International Journal of Engineering Trends and Technology
ISSN: 2231-5381/ https://doi.org/10.14445/22315381/IJETT-V74I11P108

Volume 74 Issue 11, 118-127, November 2026
© 2026 Seventh Sense Research Group®

Original Article

Need based and Race Condition Free Load Balancing in Cloud Computing Systems

Narayan A. Joshi

Department of MCA, Dharmsinh Desai University, Gujarat, India.

Corresponding Author: narayan.joshi.mca@ddu.ac.in

Received: 23 May 2026 Revised: 12 September 2026 Accepted: 18 September 2026 Published: 25 November 2026

Abstract - *Reliance on cloud computing has become important part of everyday life. To leverage the most out of cloud services, the demand for cloud-based computing solutions is growing fast. However, the growing demands have rendered the process of management of cloud resources crucial. Load balancing systems often face challenges in providing a fair distribution of resources amid fluctuating workload conditions of cloud instances. Workloads on underloaded and overburdened cloud instances with varying degrees of supply and demand should be balanced in a manner that avoids race conditions and imbalance. This paper puts forward a need based mechanism for workload balancing, where resource allocation is directed by quantified need in line with the dynamically weighted priority levels of instances. The proposed technique implements synchronization among concurrently running load balancer threads. Proposed mechanism enhances conventional balancing practices by sharing workload between the complementary overloaded and underloaded instances. Execution behavior of the recommended solution was examined on an open-source private cloud platform prototype. The experimental results demonstrate a reduction in the occurrence of instantaneous underloading and overloading, accompanied by a greater number of load balancing operations initiated. These findings offer a foundation for future investigation into the recommended mechanism's potential to minimize wastage of cloud resources. The recommended technique can be studied further on intercloud environments.*

Keywords - *Cloud computing, Need based load balancing, Priority-based workload sharing.*

1. Introduction

During the past decade, contemporary innovations in the Information & Communication technology sector have significantly accelerated the need for and development of cloud computing technology-based computing solutions. Moreover, the innovations have not only transformed how cloud-based computing solutions are designed and deployed but also have influenced various societal sectors. Consequently, the world is witnessing upward integration of cloud technology-based computing solutions into routine life, enhancing user interactivity and engagement. On the other hand, the growing reliance on cloud technology compels the management and distribution of cloud resources crucial and challenging.

There exist numerous techniques for distribution of workload and resources in cloud computing environments. However, a need for a robust and priority based enhanced mechanism for workload distribution is essential for not only optimized utilization of cloud resources but also service level agreements aspects. This paper presents an improved mechanism for optimized distribution of cloud resources in cloud computing environments. Section 2 of this paper discusses about various load balancing algorithms available for cloud computing environments. Section 3 represents the proposed enhanced mechanism for balancing workload in cloud technology-based computing environments. The execution behavior of the proposed mechanism was examined on a prototype private cloud platform. The Section 4 describes the deployment scenario. Experimental observations are presented in Section 5. Section 6 concludes the paper by outlining future expansion possibilities.

2. Literature Review

A dynamic workload distribution genetic algorithm works on exploring load of virtual machine servers. For load distribution, the algorithm looks on searching the optimal VM configuration for cloud servers [1].

A scheduling algorithm featuring a coding method based on product integration works on improving genetic approach with reference to the matrix of process constraint [2]. The algorithm addresses constraints relevant to processing sequence and introduces a technique for addressing sub-processes and families. Through the constraint matrix, the algorithm derives a hierarchical matrix representing the relationship. The technique introduces a genetic algorithm for family oriented single parent relationship.

This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

A technique [3] based on VM cloning and VM migration aims to offer highly available virtual machines on a proactive basis. The technique reports improved efficiency in replication and reduction in the active hosts for enhanced utilization of resources. On the basis of host resources utilization, the technique dynamically monitors replication settings at runtime. Load balancing measures involve supervising server workload, avoiding server underutilization and overbearing, shifting of workloads. The productivity and scalability of the load sharing algorithms in extensive cloud computing platforms are acutely essential to safeguard optimized utilization of cloud resources. The extensibility parameter covers effective organization of inbound requirements arising through multiple data centers. The sender originated, and energy effective technique works in reducing the makespan [4].

Moreover, a sudden rise and fall in traffic also play vital role in turning overloaded machines into an idle VM. Vice versa, such a sudden fall and rise in traffic also play a crucial role in turning underloaded machines into overbearing machines [5].

A genetic algorithm [6] attempts to address load balancing and job scheduling by means of live migration of dense servers to substitute machines. Results obtained using the CloudSim simulator indicated an increased number of virtual machine migrations and a decrease in energy consumption.

A job scheduling approach [7] works on processing events such as overuse and underuse for virtual machines in a cloud data center. The approach works on decreasing the number of efforts taken to maintain the necessary performance levels.

An autonomic system for management of cloud resources [8], Golondrina, works on CPU stress-based resource management in a cloud computing platform. The technique keeps exploring situations where the combined resource needs of virtual servers exceed the available resources in the underlying physical machine. The technique replicates such virtual servers to other physical machines. The technique claims replication of virtual servers in place of migration of virtual servers.

