CHAPTER 4
AN EFFECTIVE DYNAMIC PRIORITIZED LOAD BALANCING ON CLOUD MOBILE SERVICES USING ROUND ROBIN ELASTICITY MECHANISM

4.1 INTRODUCTION

Machine Flow based Energy-Power Approximation method achieves energy efficient system for the cloud mobile services. Multi-grid approximation technique reduces the energy consumption and mapping technique provides better energy usage in mobile communication. Load balancing is an emerging part of research in cloud environment. Cloud customer finds greater balancing services during cloud load balancing which resulted in performance improvement in cloud. Load balancing operation will be originated to successfully use the resources changing with the process of transfer resources to the parallel node to reduce the load values. The cloud mobile model consists of difficulty on energy saving and power utilization on particular cloud devices. Hence, the characteristic of the devices fails to enhance energy-efficient organization on the elastic cloud computing environments. The load balance approach is provides for performing various types of user requests from different cloud mobile environment. Hence Dynamic Prioritized Load Balanced Round Robin (DP-LBRR) framework is designed using the priority value.

Cost-effective Cloud High Performance Computing Resource Provisioning by Semi-Elastic Cluster (SEC) model combines all the resource strategies into a unique set on the public cloud. However, SEC does not forecast the entire number of instances within the time interval so that the CPU load is raised in the cloud elasticity
environment. Cloud Computing Services for Many-Tasks Scientific Computing focused on calculating performance of Many-Task Computing (MTC). However, performance of MTC system over utilizes the CPU load and high latency in the cloud mobile media network.

The cloud mobile resources are not efficient on transparent energy usage based on elastic applications because the components are restructured depending on the workload conditions. While public cloud combines all the resource strategies on scientific workloads, the CPU load is raised and is not effective in solving the optimization problem in different elasticity cloud applications. To overcome these issues, DP-LBRR framework is constructed. The Load Balanced Round Robin (LBRR) algorithm in proposed DP-LBRR framework generally directed on allocating the load to the CPU planning at balancing CPU load rate. The numerous requests are generated based on CPU planning from different cloud mobile environment that assigns each virtual machine in CPU with cyclic order in order to reduce the latency time. A Dynamic Priority Load Scheduler is included in proposed LBRR algorithm by using priority values. This load scheduler is located in the cloud system for minimizing the memory utilization and solves the optimization difficulties in the elasticity cloud services. The priority value is calculated in the Dynamic Prioritized Load Balanced Round Robin framework based on the active priority model aiming at reducing the memory utilization. The active priority locates the maximum and the next maximum priority nodes for easy scheduling in the cloud mobile environments and hence the scheduling efficiency in the elasticity cloud services is improved.
4.2. RESOURCE PROVISIONING AND LOAD BALANCING FOR FUTURE INTERNET IN CLOUD ENVIRONMENT

Cloud load balancing is the method of dividing workloads and calculating resources in a cloud computing environment. Load balancing allocates tasks to maintain application or workload requirements by allocating resources between several computers, networks or servers. Cloud load balancing suggests hosting the distribution of workload traffic and requirements that locate over the Internet. High Performance Computing (HPC) is introduced to adjust with the cloud platforms, cloud users have started re-considering the applicability of parallel computing based on cloud environment.

4.2.1. Cost-effective cloud HPC resource provisioning by building semi-elastic virtual Clusters

Shuangcheng Niu et al. (2013) developed a Semi-Elastic Cluster (SEC) computing model which tolerates organizations to maintain a set of cloud requests. In addition to that, Semi elastic model preserves schedule jobs surrounded by the modern capability and effectively correct the ability of the elasticity. According to the system load, ability stage with user requirements are effectively correcting the ability stage in responsiveness (queue wait time), and the cloud provider's charging granularity. This in turn combines all the resource approaches into a unique set on the public cloud.

