rule set was to increase the number of server if the utilization of the server increases >=50% of the total capacity of the pool of servers.

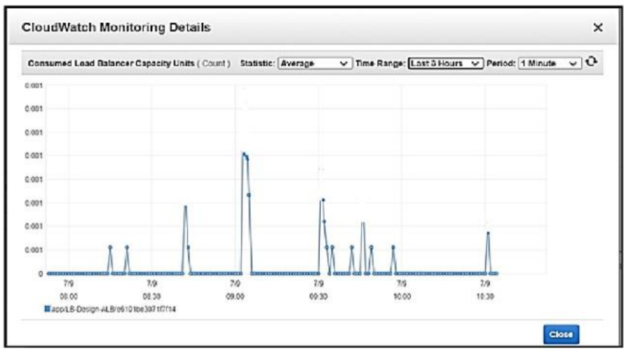


Figure 16: Consumed Load Balancer Capacity

In Figure 16, the matrix is about how optimized the resource utilization is happening with the load

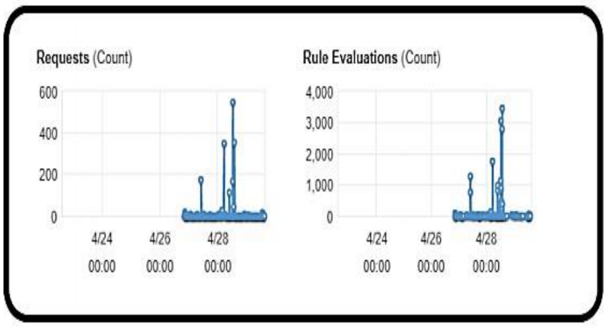
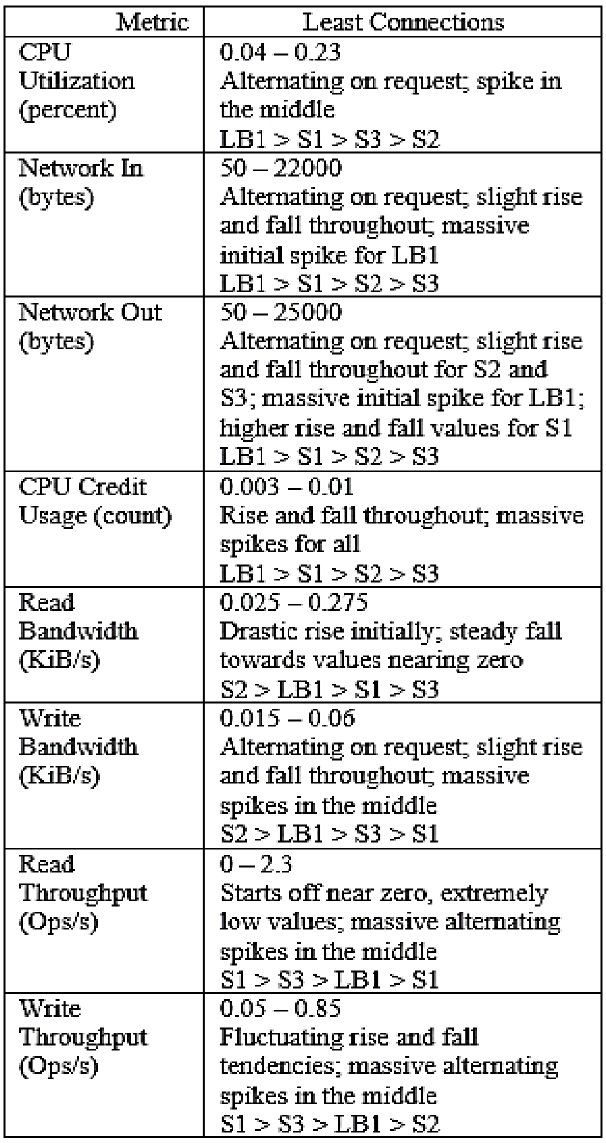


Figure 17: Graphical Analysis before the implementation of ARR algorithm

Algorithm	Single Load Balancer Results		Double Load Balancer Results	
	Less Servers	More Servers	Less Servers	More Servers
Least Connections	9	9	10	9