A cloud resource usage monitoring solution [9] works on monitoring various resources of a cloud virtual machine. The tool can be availed for tracking the consumption of various running VM resources, such as memory and CPU. The mechanism works as a utility of the open-source cloud platform OpenNebula [10].

A workload sharing technique [11] represents a dynamic scheduling algorithm. The technique works on efficiently fetching information from distributed data centers. The mechanism discusses applications of the round robin, weighted least connection and weighted divisible round robin algorithms for implementation of load balancing.

A workload distribution algorithm [12], while distributing workload, takes the priority of virtual machines in consideration. The technique distributes workload among the same priority virtual machines. Moreover, the technique may be further improvised for avoiding race conditions in the state of virtual machines that are participating in load balancing.

A deep learning-based hybrid algorithm for load sharing works through the amalgamation of recurrent neural networks and convolutional neural networks [13]. The hybrid algorithm works on computing workload parameters for cloud instances. The clustering-based workload balancing approach aims addressing various Quality of Service variables such as task prioritization, memory, CPU utilization, energy reduction and makespan minimization. Ensuring adaptation to highly dynamic workloads while preserving scalability in extremely large cloud infrastructures remains a noted concern.

A weighted priority driven workload balancing solution aims on reducing starvation. The technique is built on open source cloud computing environment set up on open source operating system. The runtime-adaptive load balancing mechanism dynamically computes the weighted priority values of cloud instances [14]. However, the mechanism can be further extended to account for race conditions while enabling complementary workload distribution.

An evaluation-oriented load distribution algorithm designed to simplify resource allocation by scoring the server capabilities works by capacity-based scoring policy. The scoring mechanism introduces a method to determine server potential on the basis of processing speed and primary memory. Its centralized scoring policy relates the high resource jobs with high score hosts and the low resource jobs with low score hosts. The algorithm focuses on simulation-based evaluation framework [15]. Furthermore, artificial intelligence-based techniques for crop analysis [16] and driver drowsiness detection [17] are likely to achieve improved performance when deployed in cloud environments augmented with load-balancing capabilities.

A throttled mechanism for workload distribution [18] functions dynamically. The dynamic mechanism aims to increase the throughput and decrease the processing time in cloud datacenter environments. A component-oriented workload management method [19] works for the cloud datacenters that are distributed geographically. The throttled technique incorporates three additional components: VM holder, VM manager and VM reader. The method focuses on simulation-based evaluation framework. The method works on optimization of overall performance, processing time and response time.

An improved throttled approach for balancing workload [20] aims on optimizing the processing time and the response time. In order to select the idle virtual machines, the approach

incorporates consumption index and the prioritized table of index. The approach focuses on simulation-based evaluation framework. An enhanced approach [21] based on dynamic distribution of workloads in cloud systems focuses on functional efficiency and advancing performance metrics, including makespan. The approach follows service level agreements and dynamically determines the ideal cloudlet for workload distribution. A simulation-based multi-phase scheduler [22] works on efficient distribution of workload in a cloud system. The scheduler employs a pipeline comprising five phases, such as resource allocation, parallel task scheduling, load monitoring, load prediction and workload distribution.

Various existing workload balancing mechanisms operate workloads' distribution between overburdened and underburdened cloud instances by means of transferring workloads from the overburdened machines to the underburdened machines. Nevertheless, the uncontrolled distribution of workload often transforms the overburdened cloud instances into an underutilized condition. Vice versa, the undisciplined distribution of workload seldom transforms the underloaded cloud instances into an overloaded condition. Furthermore, there exists a possibility of the occurrence of a race condition among the cloud instances that are participating in workload-balancing operations in a cloud computing environment.

On other side, some of the load distribution algorithms entail relocating the whole virtual machines between the idle cloud servers and overburdened cloud servers. To ascertain fair workload distribution, certain workload balancing methods handle all cloud machines evenly, scheduling workloads at the same priority level for each. These algorithms do not discriminate cloud instances based on priority levels.

Simulation-driven workload balancing algorithms, on the other hand, focus on reducing the cost factors and making spans. However, the simulation-oriented systems build upon the preexisting awareness about the actual temporal information associated with the arrival and execution of jobs in order to load balance workload in a real-world cloud environment. Nevertheless, existing algorithms offer limited insights into disciplined and unbiased load-balancing approaches. The issue of race conditions among virtual machines in cloud-based workload sharing systems remains poorly addressed, thereby highlighting a substantial scope of deeper examination.

3. Methodology

The workload distribution algorithm [12] addresses workload balancing by means of shifting workload from an overcrowded cloud instance to an idle cloud instance in a prototype cloud compute environment. For avoiding starvation, the algorithm takes cloud instances' static priority levels into consideration. The workload is balanced between the cloud machines, having the same priority value. However, the use of fixed priority levels inherently increases latency, consequently degrading system performance.

