Journal of Distributed Computing and Systems (JDCS)
Vol.8, Issue 1, 2025, Pages 71-84

Performance-Aware Hybrid Throttled-ESCE Load Balancing with Two-Stage Load Threshold Mechanism for Heterogeneous Cloud Environments

Ali Yousefi Choubini *
* Department of Computer Engineering, Faculty of Engineering and Technology, Shahid Ashrafi Esfahani University, Isfahan, Iran
*a.yousofi@ashrafi.ac.ir (alternative: alijosephy2019@gmail.com)

Article History:
Received: 21 May 2025
Received in revised form: 29 July 2025
Accepted: 18 August 2025
Available online: 17 September 2025

Abstract

Load balancing (LB) is a critical determinant of performance and efficiency in cloud computing environments. While various algorithms like Throttled Load Balancing (TLB) and Equally Spread Current Execution (ESCE) exist, they suffer from inherent limitations such as inefficient resource utilization, a lack of capacity-awareness, and the risk of Virtual Machine (VM) overload, leading to suboptimal performance. This paper introduces a novel hybrid LB method that synergistically combines the core principles of the TLB and the ESCE algorithms. The proposed method employs a sophisticated two-stage load threshold mechanism to intelligently distribute incoming tasks. It systematically prioritizes VMs by first selecting from idle pools based on processing power, then from partially loaded VMs using a capacity-weighted score, and finally from a broader pool based on historical throughput efficiency, all while strictly enforcing a load threshold to prevent VM overload. By explicitly accounting for heterogeneous VM capacities (MIPS, CPU cores, RAM) and real-time efficiency metrics, the method ensures a more balanced and intelligent workload distribution. Comprehensive simulations conducted using the CloudAnalyst simulator demonstrate the superior efficacy of proposed approach. The results show a significant 44.97% average improvement in Data Center (DC) processing time and a 23.38% reduction in average response time to clients compared to traditional algorithms (ESCE and TLB). These findings conclusively establish that the proposed method effectively mitigates the limitations of its predecessors and offers a robust, high-performance solution for enhancing QoS in modern heterogeneous cloud DCs.

Keywords: Cloud Computing, Hybrid Load Balancing, ESCE, Throttled, CloudAnalyst Simulator Tool.

I. INTRODUCTION

With the rapid growth in Information Technology, cloud computing has emerged as a vital paradigm for providing services on a pay-per-use basis, enabling universal access to a pool of configurable resources such as servers, storage, and networks [1]. Efficiently managing these resources is critical, particularly through LB, which ensures equal and dynamic distribution of workloads to maximize resource utilization and user satisfaction [2, 3]. However, the LB problem remains a significant challenge; in multi-node environments, nodes frequently experience extreme workload imbalances where some become overloaded while others remain inactive [4]. Addressing this inefficiency is essential for minimizing response times and enhancing system performance, making it a pivotal area for advancing cloud infrastructure reliability.

The existing landscape of LB is divided into Static (SLB) and Dynamic (DLB) algorithms [1, 5, 6]. While SLB methods like Round Robin (RR) and Min-Min rely on prior system knowledge, DLB algorithms like TLB and ESCE adapt to the system's state in real-time [3, 7]. Despite their popularity, these algorithms possess inherent flaws: TLB often fails to consider current processing capacities during VM scanning [8], while ESCE relies solely on load values without accounting for the complexities of task allocation [9]. Recent hybrid attempts have tried to bridge these gaps, such as Agrawal and Nigam's combination of TLB and ESCE [10], Le and Tran's Improved Throttled Algorithm (ITA) [8], and component-based approaches by Mekonnen et al. [11]. However, these studies often overlook heterogeneous VM capacities (MIPS, RAM) or lack the sophisticated preventive thresholds found in more complex predictive models [12, 13].

To address these limitations, this paper proposes a novel hybrid LB methodology (using TLB and ESCE algorithms) designed for heterogeneous cloud environments. The proposed method integrates a two-stage adaptive threshold mechanism with a multi-phase VM selection process. By moving beyond simple load-based selection, the methodology utilizes a comprehensive scoring system that evaluates VMs based on their real-time availability and inherent processing power. This approach ensures that task allocation is not only balanced in terms of quantity but is also optimized for the specific hardware capabilities of each node, thereby preventing the bottlenecks common in traditional DLB strategies.