Semi-Elastic Cluster (SEC) model developed in three ways. Initially, it combines the demands from several users, permitting a “Groupon” mode in cloud computing for attaining lower per-instance rates. Also, each stored request gets completely utilized after the organization covers the upfront reservation charge. Next, online forecast is performed
in semi-elastic cluster based on provisioning request with various types of reserved applications to enhance cost effectiveness. It can efficiently control the virtual cluster ability and arrange it resource distribution across different cloud pricing classes. Finally, sharing instances between multiple users tolerates an organization to efficiently develop residual resources incurred by common cloud charging granularity, in addition to repay the latency of booting cloud-based virtual clusters.

Cloud-based cluster is developed in SEC model for performing parallel jobs and to minimize according to the present system load. In addition to prioritizing and transmitting jobs, it also completes the resource provisioning, in the structure of dynamic cluster size scaling. Based on priority backfilling algorithm integrated cluster size scaling and batch job scheduling model is constructed. The priority of each job is designed dynamically based on back filling algorithm where the jobs are arranged in the job queue. The design of jobs is considered based on user-spaced priority level, time and job size in the queue in terms of node request. Resource condition will be made for individual or several jobs at the top of the queue (top jobs), based on the expected execution time specified in the job script.

4.2.1.1. Automatic configuration of reserved instance collection

The Semi-Elastic Cluster method obtains the benefit of economies of range by combining workloads from users. SEC system involves preserving several types of request to increase its flexibility, decrease the risk of collecting large-usage instances, and optimize its overall cost-effectiveness. The rationale at this point a guaranteed minimum level of combined usage can be fulfilled with the most heavily discounted
request class, incremented with multiple classes of more expensive. The SEC system is managed by an organization, where the combined job history can be recorded and examined. The optimal stored request provisioning approach classify the given job history based on pricing structure offline stored request algorithm. Thus, it produces the group of job submission and Semi-Elastic cluster job scheduling algorithm. As the results, significant wait time, managed by the parameter $T_{waitlim}$, serves as a “load smoother” that can extensively reduce the burst from instance demands.

The real SEC managers comprises present of online request provisioning without considering an entire trade record. Online algorithm struggles to make a rational long-term SEC usage trend prediction, to be fixed with conventional instance reservation approaches. Important perception in online algorithm is if available, a SEC cluster may acquire workload dynamics that sits among on traditional supercomputers/clusters and on social media/networks. Scheduling algorithm in semi-elastic cluster model includes dynamic request provisioning depending on the stored request work of a SEC cluster.

### 4.2.2. Load balancing for future internet: an approach based on game theory

Shaoyi Song et al. (2014) developed a Load Balancing for Future Internet based on game theory model. Load balancing model reaches energy conservation, enhancing system performance, and low cost on preserving system information. This in turn improves the scalability to the model.

Based on Game theory a load balancing approach contains two processes. Initially construct a load balancing algorithm for solving static load balancing problem.
Then next, applying semi decentralized effect to the load balancing difficulty of future internet. This result is also a mixture technique that merges a non co-operative game among users and a cooperative game among processors (NOCOG). In this model, all the nodes not need to manage as much information since in traditional method. Hence system performance is improved and better fairness between processor nodes. Load balancing of future internet method extend the entire internet’s computing load, traffic load, and further items depending on networks’ resources.

Load balancing in future internet, resource area considered as computing area presenting computation services. All jobs are approved by load managers for transmitting the user jobs based on computing center and this load manager obtain jobs from various users and it established by additional load manager. The load manager transmits the jobs to the processors which are managed directly it receives them. A job may be performed by an allotted CPU and wouldn’t be transmitted over to a different processor. Each processor preserves a queue that holds jobs to be finished. Each job is processed on a first come- first-serve (FCFS) basis and then sends the results back to users.

The static load balancing calculates the problem as cooperative game among processors for reducing overall estimated response time. However, fairness index of processors is preserved and fairness between users is hard to complete. Hence this method produces the minimum response time. The non co-operative approach is used to find bad fairness between processors and a cooperative game with processors access examines the lowest system executing time. The fairness index of customers is disregarded in cooperative game method. At the equal time, solve the load balancing
difficulty in future internet from the side of users. However, non co-operative method
fails to produce better fairness between processor nodes.