Table 3: Analysis report for Least Connections algorithm

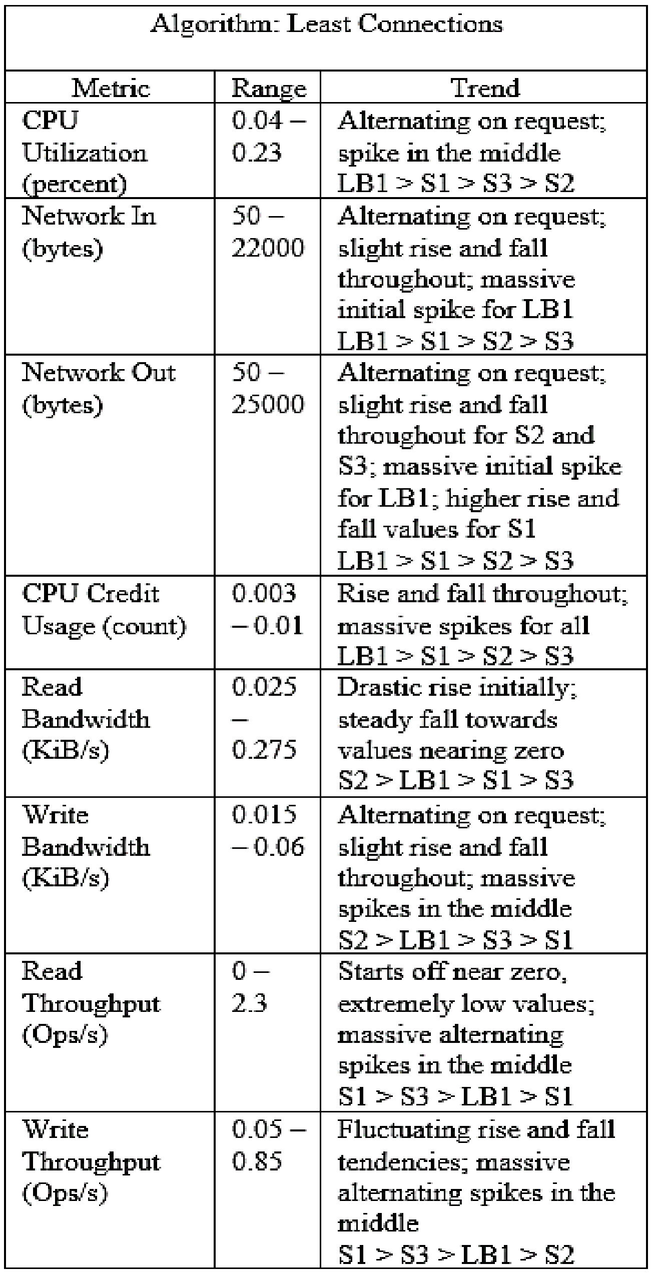
Table 5: Final analysis report



Metric	Least Connections
CPU Utilization (percent)	0.04 – 0.23 Alternating on request; spike in the middle LB1 > S1 > S3 > S2
Network In (bytes)	50 – 22000 Alternating on request; slight rise and fall throughout; massive initial spike for LB1 LB1 > S1 > S2 > S3
Network Out (bytes)	50 – 25000 Alternating on request; slight rise and fall throughout for S2 and S3; massive initial spike for LB1; higher rise and fall values for S1 LB1 > S1 > S2 > S3
CPU Credit Usage (count)	0.003 – 0.01 Rise and fall throughout; massive spikes for all LB1 > S1 > S2 > S3
Read Bandwidth (KiB/s)	0.025 – 0.275 Drastic rise initially; steady fall towards values nearing zero S2 > LB1 > S1 > S3
Write Bandwidth (KiB/s)	0.015 – 0.06 Alternating on request; slight rise and fall throughout; massive spikes in the middle S2 > LB1 > S3 > S1
Read Throughput (Ops/s)	0 – 2.3 Starts off near zero, extremely low values; massive alternating spikes in the middle S1 > S3 > LB1 > S1
Write Throughput (Ops/s)	0.05 – 0.85 Fluctuating rise and fall tendencies; massive alternating spikes in the middle S1 > S3 > LB1 > S2

balancer. this is showing a positive result in comparison to the existing systems [2]. Figure 16 And Figure 17 gives the comparison number of request and rule evaluation, where the graphs show clear advantages of implementation with ARR algorithm. table 4, table 5 and table 6 enumerates all the observations obtained from the load balancer implementation in a concise format.