The main contributions of this study are threefold: (1) the development of a two-stage adaptive threshold mechanism that dynamically prevents VM overload to maintain system stability; (2) the design of a multi-phase selection process that prioritizes resources based on idle capacity, processing power, and historical efficiency; and (3) the implementation of an integrated scoring system that accounts for heterogeneous VM capabilities (MIPS, CPU cores, and RAM). These contributions collectively enhance the novelty

of the work by providing a systematic selection process that maintains operational simplicity while significantly improving resource allocation efficiency in real-time cloud deployments.

The remainder of this paper is organized as follows: Section II provides a Literature Review of the cloud computing and LB domain. Section III explores Related Works and identifies existing research gaps. Section IV details the Proposed Method, including the hybrid algorithm and scoring mechanism. Section V describes the Experimental Setup and Simulation environment. Section VI presents the Analysis and Discussion of the experimental results. Finally, Section VII concludes the paper with a Discussion and Future Works.

II. Literature Review

This section provides a concise survey of traditional LB algorithms, detailing their core mechanisms, and succinctly outlining their principal strengths and limitations.

1. *Round Robin Load balancing (RR)*

RR method serves as one of the widely used LB methods [14]. The RR algorithm is the same as the first-come-first-served algorithm [9]. RR algorithm works in a circular and ordered procedure where each process is assigned a fixed time slot without any priority [3]. The RR technique does not take into consideration resource capabilities, priority, or assignment time. Slower response times are caused by longer jobs and higher priority [7]. Figure 1 provides an architectural design of the RR.

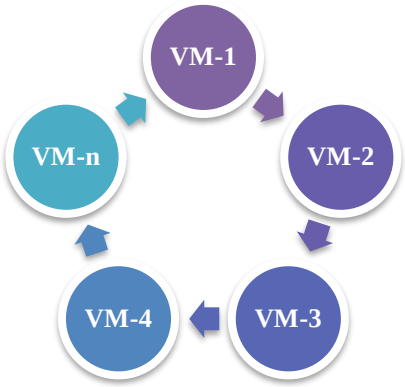

Figure 1. RR load balancer [15].

2. *Throttled Load balancing (TLB)*

The Throttled load balancer is one of the approaches for balancing workflows in dynamic environments [16] As can be seen in Figure 2 below, the purpose of the load balancer is to search for a suitable VM to perform tasks upon receiving a request from a client [3]. In the approach, first, the user will request the load balancer to find the most appropriate VM to execute the task. The TLB algorithm balances the load by maintaining a configuration table of VMs and their status. When a request is required to allocate in VMs from the DC, the load balancer chooses the VM which is first found in the list of available VMs [8]. If a VM is available and has enough space, the task is accepted and allocated to the VM. If no available VM is found, then TLB returns -1 otherwise the request is queued for fast processing. TLB performs better than the RR algorithm, however, it doesn't consider more advanced requirements for LB such as Processing Time [3].

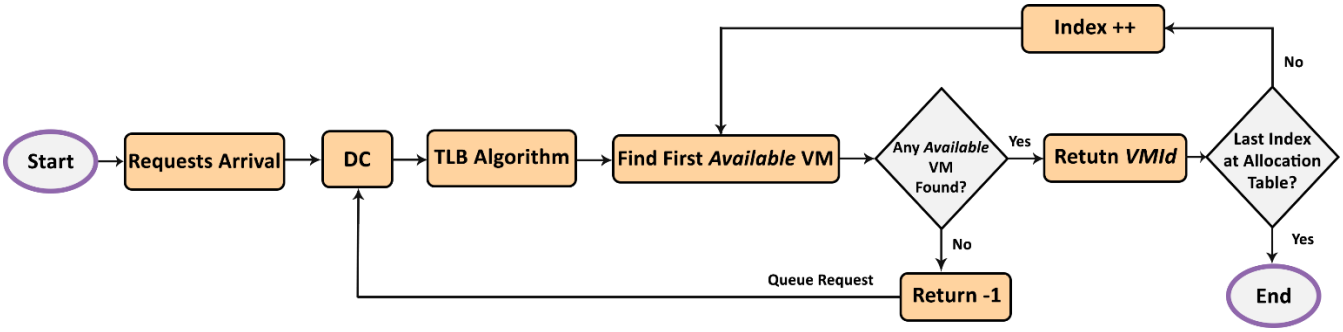

Figure 2. TLB algorithm (adopted from [3]).