Also, the priority grounded algorithm [5] for distribution of workload from crowded machine to idle machine functions in line with the priority levels of cloud instances in an inter cloud compute environment. The algorithm relies on static priority levels of cloud instances. The load distribution is executed between the instances at equal priority levels. The mechanism operates in line with predefined fixed priority levels assigned to participating cloud machines in workload sharing operations. To mitigate starvation, workloads are distributed across machines with different priority levels. Nevertheless, dependence on fixed prioritization may introduce higher latency, consequently reducing system performance. Whereas the dynamic priority grounded algorithm [14] transfers workload from crowded machines to idle machines in a cloud environment. Initially, machines are assigned an appropriate static priority level. At runtime, the algorithm keeps computing a dynamically weighted priority level using respective machine's static priority level and weight factor. The proposed enhancements to these algorithms facilitate more efficient and equitable allocation of cloud resources. Accordingly, the methodology introduces specific algorithmic refinements aimed at optimizing resource distribution while ensuring fairness, eliminating bias, and preventing race conditions during load balancing.

The proposed resource-sharing mechanism balances workload among cloud instances by offloading tasks from resource constrained instances to underutilized ones. To ensure fairness in workload distribution, the proposed load balancer mechanism NBCloudLB, performs need based task offloading among cloud instances with complementary workloads. Moreover, to eliminate race conditions among concurrently running daemon threads of execution, the proposed technique employs mutual exclusion mechanism over the shared algorithmic data structures. Hence, to facilitate concurrent execution by means of mutual exclusion, the respective thread functions have been implemented through an intrinsic synchronization approach employed to synchronize their execution. The major operational steps of the proposed load balancing mechanism are presented below:

- The proposed mechanism, `NBCloudLB`, retains the priority levels and cloud instance state values defined in [14]. In addition, the enhanced algorithm introduces an additional cloud instance state, namely `UPSHIFT`. At a time, the load balancing state of the cloud instance could be any one of the following.
- Let each instance of the data structure `CInstance` carry a profile of the computational resources available on a respective cloud instance, including operational workload,

priority, dynamic weight, IP address and various timestamps.

- Moreover, within the `CInstance` data structure, the `NBCloudLB` maintains a weight parameter for each cloud instance. This parameter, combined with the instance's static priority level, facilitates starvation-free workload distribution. Accordingly, the `NBCloudLB` mechanism dynamically updates the weight value at runtime.
- To enable efficient workload redistribution from overcrowded machines, the suggested `NBCloudLB` mechanism continuously executes a dedicated daemon thread, denoted as `thread_findOLCInstance`. This thread detects overcrowded cloud instances based on their workload state and sets them up for distribution of their workload. The state information of such machines is maintained within designated priority queues, allowing the `NBCloudLB` load balancer to dequeue an overcrowded machine and reallocate its workload to a suitably underutilized virtual machine.
- To exploit unused computational power in cloud instances, the proposed `NBCloudLB` mechanism continuously executes a dedicated daemon thread, denoted as thread_findULCInstance, to identify underutilized cloud instances. The state information of such instances is maintained in designated priority queues. Upon selection of an underloaded instance from the queue, the `NBCloudLB` mechanism coordinates workload distribution from the corresponding overburdened virtual machines to achieve balanced workload distribution.
- A dedicated load balancer thread, denoted as `thread_NBLB`, dynamically performs need-based balanced workload distribution between overburdened and underutilized machines within the cloud environment. From the designated priority queues, `thread_NBLB` extracts an overloaded machine and an appropriately underloaded machine exhibiting complementary workloads. The thread `thread_NBLB` prevents the underburdened machine from transitioning to overburdened state abruptly. Moreover, to prevent abrupt state transition from overloaded to underloaded conditions, `thread_NBLB` temporarily inhibits further workload distribution requests from CI_{OL}. Conversely, `thread_NBLB` temporarily restricts further workload transfer requests directed to the underloaded instance CI_{UL} to prevent rapid escalation to an overloaded state. Consequently, `thread_NBLB` temporarily transitions CI_{UL}'s load-balancing availability state to passive.
- In addition to `thread_NBLB`, a separate daemon thread `thread_CIStateManager`, continuously updates the state information of machines within the corresponding `CInstance` data structure. It periodically monitors the current state and available resources if cloud machines marked as `PASSIVE`, and conditionally transitions them as `ACTIVE`, enabling their participation in forthcoming workload redistribution operations.
- Figure 1 illustrates the schematic block diagram of the proposed need-based workload distribution mechanism, `NBCloudLB`, for cloud computing platforms.