Table 4: Final scores for the load balancing algorithms



Algorithm: Least Connections		
Metric	Range	Trend
CPU Utilization (percent)	0.04 – 0.23	Alternating on request; spike in the middle LB1 > S1 > S3 > S2
Network In (bytes)	50 – 22000	Alternating on request; slight rise and fall throughout; massive initial spike for LB1 LB1 > S1 > S2 > S3
Network Out (bytes)	50 – 25000	Alternating on request; slight rise and fall throughout for S2 and S3; massive initial spike for LB1; higher rise and fall values for S1 LB1 > S1 > S2 > S3
CPU Credit Usage (count)	0.003 – 0.01	Rise and fall throughout; massive spikes for all LB1 > S1 > S2 > S3
Read Bandwidth (KiB/s)	0.025 – 0.275	Drastic rise initially; steady fall towards values nearing zero S2 > LB1 > S1 > S3
Write Bandwidth (KiB/s)	0.015 – 0.06	Alternating on request; slight rise and fall throughout; massive spikes in the middle S2 > LB1 > S3 > S1
Read Throughput (Ops/s)	0 – 2.3	Starts off near zero, extremely low values; massive alternating spikes in the middle S1 > S3 > LB1 > S1
Write Throughput (Ops/s)	0.05 – 0.85	Fluctuating rise and fall tendencies; massive alternating spikes in the middle S1 > S3 > LB1 > S2

VII. CONCLUSION

The paper is presenting a load balancing model using the Amazon Web Services (AWS) cloud. Various types of load balancers were studied before finalizing our load balancer. These three EC2 instances were maintained on the ’MobaXterm’ platform for back-end, as Linux machines, using their respective IPV4 addresses. The default load balancer to be implemented was Round Robin. The algorithm used for the study is an improved Least Outstanding Request as the main load balancing strategy for the implementation. The EC2 servers were further auto scaled depending on the increasing/decreasing ’CPU Utilization’ on the EC2 configurations of the same instances in the MobaXterm. Advanced Request Routing (ARR) algorithm was also implemented. The basic if-then rules were configured for the results in the Mobaxterm. With the use of ARR algorithm, server running on the load balancer, at any time, could be traced back in any EC2 configurations running on the MobaXterm in real-time. Graphical Analysis as a measure of comparison in ’Rule Evaluation Count’ was done before and after the use of the ARR algorithm in the AWS console.

References

[1] Mohamed Belkhouraf, Ali Kartit And Hassan Ouahmane Hamza Kamal Idrissi, Zaid Kartit And Mohamed El Marraki Secured Load Balancing Architecture For Cloud Computing Based On Multiple Clusters- 978-1-4673-8149-9/15/$31.00 ©2015 IEEE

[2] Dr.B.Vinayagasundaram, R Swathy ,Load Balancing In Cloud Environment Using Stackelberg's Approach, 2017 Ninth International Conference On Advanced Computing (Icoac).

[3] Shang-Liang Chen ,Yun-Yao Chen∗,Suang-Hong Kuo, CLB: A Novel Load Balancing Architecture And Algorithm For Cloud Service, Http://Dx.Doi.Org/ 10.1016 /J.Compeleceng.2016.01.029, 0045- 7906/© 2016 Elsevier Ltd.

[4] Kashif Bilal, Saif Ur Rehman Malik, And Samee U. Khan, Trends And Challenges In Cloud Datacentres, IEEE Cloud Computing · June 2014 DOI: 10.13140/2.1.1032.2568

[5] Jianhua Gu, Jinhua Hu, Tianhai Zhao, A New Resource Scheduling Strategy Based On Genetic Algorithm In Cloud Computing Environment, © 2012 ACADEMY PUBLISHER Doi:10.4304/Jcp.7.1.42-52.

[6] Resma K. S, Akash Hegde, Dr. Sharvani G. S. Elastic Load Balancing In Real-Time Cloud Environment, International Journal Of Advanced Science And Technology Vol. 29, No. 12s, (2020), Pp. 269-283.