3. *Equally Spared Current Execution (ESCE)*

This algorithm uses the concept of a distributed spectrum and operates in such a form that the number of available activities in each VM is equal at any moment [9]. ESCE focuses on spreading the load across different nodes, earning it the name of a spread spectrum technique. By distributing the load evenly, the algorithm ensures that no specific VM becomes overloaded while others remain underutilized [7]. A common problem of ESCE is that it could cause overhead when updating the index table due to the communication that occurs between the DCC and the load balancer [3].

4. *Other Common Load Balancing Algorithms*

There are many other LB algorithms, some of the other most common LB algorithms in cloud computing are: Min-Min (MM), Weighted RR (WRR), Ant Colony Optimization (ACO) [3]. MM algorithm begins with a set of tasks that are initially not assigned to any of the nodes. For all the accessible nodes, the least completion time is first considered.

The job with the shortest anticipated completion time is then chosen and assigned to the node with the shortest execution time, as a result, the MM algorithm improves if the smaller work is larger than the smaller task [17]. The WRR server load balancer is a SLB since it distributes incoming traffic without changing the state of the servers. The WRR is meant to function with predetermined weights and jobs that are dispersed based on the weights. Processors with stronger capability are assigned a higher value [14]. The motivation behind ACO algorithm is based on the behavior of ants during their hunt for food. Ants travel randomly searching for food and when they return, it disposes an amount of chemicals known as pheromone. Such amount determines the shortest path from the nest to the food source for the other ants to follow. Although this approach is good for resource optimization, however, it results in slow Response Time and performance [3].

III. Related Works

Agrawal and Nigam (2021) in [10] proposed a hybrid load balancer that combines Throttled and ESCE techniques. The proposed hybrid approach uses a VM state list for availability and a hash-map for load tracking, enabling efficient initial allocation and dynamic load redistribution. This hybrid method demonstrates superior performance, achieving lower response times and reduced complexity compared to its constituent algorithms.

In [8], Le and Tran (2022) introduced the Improved Throttled Algorithm (ITA), an advanced LB method for cloud environments that enhances the classic Throttled approach. Instead of selecting the first available VM, ITA employs a dynamic priority index based on recent usage (Makespan) to make more intelligent assignments. The algorithm manages two index tables to track available and busy VMs, updating them in real-time as tasks are assigned and completed. This core mechanism of consistently allocating new tasks to the least-used available VM allows ITA to optimize system performance by significantly reducing response times, minimizing idle resources, and enhancing overall cost efficiency.

Mekonnen et al. (2022) in [11], proposed component based Throttled algorithm. They enhanced the Throttled algorithm by decoupling its load balancer into three specialized modules: a VM reader for tracking status, a free VM holder for aggregating available resources, and a manager that randomly assigns requests. This modular design reduces per-request overhead, cutting response and processing times while improving VM utilization.

In [18], In 2023, Zamri et al., proposed an enhancement to the ESCE algorithm by integrating it with an Optimize Response Time brokerage policy. The proposed method involved modelling the ESCE algorithm using the CloudAnalyst simulator to analyze its performance under various configurations, including different DC locations and VM quantities, with the aim of improving its load distribution capabilities in a cloud environment.

In [19], in 2023, Sharma et al., proposed a hybrid LB algorithm that integrates the features of the ESCE and the Double Priority algorithms to enhance cloud resource efficiency. The joint approach first prioritizes and schedules tasks based on their length and VM capacity using a dual-priority mechanism to minimize starvation and response time. It then employs the ESCE strategy to uniformly redistribute the load among all available VMs, thereby preventing any node from being underloaded or overloaded. The authors demonstrated through simulation in CloudAnalyst that the joint approach achieves a significant reduction in overall response time compared to using the individual algorithms.