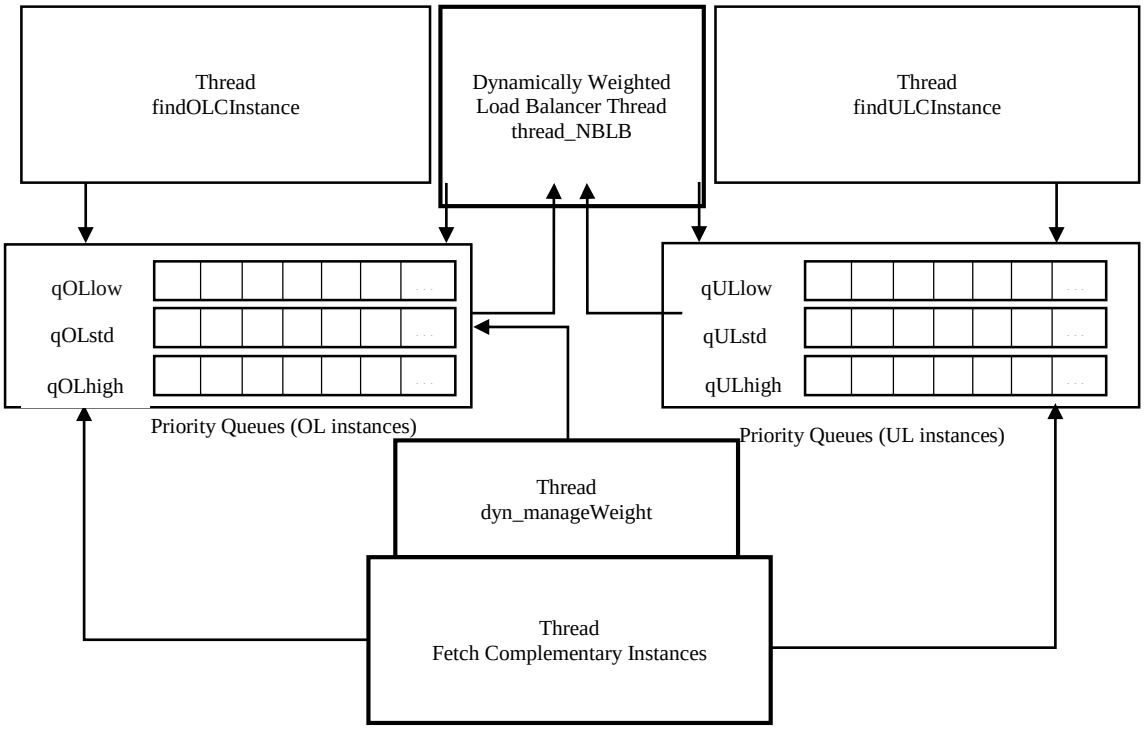

Fig. 1 Schematic diagram of proposed need-based workload balancer mechanism

- The key design aspects of the proposed NBCloudLB mechanism are outlined below:
  1. The proposed load redistribution module, NBCloudLB, maintains information about each cloud instance in a data structure named CInstance. The CInstance structure stores cloud machine-specific details, including the current machine state and resource inventory metadata.

```
struct CInstance {
  int CIstate;
  float operational_load;
  short priority;
  float dyn_weight;
  char IP[16];
  float passive_tm, max_passive_tm;
} //struct
```

  2. At any given time, a cloud machine may be in one of the following load-balancing states [14]:

     UNAVAILABLE: A cloud instance which is not presently active is assigned the status UNAVAILABLE. Additionally, an active cloud instance that is set not to participate in workload balancing is also assigned the status UNAVAILABLE

     The NBCloudLB workload distributor prevents cloud machines that have recently participated in a load-balancing operation from participating repeated workload distribution operations in quick succession. Subsequently, the timestamp indicating when a cloud machine last participated load balancing is stored in the machine-specific passive_tm field within the CInstance data structure.

     An active cloud machine that is available to participate in workload balancing and has a passive_tm value of zero represents that it is not presently engaged in any workload balancing activities. A cloud machine meeting these conditions, being available for workload balancing, is labelled READY in the state field of the CInstance.

     The NBCloudLB continuously tracks the workload-balancing state of cloud machines. A Cloud machine that has received supplementary cloud resources through NBCloudLB is assigned the OL_PASSIVE_LB status value in its corresponding data structure CInstance for a short interval known as max_passive_tm. Consequently, if the passive_tm duration of an overloaded OL_PASSIVE_LB cloud machine is less than max_passive_tm, the NBCloudLB workload balancer moves it to the OL_PASSIVE state.

     The recommended workload distributor, NBCloudLB, continuously identifies cloud machines that are open to receive load-balancing requests. An underutilized cloud machine that is assigned an incoming load balancing request is set to the UL_PASSIVE_LB status value in its corresponding data structure CInstance for a short interval known as max_passive_tm. Consequently, if the passive_tm duration of an underutilized UL_PASSIVE_LB cloud machine is less than max_passive_tm, the NBCloudLB workload balancer moves it to the UL_PASSIVE state.