4.3. DYNAMIC PRIORITIZED LOAD BALANCED ROUND ROBIN (DP- LBRR)
FRAMEWORK

DP-LBRR framework is constructed based on the LBRR algorithm. The
distribution of load to the CPU is determined by maintaining the load rate with the help
of Load Balancing Round Robin algorithm. In addition, Dynamic Priority Load
Scheduler is used for reducing the memory consumption in cloud services. By using
Load Balanced Round Robin and Dynamic Prioritized model reduce the CPU load in the
cloud elasticity environment.

4.3.1. Load Balanced Round Robin Model

Initially, Load Balancing Round Robin algorithm is designed for the construction
of proposed DP-LBRR framework. The frequency of cloud customers has been
increasing exponentially and therefore proper scheduling of virtual machines in the cloud
mobile environments becomes a significant issue to analyze.

The main objective of Load Balanced Round Robin (LBRR) algorithm in DP-
LBRR framework is allocating the load to the CPU for executing the various categories
of requirements from several cloud mobile environments. LBRR algorithm distributes
each virtual machine in a repeated order. Figure 4.1 illustrates the block diagram of Load
Balancing Round Robin.
As shown in the figure 4.1, at first appeared loads from cloud environment are measured. Then, according to the load arrived, the load scheduler balances the load, allocates the load uniformly to all the CPUs. The load scheduler assigns single Virtual Machine (VM) to a CPU in a repeated order.

![Block diagram of Load Balanced Round Robin algorithm](image)

**Figure 4.1 Block diagram of Load Balanced Round Robin algorithm**

The load scheduler begins with a CPU and progress on to following CPU after a Virtual Machine (VM) is allocated to that CPU. This procedure of scheduling is frequented until all the CPUs have been assigned at least one VM and then the scheduler returns to the first CPU again. Therefore, in this case, the scheduler does not wait for the exhaustion of the properties (i.e. load) of a CPU ahead moving on to the next CPU.
Consider a model with a scheduler ‘S’, ‘n’ CPUs, ‘\( \text{CPU}_i = \text{CPU}_1, \text{CPU}_2, \ldots, \text{CPU}_n \)’, and ‘n’ VMs ‘\( \text{VM}_i = \text{VM}_1, \text{VM}_2, \ldots, \text{VM}_n \)’ to be planned, then as shown in the figure using LBRR. All ‘CPU’ is dispersed with one ‘VM’ in such a technique that all the ‘CPUs’ have enough accessible resources to work the ‘VMs’.

\[
\text{Req}_i = \sum_{i=1}^{n} \text{CPU}_i \rightarrow \text{VM}_i
\]

\[\ldots \ldots \text{Eqn (4.1)}\]

From above equation (4.1), for each user request ‘\( \text{Req}_i \)’, a CPU ‘\( \text{CPU}_i \)’ is allotted with one ‘\( \text{VM}_i \)’. The LBRR algorithm in DP-LBRR framework employs each and every resources (i.e. load) in a balanced manner. The identical amount of VMs is consigned to every CPU, which in turn ensures fairness and therefore CPU load rate is balanced.

The Load Balanced Round Robin algorithm develops the power utilization by spinning off the VMs which are not presently used. Instead of charging all the VMs turned on, the Load Balanced Round Robin algorithm turns off the unused VMs which extensively decrease the latency time in elasticity cloud computing. With LBRR algorithm, the load scheduler assigns a VM to the CPU and then crosses through the record of CPUs to verify if the VM is vacant and if establish to be unused, turns it off. If the resource of a CPU which has been switched off is necessary for the allocation of a VM, the load scheduler turns the VM on following that shares the VM to that CPU. In this manner, the LBRR algorithm turn off the unused virtual machine decreasing the latency time in the elasticity cloud computing. Below algorithm 4.1 demonstrates the Load Balanced Round Robin algorithm.
The LBRR algorithm contains three major steps. Initially load scheduler consigns each CPU with a virtual machine VM for the consequent user. In this method CPU load rate is balanced in a broad way. Then next step is carried out by the traversal of load scheduler through the record of CPUs to observe that if any vacant VM are present in the cloud environment.