According to [12], Rashid and Nakpih (2024) introduced ReT-ELBa, a rule-based and content-aware throttled load balancer. It dynamically manages cloud resources by monitoring VM workloads and task characteristics in real time. The system organizes VMs into three pools: a primary pool (all VMs), an engaged pool (busy VMs), and a secondary pool (idle backups). Routing decisions are governed by two thresholds: 70% for overall system utilization and 50% for the proportion of large jobs. When both limits are exceeded, ReT-ELBa diverts smaller tasks to the secondary pool. This strategy reduces response time while improving utilization over traditional TLB and RR algorithms.

Akinola et al. (2024) proposed an intra-balancer in [20] that adapts the Throttled algorithm for LB within individual DCs of a federated cloud. The system maintains a live VM-status table, scans it for the first available VM, and assigns incoming requests accordingly; if all VMs are busy, requests are enqueued. This lightweight, dynamic approach minimizes decision overhead while ensuring even task distribution. Implemented in CloudAnalyst, it demonstrated lower response time than RR and ESCE.

In [14], Priya and Rajendran (2025) proposed the Enhanced Weighted Round Robin (EWRR) algorithm, a dynamic extension of WRR that continuously adapts VM allocation shares using real-time inputs like CPU capacity, instantaneous load, task length, and queue depth. During operation, EWRR polls all VMs, recalculates their weights based on current processing capability, and dispatches tasks following the WRR sequence. If a VM exceeds a predefined load threshold, surplus tasks are redirected to less-utilized VMs. This strategy combines feedback-driven weight tuning with targeted task migration to balance resources and prevent overload.

In [13], Matiwure et al. (2025), introduced an ML-Enhanced Throttled Weighted Round Robin load balancer. It employs a gradient boosting regressor to predict the optimal server for each task based on task features and server-state data. This prediction is then validated against real-time queue length and estimated processing time. If the chosen server is overloaded, the system falls back to a Throttled WRR algorithm to select the next-best host. This hybrid policy balances predictive accuracy with runtime robustness, aiming to minimize response and completion times while reducing queue lengths and improving throughput.

Table 1 compares the proposed method with other modern algorithms to highlight its unique niche. Unlike existing logic, our method uses a two-stage load threshold L_1, L_2) and a multi-phase hierarchy (Idle, Partial and High Load). This makes it fully capacity-aware and heterogeneity-aware, as it normalizes hardware metrics (MIPS, CPU Cores and RAM) to prioritize the most capable resources in diverse cloud environments.

Table 1. Comparative Analysis of Decision Logic and Feature Awareness

Algorithm	Year	Decision Logic	Capacity Awareness	Heterogeneity Awareness	Load Threshold Mechanism	Selection Hierarchy
ITA [8]	2022	Dynamic priority based on least recent usage (Makespan).	Low (based on usage history).	Moderate (usage-based).	No.	History-based priority.
ESCE & Double Priority [19]	2023	Tasks scheduled by length/capacity then redistributed via ESCE.	Moderate (VM capacity for scheduling).	Moderate.	No. (Preventative distribution only).	Starvation prevention via dual-priority.
ReT-ELBa [12]	2024	Rule-based routing to primary/secondary pools	Content-aware (task size focus).	Moderate (three-pool system).	Single-stage (70% utilization).	Utilization-based backup pool.
EWRR [14]	2025	Dynamic weight recalculation based on CPU and queue.	High (CPU, load, queue).	High (Weight-based).	Single-stage (redirection threshold).	Feedback-driven weight tuning.
ML-Enhanced Throttled WRR [13]	2025	Predictive regressor model for server selection	High (Server-state and estimated processing time)	High (Machine learning based on server features)	Fallback logic for overloaded servers	ML-predicted best host with Throttled WRR backup
Proposed Method	**2026**	**Multi-Phase Hierarchical Selection**	**Full (MIPS, CPU cores, RAM).**	**Full (Normalizes hardware scores).**	**Two-Stage Dynamic Threshold (L_1, L_2)**	**Sequential Fallback: Idle, Partial and Highly Loaded, Queue.**