  3. Moreover, a concurrently running thread of execution thread_CIStateManager supervises and updates the cloud machine's load balancing state information maintained in the CInstance. [14]

```
thread_CIStateManager {
  CInstance CI;
  …
 if CI->passive_tm == 0;
   CI->state = READY;
 if CI_OL is awaiting CI resources,
   CI_OL->state = OL_PASSIVE_WAITING;
 if CI_OL is permitted, workload sharing,
   CI_OL->state = OL_PASSIVE_LB;
 if CI_OL->state == OL_PASSIVE_LB
   if CI_OL->passive_tm >= max_passive_tm
     CI_OL->state = READY;
   else
     CI_OL->state = OL_PASSIVE;
    if CI_UL is awaiting incoming LB
requests,
      CI_UL->state = UL_PASSIVE_WAITING;
 if CI_UL is assigned loadsharing request,
   CI_UL->state = UL_PASSIVE_LB;
 if CI_UL->state == UL_PASSIVE_LB
   if CI_OL->passive_tm >= max_passive_tm
     CI_UL->state = READY;
   else
     CI_UL->state = UL_PASSIVE;
  …
}
```

  4. The NBCloudLB maintains the following priority queues for holding information about the underutilized and overutilized cloud machines:
     - QUL<high>: A priority queue that stores CInstance data structures about the underutilized cloud machines with high priority.
     - QOL<high>: A priority queue that stores CInstance data structures about the overutilized cloud machines with high priority.
     - QUL<std>: A priority queue that stores CInstance data structures about the underutilized cloud machines with standard priority.
     - QOL<std>: A priority queue that stores CInstance data structures about the overutilized cloud machines with standard priority.
     - QUL<low>: A priority queue that stores CInstance data structures about the underutilized cloud machines with low priority.

- QOL<low>: A priority queue that stores CInstance data structures about the overutilized cloud machines with low priority.

5. A thread thread_findOLCInstance synchronously finds the overloaded machines in the cloud:

```
synchronous thread_findOLCInstance {
  CInstance* CI_OL = NULL; // OL machine
  while(true) {
      CI_OL = findOLCInstance();
      if(CI_OL->CIstate in
        [OL_PASSIVE, OL_PASSIVE_LB,
         OL_PASSIVE_WAITING])
        continue;
      // append CI_OL to relevant OL queue
  if(CI_OL)
    Q_OL<high,low,std>.append(CI_OL);
} // while
}
```

6. A thread thread_findULCInstance synchronously finds the overloaded machines in the cloud:

```
synchronous thread_findULCInstance {
  CInstance* CIUL = NULL; // UL machine
  while(true) {
      CIUL = findULCInstance();
      if(CIUL->CIstate in
      [UL_PASSIVE, UL_PASSIVE_LB,
       UL_PASSIVE_WAITING])
        continue;
      // append CIUL to relevant UL queue
  if(CIUL)
    QUL<high,low,std>.append(CIUL);
} // while
}
```

7. Let the synchronous thread thread_fetch_complementary_instances determine the complementarily loaded overburdened and under burdened cloud instances CI_{OL} to the CI_{UL}.

```
// Synchronously get the complementarily
    loaded instances
synchronous
    thread_fetch_complementary_instance
    s {
   // return if the queues are empty
  if (QOL is empty or QUL is empty)
     return (null, null);
       for OLMachine in QOL {
                  //   ignore   if   OL
    machine's LB state is passive
         if(!(OLMachine.isOverloaded()
            ||

    OLMachine.CIstate==OL_PASSIVE_WAITI
    NG
     ||
OLMachine.CIstate == OL_PASSIVE))
    continue; // ignore such machine now
    for LB
 else  {  //  Yes,  an  OL  machine  is
    available for LB
    CInstance OLMachine = QOL.next();
    break;
        // found OL machine for LB, so
    move on to
        // find complementary UL machine
      }
    } // for
         // find surplus overload
  OLsurplus_load = OLMachine->
                operational_load - 100;

        // fetch relevant complementary
    UL machine
    diff[] = OLsurplus_load -
             QUL->operational_load;
    CInstance ULMachine = null;
    ULMachine = QUL[diff.min().index()];
          // not having PASSIVE state.
       return (OLMachine,ULMachine);
} //function
```

8. The NBCloudLB load balancer consists of a concurrently executing daemon thread thread_NBLB. The thread synchronously keeps identifying the most nearly complementarily loaded overburdened and under burdened cloud machines named as CI_{OL} and CI_{UL}, respectively. The thread_NBLB executes workload distribution from the CI_{OL} to the CI_{UL}.

```
    synchronous thread_NBLB {
      CInstance* CIOL, CIUL;
      …
      while(true) {
        CIOL, CIUL =
     fetch_complementary_instances();
    if(CIOL == null or CIUL == null)
      continue;
        QOL<high,std,low>.remove(CIOL);
        QUL<high,std,low>.remove(CIUL);
        CIOL->CIstate = OL_PASSIVE_LB;
        CIUL->CIstate = UL_PASSIVE_LB;
       CIOL->passive_tm                =
    current_time_stamp;
        CIUL->passive_tm                =
    current_time_stamp;
        load_balance(CIOL, CIUL);
 …
   } //while
} //thread_NBLB
```

9. Moreover, suggested load balancer NBCloudLB, helps evade starvation by implementing the concurrent thread of execution thread_dynManageWeight. Given the initial weight value dyn_weight, and the priority priority value, inside the CInstance object of respective cloud instance, the thread dynamically computes and manages the runtime value of dyn_weight.

```
synchronous thread_dynManageWeight
{
  …
  if(CIOL->CIstate==
    OL_PASSIVE_WAITING)
     CIOL->dyn_weight =
```

```
        (CIOL->priority      *      CIOL-
    >dyn_weight
         * 0.25) + 1;
if(CIOL->dyn_weight                      >
    dynWeightThreshold)
     CIOL->dyn_weight = 0;
 UPSHIFT CIOL to apt priority-queue
for ongoing LB task
   }// thread_dynManageWeight
```