<table>
<thead>
<tr>
<th>Input: Request ‘Reqᵢ = Req₁,Req₂,...,Reqₙ’, Load Scheduler ‘S’, CPU ‘CPUᵢ = CPU₁,CPU₂,...,CPUₙ’ Virtual Machine ‘VMᵢ = VM₁,VM₂,...VMₙ’</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: CPU load balanced with reduced latency</td>
</tr>
</tbody>
</table>

Step 1: Begin
Step 2: For each user request Reqᵢ
Step 3: Load Scheduler assigns each ‘CPU’ with one ‘VM’
Step 4: Traversal of load scheduler through the list of CPUs
Step 5: If VM is unused
Step 6: Load scheduler turn off the VM
Step 7: End if
Step 8: If CPU required for allocation
Step 9: Load scheduler turn on the VM
Step 10: End if
Step 11: End for
Step 12: End

Algorithm 4.1 Load Balanced Round Robin algorithm
This is because with the presence of vacant VM major quantity of power will be addicted. Hence with the intention of reducing the power utilization, the load scheduler in the LBRR algorithm turns off the vacant VM. Alternatively, if CPU is additional required for distribution, the load scheduler in the LBRR algorithm turns on the VM ensuring fairness. Therefore, the latency time is reduced in a significant way.

4.3.2. Dynamic Priority Load Scheduler

The Load Balanced Round Robin with Prioritized framework organize the mission (i.e. load) in the elasticity cloud services by means of priority value. The LBRR algorithm reduce the memory utilization and to solve the optimization problem by adding dynamic priority framework into the system. Based on the active priority concept, the priority value in the Prioritized Load Balanced Round Robin framework is estimated.

![Figure 4.2 Block diagram of Dynamic Priority Load Scheduler](image-url)

**Figure 4.2 Block diagram of Dynamic Priority Load Scheduler**
The active priority concept decreases the memory utilization through load scheduling by discovers the highest and the next highest priority CPUs for simple scheduling in the cloud mobile environments. Figure 4.2 explains the block diagram of Dynamic Priority Load Scheduler.

As exposed in above figure 4.2, the active priority concept in dynamic priority load scheduling algorithm is used to identify the highest and the next highest priority CPU to be allocated with load in cloud mobile environments.

Evaluating the priority value, Dynamic Priority Load Scheduler in DP-LBRR framework schedules the mission for each customer request in the elasticity cloud services. The active priority concept by using priority value is mathematically formalized as given below.

$$\text{Priority Value} = \sum_{i=1}^{n} \text{MAX}(CPU_i), \text{MAX}(CPU_{i+1})$$

......... Eqn (4.2)

In DP-LBRR framework, a Dynamic Priority Load Scheduler uses dynamic priority for the CPUs based on which the VMs are scheduled. It schedules the VMs to the CPUs depending upon their priority value, which varies in a dynamic manner on the origins of their load value in cloud mobile devices. This dynamic priority load scheduling produced better utilization of the CPUs. The Load Scheduler has to schedule a Virtual Machine (VM) when a user request from different cloud mobile environments appears. The load scheduler verifies for the maximum resource CPU and discovers out
whether its priority has been allocated with the purpose of improving the scheduling efficiency. If the highest resource CPU has not been found, then it is establish earliest.

\[ \text{MCPU} = \sum_{i=1}^{n} \text{MAX} (\text{CPU}_i) \]

\[ \text{......... Eqn (4.3)} \]

The CPU acquiring highest resource is recognized and then it is ensured whether the CPU has a load value less than threshold factor \( \tau \). The explanation after setting the threshold factor is to preserve a CPU from being congested. Ahead successful identification of the CPU, it is allocated with the highest priority and the VM is assigned to that consequent CPU. If the highest priority has a load value better than \( \tau \), then it verifies if the next highest CPU has been recognized. If it has been recognized and if its load value is less than \( \tau \), then the VM is scheduled to that CPU and the search for the next highest resource CPU with the load value less than \( \tau \) is routed.

The current highest resource CPU is allocated to earlier highest resource CPU and the earlier highest resource CPU is allocated the next highest priority when before allocating the new CPU as the highest priority. Below algorithm shows the Dynamic Priority Load Scheduling algorithm.