[7] Sambit Kumar Mishra, Bidhudatta Sahoo, Priti Paramita Parida, Load Balancing In Cloud Computing A Big Picture. 1319-1578/ 2018 The Authors. Production And Hosting By Elsevier B.V. On Behalf Of King Saud University.

[8] Rafiqul Zaman Khan, Md Oqail Ahmad, Load Balancing Challenges On Cloud Computing: A Survey, © Springer India 2016 D.K. Lobiyal Et Al. (Eds.), Proceedings Of The International Conference On Signal, Networks, Computing, And Systems, Lecture Notes In Electrical Engineering 396, DOI 10.1007/978-81-322-3589-7_3

[9] Resma K. S, Dr. Sharvani G. S, Amruta Tapas Paul, A Closer Look At The Network Middleboxes, 978-1-7281-0418-8/19/$31.00©2019 IEEE

[10] Kousik Dasguptaa, Brototi Mandalb, Paramartha Duttac, Jyotsna Kumar Mondald, Santanu Dame ,A Genetic Algorithm (GA) Based Load Balancing Strategy For Cloud Computing, 2212-0173 © 2013 The Authors. Published By Elsevier Ltd. Open Access Under CC BY-NC-ND License.

[11] Resma K. S, Dr. Sharvani G. S, Edge Distributed Cloud Middleboxes International Journal Of Advance Research, Ideas And Innovations In Technology, ISSN: 2454-132X, Volume 5, Issue 3.
Luo, J., Rao, L., & Liu, X. (2017). Spatio-Temporal Load Balancing For Energy Cost Optimization In Distributed Internet Data Centres. IEEE Transactions On Cloud Computing, 3(3), 387– 397.

[12] Nahir, A., Orda, A., & Raz, D. (2016). Replication Based Load Balancing. IEEE Transactions On Parallel And Distributed Systems, 27(2), 494–507.

[13] Resma, K. S., G. S. Sharvani, And Ramasubbareddy Somula. "Optimization Of Cloud Load Balancing Using Fitness Function And Duopoly Theory." International Journal Of Intelligent Computing And Cybernetics (2021).

[14] KS Resma, GS Sharvani, Datacenter Reliability Pursuance With Optimized Load Balancers, Cloud Reliability Engineering: Technologies And Tools, 2021,CRC Press.

[15] Akash Hegde, Resma K. S., Dr. Sharvani G. S. (2020). Elastic Load Balancing In Real-Time Cloud Environment. International Journal Of Advanced Science And Technology, 29(12s), 269 - 283. Retrieved From Http://Sersc.Org/Journals/Index.Php/IJAST/Article/View/21935.

[16] Somula, R. S., & Sasikala, R. (2018). A Survey On Mobile Cloud Computing: Mobile Computing+ Cloud Computing (MCC= MC+ CC). Scalable Computing: Practice And Experience, 19(4), 309-337.

[17] Somula, R., Anilkumar, C., Venkatesh, B., Karrothu, A., Kumar, C. P., & Sasikala, R. (2019). Cloudlet Services For Healthcare Applications In Mobile Cloud Computing. In Proceedings Of The 2nd International Conference On Data Engineering And Communication Technology (Pp. 535-543). Springer, Singapore.

[18] Somula, R., & Sasikala, R. (2018). Round Robin With Load Degree: An Algorithm For Optimal Cloudlet Discovery In Mobile Cloud Computing. Scalable Computing: Practice And Experience, 19(1), 39-52.

[19] Somula, R., & Sasikala, R. (2019). A Load And Distance Aware Cloudlet Selection Strategy In Multi-Cloudlet Environment. International Journal Of Grid And High Performance Computing (IJGHPC), 11(2), 85-102.

[20] Somula, R., & Sasikala, R. (2019). A Honey Bee Inspired Cloudlet Selection For Resource Allocation.

In Smart Intelligent Computing And Applications (Pp. 335-343). Springer, Singapore.

[21] Somula, R., & Sasikala, R. (2019). A Research Review On Energy Consumption Of Different Frameworks In Mobile Cloud Computing. Innovations In Computer Science And Engineering, 129-142.