10. The proposed mechanism incorporates the `launch()` function to initiate the `NBCloudLB`, the load-balancing module. The `launch()` function prepares various data structures, accompanied by launching various threads. The `launch()` function:
    - Prepares `CInstance` data structures for all cloud instances that are partaking in load balancing.
    - Prepares various priority queues: $Q_{OL<high,std,low>}$ and $Q_{UL<high,std,low>}$.
    - Initializes various time variables: `passive_tm` and `max_passive_tm`.
    - Sets the `WeightThreshold`, `passive_tm` and `max_passive_tm` data structures.
    - Initializes various load-balancing state constants.
    - Sets up running concurrently running threads: `thread_NBLB`, `thread_CIStateManager`, `thread_fetch_complementary_instances`, `thread_dynManageWeight`, `thread_findULCInstance` and `thread_findOLCInstance`.

4. Implementation

To evaluate the operational performance of the proposed `NBCloudLB` workload balancer technique, a prototype implementation was deployed on a pilot private cloud computing environment. An architype Infrastructure-as-a-Service (IaaS) platform was implemented and deployed within a private cloud platform. The Infrastructure-as-a-Service was deployed using a freely accessible OpenStack [23] cloud computing environment hosted on a Fedora 34 Linux server [24] developed by the open-source community. The performance of the proposed load balancer `NBCloudLB`, was evaluated by comparing cloud machine workloads in scenarios with and without `NBCloudLB`. A baseline set of results was first obtained without employing any load-balancing mechanism. The experiments were subsequently repeated using the existing load balancer mechanisms and the proposed `NBCloudLB` load balancer mechanism, respectively.

5. Results and Discussion

The recommended workload balancer `NBCloudLB` commences operation by setting up the `CInstance` data structures for all cloud machines. It subsequently maintains six priority queues to manage the load balancing information about underutilized and over utilized cloud machines with different priority levels. The mechanism subsequently activates the background threads: thread_NBLB, thread_CIStateManager, thread_fetch_complementary_instances, thread_dynManageWeight, thread_findULCInstance and thread_findOLCInstance.

Table 1. Count of LB requests processing initiated

Number of cloud instances available	Count of LB requests processing initiated using		
	A. existing load balancer	B. existing load balancer	C. proposed `NBCloudLB` load balancer
10	4	6	7
20	9	13	16
30	12	15	19
40	15	17	24
50	21	30	39

Table 1 summarizes the experimental observations on the average number of workload balancing requests operated on the cloud computing platform. The measurements were obtained for both the existing workload balancer mechanisms and the proposed workload balancing mechanism.

Column A reports the number of workload balancing requests handled by the existing load balancer algorithm [12]. Column B reports the number of workload balancing requests handled by the existing load balancer algorithm [14], while Column C reports the number of workload balancing requests processed by the proposed `NBCloudLB` workload balancing mechanism.

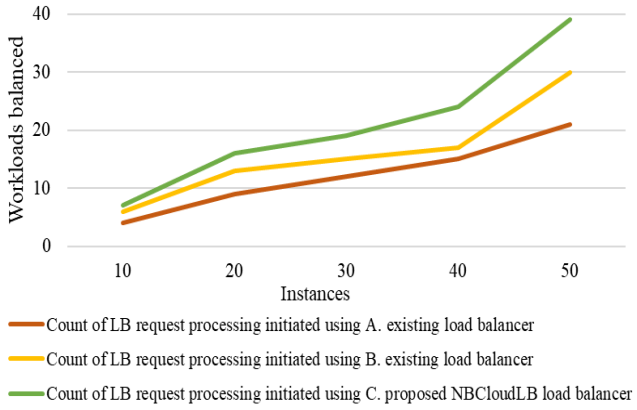

Fig. 2 Comparative analysis of workload balancers: Count of LB requests processing initiated

Figure 2 depicts the comparative performance of the existing and the proposed load balancer mechanisms with respect to the number of workload balancing requests handled. The results indicate that the proposed workload balancing mechanism delivers greater workload balancing performance.

Table 2 summarizes the experimental observations on the number of instantaneous overburdening observed post workload balancing on the cloud computing platform. The measurements were obtained for both the existing workload balancer mechanisms and the proposed workload balancing mechanism. Column A reports the number of instantaneous overburdenings observed while practicing the existing load balancer. [12] Column B reports the number of instantaneous overburdenings observed while practicing the existing load balancer [14], while Column C reports the number of instantaneous overburdenings observed using the proposed `NBCloudLB` workload balancing mechanism.