**Input:** Request \( \text{Request}_i = \text{Request}_1, \text{Request}_2, ..., \text{Request}_n \), Load Scheduler \( \text{Scheduler}_S \), CPU \( \text{CPU}_i = \text{CPU}_1, \text{CPU}_2, ..., \text{CPU}_n \) Virtual Machine \( \text{VM}_i = \text{VM}_1, \text{VM}_2, ..., \text{VM}_n \), threshold factor \( \tau \)

**Output:** Optimized memory consumption

**Step 1:** Begin
Step 2: For each user request Req_i

//Perform priority based load scheduling
Step 3: Identify maximum resource CPU
Step 4: If load (CPU_i) < \tau
Step 5: Assign (CPU_i) with highest priority
Step 6: Assign (VM_i) to the corresponding (CPU_i)
Step 7: End if
Step 8: If load (CPU_i) > \tau
Step 9: Identify next maximum resource CPU
Step 10: Assign (CPU_i) with highest priority
Step 11: End if
Step 12: End for
Step 13: End

Algorithm 4.2 Dynamic Priority Load Scheduling algorithm

The above algorithm 4.2 explains the CPU with maximum priority for each user request is assigned using Dynamic Priority Load Scheduling algorithm. Continued by this, the next highest resource CPU having the next higher priority and recognized and VM is then assigned with the related CPU. This is performed in an iterative manner until all the users request from cloud mobile environments are capably programmed.
4.4. EXPERIMENTAL EVALUATION OF DP-LBRR FRAMEWORK

DP-LBRR framework is designed and the experiments were conducted on CloudSim simulator environment. The CloudSim simulator executed on Cloud environment provides different resource patterns for a number of virtual machines. Each virtual machine is constructed with an exact amount of memory, CPUs, and local storage. The DP-LBRR framework is prepared with two quad core 2.33-2.66 GHz Xeon processors (8 cores total), 7 GB RAM, and 1690 GB local disk storage.

The DP-LBRR framework with efficient data calculation and information distribution using Amazon, a well-known and widely recognized cloud service provider simulates the dynamic benchmarking technique using on-claim cloud services.

The DP-LBRR framework employs the Standard Small EC2 instance (m1.small) and High CPU EC2 instances (cl.medium and cl.xlarge) to build the cloud computing environment. This helps to recover CPU load scheduling efficiency and reduce the memory utilization during load scheduling.

It is compared with existing methods namely, Semi Elastic Cluster method introduced by Shuangcheng Niu et al. (2013) and NOCOG developed by Shaoyi Song et al. (2014). The performance of the DP-LBRR framework is measured in terms of:

i) CPU Load Rate

ii) Latency Time

iii) Memory Consumption

iv) Load Scheduling Efficiency
4.5. PERFORMANCE ANALYSIS OF DP-LBRR FRAMEWORK

The DP-LBRR framework is compared against the two existing methods. The compared existing methods are namely, Semi Elastic method by Shuangcheng Niu et al. (2013) and NOCOG by Shaoyi Song et al. (2014). To evaluate the DP-LBRR framework, the following metrics are used.

4.5.1. Performance analysis of CPU Load Rate

The performance analysis of CPU load rate is measured with the product of user request based on the CPU load in cloud environment. It gives the amount of load leaking out from the cloud environment when a number of user requests are located in cloud. It is measured in terms of hertz (Hz) and mathematically formulated as given below.

\[ \text{CPU}_{L} = \sum_{i=1}^{n} \text{Req}_{i} \times \text{CPU load} \]

\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldOTS
Table 4.1 Tabulation for CPU load rate