[22] Somula, R., Narayana, Y., Nalluri, S., Chunduru, A., & Sree, K. V. (2019). POUPR: Properly Utilizing User-Provided Recourses For Energy Saving In Mobile Cloud Computing. In Proceedings Of The 2nd International Conference On Data Engineering And Communication Technology (Pp. 585-595). Springer, Singapore.

[23] Ramasubbareddy, S., Ramasamy, S., Sahoo, K. S., Kumar, R. L., Pham, Q. V., & Dao, N. N. (2020). CAVMS: Application-Aware Cloudlet Adaption And VM Selection Framework For Multicloudlet Environment. IEEE Systems Journal.

[24] Sahoo, K. S., Mishra, P., Tiwary, M., Ramasubbareddy, S., Balusamy, B., & Gandomi, A. H. (2019). Improving End-Users Utility In Software-Defined Wide Area Network Systems. IEEE Transactions On Network And Service Management, 17(2), 696-707.

[25] Sahoo, K. S., Tripathy, B. K., Naik, K., Ramasubbareddy, S., Balusamy, B., Khari, M., & Burgos, D. (2020). An Evolutionary SVM Model For DDOS Attack Detection In Software Defined Networks. IEEE Access, 8, 132502-132513.

[26] Sennan, S., Balasubramaniyam, S., Luhach, A. K., Ramasubbareddy, S., Chilamkurti, N., & Nam, Y. (2019). Energy And Delay Aware Data Aggregation In Routing Protocol For Internet Of Things. Sensors, 19(24), 5486.

[27] Sankar, S., Srinivasan, P., Luhach, A. K., Somula, R., & Chilamkurti, N. (2020). Energy-Aware Grid-Based Data Aggregation Scheme In Routing Protocol For Agricultural Internet Of Things. Sustainable Computing: Informatics And Systems, 28, 100422.

[28] Mohanta, B. K., Jena, D., Ramasubbareddy, S., Daneshmand, M., & Gandomi, A. H. (2020). Addressing Security And Privacy Issues Of Iot Using Blockchain Technology. IEEE Internet Of Things Journal, 8(2), 881-888.

[29] Sahoo, K. S., Tiwary, M., Sahoo, B., Mishra, B. K., Ramasubbareddy, S., & Luhach, A. K. (2019). RTSM: Response Time Optimisation During Switch Migration In Software-Defined Wide Area Network. IET Wireless Sensor Systems, 10(3), 105-111.

[30] Sennan, S., Ramasubbareddy, S., Luhach, A. K., Nayyar, A., & Qureshi, B. (2020). CT-RPL: Cluster Tree Based Routing Protocol To Maximize The Lifetime Of Internet Of Things. Sensors, 20(20), 5858.

[31] Sennan, S., Balasubramaniyam, S., Luhach, A. K., Ramasubbareddy, S., Chilamkurti, N., & Nam, Y. (2019). Energy And Delay Aware Data Aggregation In Routing Protocol For Internet Of Things. Sensors, 19(24), 5486.

[32] Sennan, S., Balasubramaniyam, S., Luhach, A. K., Ramasubbareddy, S., Chilamkurti, N., & Nam, Y. (2019). Energy And Delay Aware Data Aggregation In Routing Protocol For Internet Of Things. Sensors, 19(24), 5486.

[33] Mohanta, B. K., Jena, D., Mohapatra, N., Ramasubbareddy, S., & Rawal, B. S. (2021). Machine Learning Based Accident Prediction In Secure Iot Enable Transportation System. Journal Of Intelligent & Fuzzy Systems, (Preprint), 1-13.

[34] Sennan, S., Ramasubbareddy, S., Balasubramaniyam, S., Nayyar, A., Abouhawwash, M., & Hikal, N. A. (2021). T2FL-PSO: Type-2 Fuzzy Logic-Based Particle Swarm Optimization Algorithm Used To Maximize The Lifetime Of Internet Of Things. IEEE Access.

[35] Ramasubbareddy, S., Swetha, E., Luhach, A. K., & Srinivas, T. A. S. (2021). A Multi-Objective Genetic Algorithm-Based Resource Scheduling In Mobile Cloud Computing. International Journal Of Cognitive Informatics And Natural Intelligence (IJCINI), 15(3), 58-73.