IV. METHODOLOGY

To overcome the limitations of both the TLB and ESCE algorithms, this paper introduces a novel hybrid method. The principal contribution is a synergistic integration of the TLB algorithm's state aware VM management with ESCE's distributed load spreading strategy, augmented by a multi-phase VM selection procedure controlled by adaptive, dynamic load thresholds. This design preserves the rapid availability checks of TLB scheduling and the global dispersion benefits of ESCE while adding processing capacity and efficiency aware decision logic to prevent VM overload and improve overall service latency and throughput.

The primary components of the proposed method are designed to overcome specific shortcomings of its constituent algorithms. The standard TLB algorithm utilizes a binary (Available/Busy) state and a simplistic top-down index search, which fails to consider the heterogeneous computational capacity of VMs, often leading to sub-optimal assignments. Conversely, while ESCE effectively distributes load, it does not intrinsically account for the processing power of individual VMs, risking the overloading of weaker VMs. The proposed method elegantly resolves these issues through a two-stage load threshold mechanism.

This mechanism prevents VMs from being overloaded by setting a strict maximum capacity limit. By explicitly factoring in VM capacity (including MIPS, CPU cores, and RAM) and real-time efficiency metrics, the method intelligently distributes the load, thereby significantly reducing the average processing time and response time for user requests. The proposed architecture comprises a dynamic filtering mechanism that operates in four sequential phases. In Phase 1, the system detects idle VMs. In Phase 2, it assesses partially loaded VMs (L_1) and computes their efficiency scores. In Phase 3, the mechanism selects and leverages highly utilized VMs (L_2) according to the calculated efficiency metric (throughput). Finally, in Phase 4, if all VMs exhibit heavy load, the load balancer issues a '-1' signal to the DCC to queue incoming requests.

The system utilizes two distinct load thresholds, L_1 and L_2, where L_2 and L_1 are natural numbers and $L_1 < L_2$ ($\{L_1, L_2 \in \mathbb{N} \mid 1 \leqslant L_1 < L_2\}$). This two-stage mechanism intelligently distributes tasks to busy VMs by categorizing them according to their current load (L_i), ensuring that VMs receive new requests only when they have sufficient capacity, thereby optimizing performance and preventing overload.

1. *Phase 1: Selection of Idle VMs*

The initial phase of the proposed selection hierarchy focuses on the immediate activation of underutilized resources within the cloud cluster. By prioritizing VMs with zero current allocations (L_i=0, state='Available'), the algorithm ensures that idle capacity is brought into service before adding load to already active nodes. This strategy minimizes the initial response latency for new tasks and prevents high-performance hardware from sitting dormant while other resources are burdened, establishing a clean baseline for efficient workload distribution across the heterogeneous environment.

For these idle resources, the selection priority is determined by a baseline processing power metric ($Score_1$). Since the VM has no active tasks to impede its performance,

the score is calculated solely based on its inherent hardware capabilities, such as MIPS and RAM. This allows the load balancer to rank the available VMs, ensuring that the most powerful idle resource is always the first to be engaged. This prioritized entry point serves as the foundation for the multi-stage hierarchy, transitioning the system from simple availability to complex, capacity-aware management as the global load increases.

$$Score_1\ (v_i) = MIPS_i * PEs_i \quad (1)$$

In Equation (1), $MIPS_i$ shows processing power of VM *i* and PEs_i shows number of the processing elements of VM *i*.