<table>
<thead>
<tr>
<th>User requests</th>
<th>CPU load rate (Hz)</th>
<th>Existing SEC</th>
<th>Existing NOCOG</th>
<th>Proposed DP-LBRR</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.9</td>
<td>1.2</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>1.1</td>
<td>1.5</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>1.3</td>
<td>1.8</td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>1.6</td>
<td>1.9</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1.9</td>
<td>2.3</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>2.1</td>
<td>2.5</td>
<td>1.7</td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>2.5</td>
<td>2.8</td>
<td>1.9</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>2.7</td>
<td>3.1</td>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>2.9</td>
<td>3.2</td>
<td>2.3</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>3.2</td>
<td>3.5</td>
<td>2.6</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1 show the analysis of CPU load rate with respect to number of user request being placed ranging between 20 and 200 Cloud paradigms and it is measured in terms of hertz (Hz). The proposed DP-LBRR Method with different number of user request is taken for experimental purpose using java language. The performance of proposed DP-LBRR Method is compared with existing two methods namely, Semi Elastic Cluster and Non co-operative game among users and a cooperative game among processors. From the table value, it is illustrative that the CPU load rate using DP-LBRR Method is reduced as compared to the other existing methods.
Figure 4.3 Measure of CPU load rate

Figure 4.3 explains the CPU load rate efficiency with different number of user requests being made in the cloud environment. As illustrated in above figure, the proposed DP-LBRR method performs relatively well when compared to two other methods namely Semi Elastic Cluster and Non co-operative game among users and a cooperative game among processors.

Besides, while increasing the number of user request, the CPU load rate also gets increased, but comparatively the CPU load rate using DP-LBRR method is significantly minimum than the other methods. With the application of Load Balanced Round Robin, the load scheduler allocates the load to the CPU in an efficient manner by turning off the virtual machines which are not currently used.
This in turn operates all the resources (i.e. load) in a balanced manner by not wait for the exhaustion of the resources for each user request. This in turn decreases the CPU load rate using proposed DP-LBRR method by 30% when compared with SEC method by Shuangcheng Niu et al. (2013) and 59% when compared with NOCOG algorithm by Shaoyi Song et al. (2014) respectively.

4.5.2. Performance analysis of Latency Time

Latency time is defined as time interval between the user requests being made and the response from the user (i.e. the CPU assigned with the Virtual Machines). Latency time is referred as the delay time. The mathematical calculation for latency time is as given below.

\[
\text{Latency Time} = \sum_{i=1}^{n} \text{Time (Req}_i\text{)} \ast \text{Time (response)}
\]

…… Eqn (4.5)

From equation (4.5) the latency time ‘LT’, is measured by considering the requests being made ‘Req\_i’ and the response ‘response’ for the corresponding request. It is measured in terms of millisecond (ms).
Table 4.2 Tabulation for Latency Time

<table>
<thead>
<tr>
<th>User requests</th>
<th>Latency time (ms)</th>
<th>Existing SEC</th>
<th>Existing NOCOG</th>
<th>Proposed DP-LBRR</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td></td>
<td>0.86</td>
<td>0.69</td>
<td>0.52</td>
</tr>
<tr>
<td>40</td>
<td></td>
<td>0.95</td>
<td>0.83</td>
<td>0.67</td>
</tr>
<tr>
<td>60</td>
<td></td>
<td>1.06</td>
<td>0.91</td>
<td>0.74</td>
</tr>
<tr>
<td>80</td>
<td></td>
<td>1.21</td>
<td>0.99</td>
<td>0.86</td>
</tr>
<tr>
<td>100</td>
<td></td>
<td>1.28</td>
<td>1.13</td>
<td>0.93</td>
</tr>
<tr>
<td>120</td>
<td></td>
<td>1.35</td>
<td>1.21</td>
<td>1.11</td>
</tr>
<tr>
<td>140</td>
<td></td>
<td>1.52</td>
<td>1.34</td>
<td>1.13</td>
</tr>
<tr>
<td>160</td>
<td></td>
<td>1.58</td>
<td>1.45</td>
<td>1.29</td>
</tr>
<tr>
<td>180</td>
<td></td>
<td>1.78</td>
<td>1.53</td>
<td>1.38</td>
</tr>
<tr>
<td>200</td>
<td></td>
<td>1.93</td>
<td>1.78</td>
<td>1.52</td>
</tr>
</tbody>
</table>

Table 4.2 illustrates the comparison of latency time using DP-LBRR Method with existing SEC method and NOCOG algorithm. In order to measures the latency time, user request is considered in the range 20 to 200. The performance of proposed DP-LBRR Method is compared with existing two methods namely, Semi Elastic Cluster and Non co-operative game among users and a cooperative game among processors. From the table value, it is descriptive that the latency time using proposed DP-LBRR Method is reduced as compared to other existing methods.