Table 2. Count of instantaneous overburdening

Number of cloud instances available	Count of instantaneous overburdening observed using		
	A. existing load balancer	B. existing load balancer	C. proposed `NBCloudLB` load balancer
10	7	5	3
20	15	12	7
30	19	16	13
40	29	24	16
50	35	27	24

Figure 3 depicts the comparative performance in terms of the number of instantaneous overburdening observed while experimenting the existing and the proposed load balancer mechanisms. The results indicate that the proposed workload balancing mechanism delivers higher workload balancing performance in terms of the instantaneous overburdening.

Table 3 summarizes the experimental observations on the number of instantaneous under burdening observed post workload balancing on the cloud computing platform. The measurements were obtained for both the existing workload balancer mechanisms and the proposed workload balancing mechanism. Column A reports the number of instantaneous overburdening observed while practicing the existing load balancer. [12] Column B reports the number of instantaneous under burdening observed while practicing the existing load balancer [14], while Column C reports the number of instantaneous under burdening observed using the proposed `NBCloudLB` workload balancing mechanism.

Table 3. Count of instantaneous under burdening

Number of cloud instances available	Count of instantaneous underburdening observed using		
	A. existing load balancer	B. existing load balancer	C. proposed `NBCloudLB` load balancer
10	6	3	2
20	13	10	5
30	17	13	11
40	23	19	12
50	32	28	21

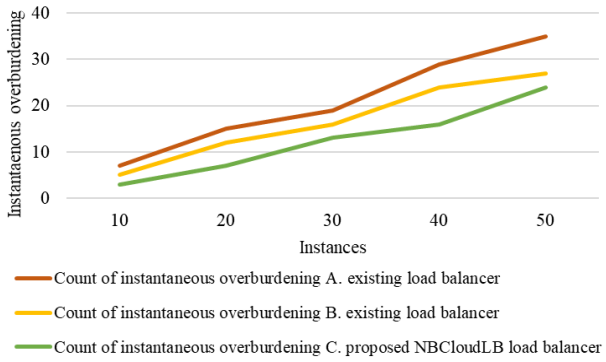

Fig. 3 Comparative analysis of workload balancers: Number of instantaneous overburdening

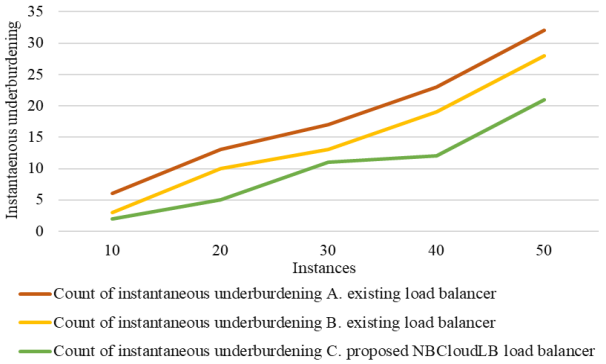

Fig. 4 Analysis of workload balancers: Number of instantaneous under burdening

Figure 4 depicts the comparative performance in terms of the number of instantaneous under burdening observed while experimenting the existing and the proposed load balancer mechanisms. The results indicate that the proposed workload balancing mechanism delivers higher workload balancing performance in terms of the instantaneous under burdening.

Figure 2 points out that performance improves with an increase in the number of workload balancing requests initiated by the proposed NBCloudLB technique. Figure 3 implies that the recommended workload balancer mechanism helps decrease the number of instantaneous overburdenings of cloud virtual machines, leading to improved performance. Figure 4 indicates that the NBCloudLB also brings down the number of instantaneous under burdening of cloud virtual machines, resulting in enhanced efficiency. Therefore, the proposed NBCloudLB technique achieves higher productivity and improved end-user satisfaction by mitigating abrupt load imbalances and enhancing mobile resource utilization.

Hence, the recommended cloud-resource distribution method accommodates workloads among cloud instances

through offloading jobs from overladen machines to the underladen instances. To safeguard fairness in workload allocation, the recommended workload allocation mechanism NBCloudLB, executes need driven task redistribution among virtual machines having complementary loads. The proposed load balancer approach incorporates synchronization mechanisms to prevent race conditions while processing concurrent access to the shared data structures.

6. Conclusion

This paper introduced an enhanced, dynamically weighted, priority-driven load balancing mechanism, termed NBCloudLB, for efficient workload sharing in cloud computing systems. The proposed approach significantly enhances resource distribution by maximizing the utilization of underutilized cloud resources and enabling adaptive workload redistribution between overloaded (starving) and dynamically identified underloaded machines.

As a result, the mechanism effectively reduces latency and mitigates abrupt load fluctuations. Experimental results demonstrate that NBCloudLB achieves superior performance in terms of throughput, responsiveness, and the number of successfully processed workload balancing requests when compared with existing approaches. These findings highlight the effectiveness and scalability of the proposed mechanism in dynamic cloud environments. Future research may consider enhancing the proposed load balancing framework by incorporating robust security mechanisms and addressing licensing constraints associated with cloud resource sharing.