2. *Phase 2: Capacity-Aware Selection of partially Loaded VMs*

If no VMs are idle ($L_i > 0$), the method proceeds to its second phase, evaluating busy VMs with a current allocation count (L_i) below the medium load threshold ($L_i < L_1$, state='Busy'). For these VMs, a comprehensive score is calculated, which balances normalized processing capacity (MIPS, RAM) against current load (L_i) to identify the most suitable candidate, as formalized in Equation (2).

$$Score_2\ (v_i) = \frac{w(\frac{MIPS_i}{MIPS_{max}} * PEs_i) + (1-w)(\frac{RAM_i}{RAM_{max}})}{L_i + \varepsilon} \quad (2)$$

where:

- $\{w \in \mathbb{R} \mid 0 \leq w \leq 1\}$: which controls the weighting between CPU and memory resources.
- $MIPS_i$: MIPS rating of VM *i*.
- $MIPS_{max}$: maximum MIPS among all VMs (keeps MIPS normalized).
- PEs_i: number of virtual processing elements of VM *i*.
- RAM_i: RAM of VM *i*.
- RAM_{max}: maximum RAM among all VMs (keeps RAM normalized).
- L_i: number of concurrent tasks on VM *i*.
- ε: very small value to ensure numerical stability for newly added VMs.

The capacity-weighted score ($Score_2$) serves as a dynamic metric to evaluate VM strength relative to its current workload. As defined in Equation (2), the score is directly proportional to hardware specifications, including MIPS and RAM, adjusted by a weighting factor *w*. This ensures that in a heterogeneous cluster, more powerful VMs receive a higher baseline priority, allowing the system to leverage superior hardware for more intensive tasks.

Conversely, the score is inversely proportional to the number of active tasks (L_i), acting as a natural regulator to prevent VM congestion. As a VM accumulates more tasks, its $Score_2$ decreases, which automatically shifts the load balancer's preference toward other capable but less-burdened resources. This mathematical relationship balances raw processing power with real-time availability, ensuring no single node becomes a bottleneck while maintaining high throughput across the environment.

3. *Phase 3: Efficiency-Aware Selection of Highly Loaded VMs*

When the overall system load escalates and no VMs are available in the idle or partially loaded pools (Phases 1 and 2), the algorithm transitions to Phase 3. This stage is triggered when all candidate VMs have exceeded the primary load threshold L_1 but remain below the critical upper limit L_2. At this high-load intensity, the decision logic shifts from simple capacity-based selection to an efficiency-driven approach. The objective here is to distinguish between VMs that are busy but fast and those that are busy and slow, ensuring that incoming tasks are routed to the most productive resources even under stress.

$$Score_3\ (v_i) = \frac{N_i}{T_i} \quad (3)$$

where:

- N_i: total number of completed tasks on VM *i*.
- T_i : total active (busy) time accumulated for VM *i* in millisecond (ms).

In this phase, the selection is governed by a specialized Efficiency Score ($Score_3$) as shown in Equation (3). This metric integrates the VM's raw processing power (MIPS) with its real-time throughput efficiency. The logic assumes that in a heterogeneous environment, a high-capacity VM processing many tasks may still be more efficient than a low-capacity VM with fewer tasks. By normalizing the historical throughput against the maximum processing capability of the cluster, the algorithm assigns a dynamic rank to each VM. This allows the load balancer to identify the workhorse VMs that consistently demonstrate higher execution rates, thereby preventing the common tail-latency problem where tasks get stuck on slower nodes during peak traffic.

4. *Phase 4: Prevent Overloading*

The final stage of the proposed allocation mechanism functions as a robust admission control system to ensure global cloud stability. When the workload surpasses the critical high-load threshold (L_2) across all active VMs, the system prioritizes architectural integrity over immediate execution. At this saturation point, every VM in the heterogeneous environment is already operating at its maximum efficient capacity; therefore, assigning additional tasks would lead to exponential increases in response times and potential resource starvation. By transitioning to this phase, the algorithm explicitly prevents performance degradation and ensures that the system does not violate

Service Level Agreement (SLA) commitments during peak traffic periods.

When this saturation state is identified, the load balancer returns a symbolic '-1' status to the DCC, triggering a back-pressure mechanism. This signal instructs the DCC to halt immediate distribution and redirect incoming requests into a prioritized waiting queue. This strategic delay allows the heavily loaded VMs to complete their current execution cycles without the overhead of excessive context switching that typically occurs under extreme stress. By enforcing these strict load boundaries, the proposed hybrid method ensures that average response times remain predictable and optimized, directly contributing to the superior processing efficiency demonstrated in experimental results.