Below figure 4.4 shows the measure of latency time using the proposed DP-LBRR method. As illustrated in above figure, the proposed DP-LBRR method performs
relatively well when compared to two other methods namely Semi Elastic Cluster and Non co-operative game among users and a cooperative game among processors.

![Figure 4.4 Measure of latency time](image)

**Figure 4.4 Measure of latency time**

Besides, while increasing the number of user request, the latency time also gets increased, but comparatively the latency time using DP-LBRR method is significantly minimum than the other methods. For example, user request is considered as 20, latency time attained as 0.86ms and 0.69ms in existing SEC and NOCOG methods, while in proposed DP-LBRR method 0.52ms of latency time is obtained. By applying Load Balanced Round Robin algorithm, the Load Scheduler turns on and the Virtual Machine is turned off in repeated manner to avoid the CPU from being overloaded. This in turn aids in reducing the latency time of DP-LBRR by 36% when compared to SEC method by Shuangcheng Niu et al. (2013) and 18% when compared with NOCOG algorithm by Shaoyi Song et al. (2014) respectively.
4.5.3. Performance analysis of Memory Consumption

Memory consumption is calculated using free memory, buffer memory and cached memory with total memory. It is defined as the difference between total memory and sum of free, buffer and cached memory. The mathematical formulation of memory utilization is as given below.

\[
\text{Memory Consumption} = \text{Total memory} - (\text{Free} + \text{Buffer} + \text{Cached})
\]

……. Eqn (4.6)

From (4.6), the memory utilization is calculated in terms of megabytes (MB). Lower the memory consumption more efficient the method is said to be.

**Table 4.3 Tabulation for memory consumption**

<table>
<thead>
<tr>
<th>Number of tasks</th>
<th>Memory consumption (MB)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Existing SEC</td>
<td>Existing NOCOG</td>
<td>Proposed DP-LBRR</td>
</tr>
<tr>
<td>5</td>
<td>85</td>
<td>69</td>
<td>58</td>
</tr>
<tr>
<td>10</td>
<td>90</td>
<td>79</td>
<td>65</td>
</tr>
<tr>
<td>15</td>
<td>104</td>
<td>91</td>
<td>79</td>
</tr>
<tr>
<td>20</td>
<td>112</td>
<td>94</td>
<td>84</td>
</tr>
<tr>
<td>25</td>
<td>125</td>
<td>107</td>
<td>92</td>
</tr>
<tr>
<td>30</td>
<td>132</td>
<td>119</td>
<td>104</td>
</tr>
<tr>
<td>35</td>
<td>140</td>
<td>125</td>
<td>110</td>
</tr>
<tr>
<td>40</td>
<td>147</td>
<td>133</td>
<td>122</td>
</tr>
<tr>
<td>45</td>
<td>155</td>
<td>142</td>
<td>128</td>
</tr>
<tr>
<td>50</td>
<td>169</td>
<td>156</td>
<td>138</td>
</tr>
</tbody>
</table>
Table 4.3 shows the memory consumption using DP-LBRR Method, and SEC method and NOCOG algorithm. The DP-LBRR framework calculates the transmission delay on cloud environment to decrease the CPU load with respect to different number of tasks. It is calculated in terms of Mega Bytes (MB). The memory utilization for scheduling CPU load using proposed DP-LBRR framework is reduced as compared to other existing methods.