References

[1] Zhen Wang et al., "Dynamic Load Balancing of Virtual Machine Servers using Genetic Algorithm in a Cloud Computing Environment," *2025 4th International Conference on Artificial Intelligence, Internet of Things and Cloud Computing Technology (AIoTC)*, Guilin, China, pp. 776-780, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[2] Chao Qian, and Jianxin Zhang, "An Integrated Scheduling Genetic Algorithm based on Process Constraint Matrix and Family Definition Coding," *2025 3rd International Conference on Materials Science and Intelligent Manufacturing*, *Journal of Physics: Conference Series*, vol. 3009, pp. 1-7, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[3] Younes Khair, Haouari Benlabbes, and Mourad Belabed, "Proactive VMs Replication for High Availability in OpenNebula Cloud Environments," *Computing*, vol. 107, no. 6, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[4] Parvaz Ahmad Malla et al., "Energy-Efficient Sender-Initiated Threshold-based Load Balancing (e-STLB) in Cloud Computing Environment," *Concurrency and Computation: Practice and Experience*, vol. 36, no. 5, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[5] Narayan A. Joshi, "Rational Load Balancing in Collaborated Cloud Computing Environments," *Engineering Research Express*, vol. 6, no. 2, pp. 1-14, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[6] Samira Rezapourshahir, and Shahram Babaie, "A New Load-Balancing Approach for Cloud Computing based on an Improved Genetic Algorithm," *The Journal of Supercomputing*, vol. 81, no. 9, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[7] Younes Khair, and Haouari Benlabbes, "Opportunistic Load Balancing for Virtual Machines Scheduling in a Cloud Environment," *Engineering Proceedings*, vol. 29, no. 1, pp. 1-6, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[8] Gaston Keller, and Hanan Lutfiyya, "Dynamic Resource Management in Virtualized Environments through Virtual Server Relocation," *International Journal on Advances in Software*, vol. 3, no. 3-4, pp. 333-350, 2010. [Google Scholar] [Publisher Link]

[9] Younes Khair, Abdeslem Dennai, and Youssef Elmir, "Dynamic and Elastic Monitoring of VMs in Cloud Environment," *Journal of Supercomputing*, vol. 78, no. 17, pp, 19114-19137, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[10] The Open Platform for Sovereign Cloud Infrastructure, OpenNebula, 2025. [Online]. Available: https://opennebula.io/

[11] Santosh T. Waghmode, and Bankat M. Patil, "Adaptive Load Balancing in Cloud Computing Environment," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 11, no. 1s, pp. 209-217, 2023. [Google Scholar] [Publisher Link]

[12] Narayan A. Joshi, "Starvation Free and Priority Oriented Resource Sharing in Cloud Computing Environment," *AIP Conference Proceedings*, vol. 2796, no. 1, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[13] Ahmad Raza Khan, "Dynamic Load Balancing in Cloud Computing: Optimized RL-based Clustering with Multi-Objective Optimized Task Scheduling," *Processes*, vol. 12, no. 3, pp. 1-23, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[14] Narayan A. Joshi, "Designing and Testing Dynamically Weighted Load Balancer for Cloud Computing Environment," *IAENG International Journal of Computer Science*, vol. 53, no. 2, pp. 808-818, 2026. [Publisher Link]

[15] Sunil Tekale, J. Gladson Maria Britto, and A.S. Gousia Banu, "Load Balancing in Cloud Computing," *International Journal of Engineering and Advanced Technology*, vol. 8, no. 6S3, pp. 2164-2166, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[16] Peiqian Wu, and Yuxin Zhong, "Artificial Intelligence in Sustainable Agriculture: Towards a Socio-Technical Roadmap," *Smart Agricultural Technology*, vol. 12, pp. 1-11, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[17] Zakwan AlArnaout et al., "Exploiting Heart Rate Variability for Driver Drowsiness Detection using Wearable Sensors and Machine Learning," *Scientific Reports*, vol. 15, no. 1, pp. 1-24, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[18] Ramana Reddy B, and Indiramma M, "Efficient Throttled Load Balancing Algorithm to Improve the Response Time and Processing Time in Data Center," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 23, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[19] Dawit Mekonnen et al., "Designing a Component-based Throttled Load Balancing Algorithm for Cloud Data Centers," *Mathematical Problems in Engineering*, vol. 2022, no. 1, pp. 1-12, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[20] Hanumanth Rao Panuganti, and Rajakumar Subramanian, "Enhanced Throttled Load Balancing for Virtual Machine Allocation in Multiple Data Centers," *Scalable Computing: Practice and Experience*, vol. 25, no. 5, pp. 3453-3467, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[21] Raiymbek Zhanuzak et al., "Optimising Cloud Computing Performance with an Enhanced Dynamic Load Balancing Algorithm for Superior Task Allocation," *IEEE Access*, vol. 12, pp. 183117-183132, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[22] Nasser S. Albalawi, "Dynamic Scheduling Strategies for Cloud-based Load Balancing in Parallel and Distributed Systems," *Journal of Cloud Computing*, vol. 14, no. 1, pp. 1-25, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[23] The Most Widely Deployed Open Source Cloud Software in the World, OpenStack Website, 2026. [Online]. Available: https://www.openstack.org/

[24] It's Your Operating System, The Fedora Project. [Online]. Available: https://fedoraproject.org/