5. *Allocation Mechanism*

The integration between the load balancer and the DCC follows a rigorous state-management protocol to ensure real-time accuracy in task distribution. Upon the selection of a VM through the hierarchical logic, the DCC receives both the unique VM identifier and the specific task metadata. If the selected VM is currently in an 'Available' state, the system immediately updates its status to 'Busy' and increments its active task counter (L_i) by one. This synchronization is crucial for preventing the double-booking of resources, as it ensures that the state table reflects the true workload of the hardware before the next scheduling decision is made. By maintaining this precise count, the algorithm can accurately assess whether a VM remains within its assigned load thresholds (L_1 or L_2).

Once the task execution is finalized, the system initiates a cleanup and metric-recording phase to maintain the feedback loop. The VM's running task count is decremented, and if the count reaches zero, the VM's status is reverted to 'Available' in the central registry, signaling its readiness for Phase 1 selection in future cycles. Simultaneously, the total processing time for the completed task is recorded and utilized to calculate the VM's historical throughput efficiency. This continuous update of performance data is what drives the Efficiency-Aware logic in Phase 3, allowing the load balancer to adapt to the actual performance characteristics of heterogeneous hardware rather than relying on static specifications alone.

6. *Summary of Proposed Method*

The proposed hybrid load-balancing method integrates the deterministic resource management of the TLB algorithm with the distributive efficiency of the ESCE approach. To address the limitations of these parent algorithms, such as lack of capacity awareness and potential node saturation, this method implements a sophisticated three-stage dispatching strategy. Initially, the logic prioritizes idle VMs to ensure that high-performance hardware does not remain underutilized. As the system load increases, the algorithm transitions to a secondary phase, assigning tasks to partially loaded VMs using a tunable capacity-weighted score. This score uniquely balances CPU and memory priorities through a weighting factor (w), allowing the system to adapt to diverse task requirements while maintaining optimal resource alignment.

To ensure system stability during peak traffic, the method incorporates a third fallback stage that identifies the most efficient VMs based on historical throughput metrics. This hierarchical selection is governed by a strict two-stage load threshold mechanism (L_1, L_2) that prevents any single VM from becoming a bottleneck through excessive overloading. By enforcing these operational boundaries, the proposed method mitigates the risk of performance degradation common in standard load balancers. Experimental results confirm that this balanced, capacity-aware hybrid approach significantly reduces overall DC processing times and improves user response rates, establishing a more robust solution for heterogeneous cloud environments.

7. *Pseudocode and Flowchart*

The operational workflow is illustrated in Figure 3, which maps the system's adaptive behavior from task arrival through the hierarchical allocation stages. Each decision node serves as a critical checkpoint, enforcing strict load thresholds to ensure that no single resource exceeds its efficient operating limit while transitioning tasks between idle, partially loaded, and high-efficiency VM pools. Complementing this, the pseudocode in Figure 4 provides a formal technical specification of these routines, detailing the logic for capacity-weighted scoring and the fallback mechanism for task queuing. Together, these visual and technical guides offer a transparent and reproducible framework for implementing the proposed capacity-aware logic in heterogeneous cloud environments.