![Figure 4.5 Measure of memory consumption](image)

Figure 4.5 Measure of memory consumption

Figure 4.5 submits the characteristic of memory consumption with respect to number of task used to calculate CPU load rate. The result offered in figure 4.6 verifies that the proposed DP-LBRR framework significantly outperforms the other two methods namely Semi Elastic Cluster method (SEC) by Shuangcheng Niu et al. (2013) and NOCOG by Shaoyi Song et al. (2014). The memory consumption is decreased in the DP-LBRR framework using the Dynamic Priority Load Scheduling algorithm. By the
implementation of dynamic priority scheduling algorithm with their function, priority value is measured. Based on this value, highest priority CPU is identified and the next highest priority CPU is identified for each user requests. Continued by this, based on the dynamic priority, the CPU is then allocated with virtual machine in cloud mobile devices. This dynamic mixture of priority in DP-LBRR framework in turn reduces the memory consumption by 30% when compared to SEC method by Shuangcheng Niu et al. (2013) and 14% when compared with NOCOG algorithm by Shaoyi Song et al. (2014) respectively.

4.5.4. Performance analysis of Load Scheduling Efficiency

The active priority finds the highest and the next highest priority nodes for easy scheduling in the cloud mobile environments. In order to measure the scheduling efficiency rate, 20 – 200 user requests with 50 tasks were used. Scheduling efficiency is measured in terms of percentage (%).

Table 4.4 Tabulation for load scheduling efficiency

<table>
<thead>
<tr>
<th>User requests</th>
<th>Load scheduling efficiency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Existing SEC</td>
</tr>
<tr>
<td>20</td>
<td>57.36</td>
</tr>
<tr>
<td>40</td>
<td>59.21</td>
</tr>
<tr>
<td>60</td>
<td>61.82</td>
</tr>
<tr>
<td>80</td>
<td>62.03</td>
</tr>
<tr>
<td>100</td>
<td>62.87</td>
</tr>
<tr>
<td>120</td>
<td>63.45</td>
</tr>
<tr>
<td>140</td>
<td>65.11</td>
</tr>
<tr>
<td>160</td>
<td>66.84</td>
</tr>
<tr>
<td>180</td>
<td>68.32</td>
</tr>
<tr>
<td>200</td>
<td>69.21</td>
</tr>
</tbody>
</table>
Table 4.4 shows the load scheduling efficiency using DP-LBRR Method, and SEC method and NOCOG algorithm. For the purpose of experimental evaluation, user request is considered in the range of 20 to 200. From the table value, it is descriptive that the load scheduling efficiency using proposed DP-LBRR Method is increased as compared to other existing methods.

![Figure 4.6 Measure of load scheduling efficiency](image)

**Figure 4.6 Measure of load scheduling efficiency**

Figure 4.6 shows the scheduling efficiency for calculating CPU load in cloud environment using three different methods versus various user requests. From the figure it is clear that the scheduling efficiency rate using DP-LBRR framework is comparative increased than the other two existing methods namely Semi Elastic method (SEC) by Shuangcheng Niu et al. (2013) and NOCOG by Shaoyi Song et al. (2014). With the application of Dynamic Priority Load Scheduler the scheduling efficiency rate is enhanced in DP-LBRR framework. Based on the priority value VMs are scheduled to the
CPU by applying the Dynamic Priority Load Scheduler. This dynamic mixture of priority in DP-LBRR framework in turn increase the scheduling efficiency rate by 15% when compared to SEC method by Shuangcheng Niu et al. (2013) and 8% when compared with NOCOG algorithm by Shaoyi Song et al. (2014) respectively.

4.6. SUMMARY

DP-LBRR framework is constructed based on the LBRR algorithm. The LBRR algorithm in DP-LBRR framework executing various types of user requests from different cloud mobile environments by allocating each virtual machine in a cyclic order for reducing the latency time. This framework enhances the CPU load scheduling efficiency and reduces the memory utilization through load scheduling in cloud elasticity environment. The framework uses a mechanism of LBRR algorithm in a dynamic manner it balances the CPU load rate during efficient addressing of user request using Load Scheduler. Executing the Dynamic Priority Load Scheduler with LBRR algorithm in DP-LBRR framework, decreases the memory utilization and therefore improving the scheduling efficiency in an efficient way between successive user requests. Future, Dual-Cost Responsive on Cloud Mobile Services (DRCMS) Mechanism is implemented for improving the elasticity on cloud services. The dual cost responsive algorithm with extensive results verifies the higher flexibility rate in cloud mobile devices.