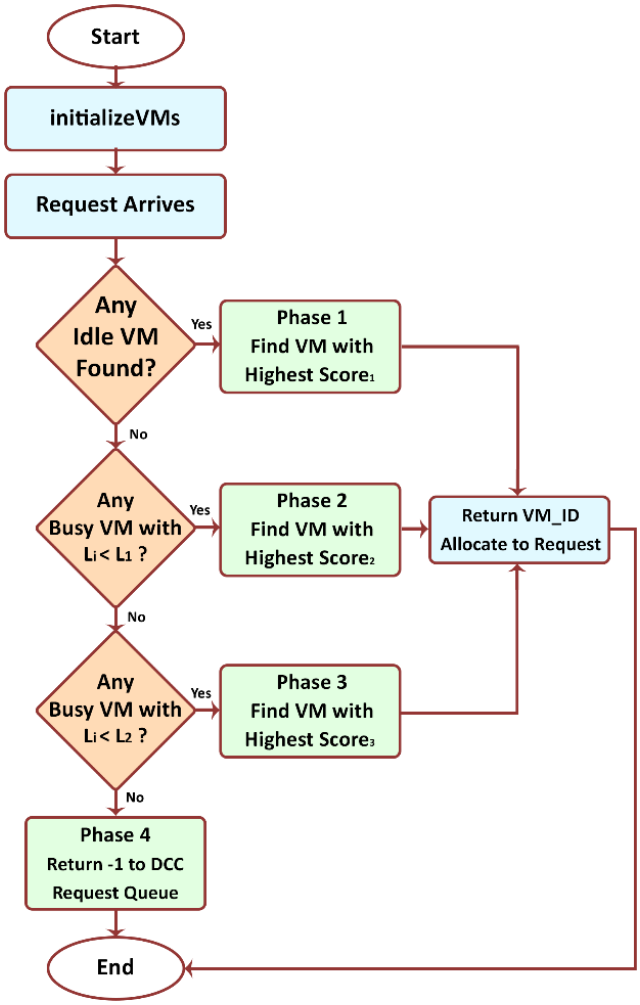

Figure 3. Workflow of proposed method

```
function GET_NEXT_VM():
  best_idle_vm ← null; best_idle_score ← -∞
  best_medium_load_vm ← null;  best_capacity_score ← -∞
  best_high_load_vm ← null; best_throughput_score ← -∞
  for each vm_id in all_VMs do
    if VM_STATE(vm_id) = AVAILABLE then
      // Phase 1: Select most powerful idle VM
      Score1 ← MIPS_Power(vm_id)  // Equation (1)
      if Score1 > best_idle_score then
        best_idle_score ← Score1
        best_idle_vm ← vm_id
    else if Li ≤ L1 then
      // Phase 2: Capacity-based selection
      Score2 ← Processing_Capacity(vm_id)  // Equation (2)
      if Score2 > best_capacity_score then
        best_capacity_score ← Score2
        best_medium_load_vm ← vm_id
    else if Li < L2 then
      // Phase 3: Efficiency-based selection
      Score3 ← Throughput(vm_id) // Equation (3)
      if Score3 > best_throughput_score then
        best_throughput_score ← Score3
        best_high_load_vm ← vm_id
  // Allocation (idle → medium load → high load)
  if best_idle_vm ≠ null then
    return best_idle_vm;
  else if best_medium_load_vm ≠ null then
    return best_medium_load_vm;
  else if best_high_load_vm ≠ null then
    return best_high_load_vm;
  else
    return -1;  // Queue task (Phase 4)
end function
```

Figure 4. Pseudocode of proposed method

V. Experimental Setup and Simulation

Researchers in cloud computing need to perform actual experiments in their studies, setup, implementation, and experiments in the cloud; they often have a massive cost of creating a real cloud environment. Using models and simulation software to replicate cloud environments and perform prerequisite testing is one viable alternative [21]. There are a multitude of open-source tools such as CloudSim, CloudAnalyst, Cloud Reports, and Green Cloud, available for simulating the LB strategies in an IaaS cloud model [22].

1. *CloudAnalyst Simulator*

CloudAnalyst is a graphical user interface-based simulator that provides support for the evaluation of social network tools based on the geographic distribution of users and DCs. It was designed with the extended capabilities of CloudSim. CloudAnalyst offers a sophisticated simulation framework through Map Interface for deploying real-time DCs and monitoring LB, cloud cluster monitoring, and DC data flow in real-time [23].

The architectural design of CloudAnalyst is demonstrated in Figure 5 which expands over CloudSim toolbox [24].

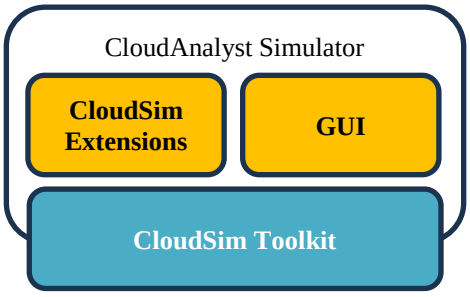

Figure 5. Architecture of CloudAnalyst (adopted from [24]).

The fundamental classes and their obligations are as per the following [24]:

- GUI Main: This is a graphical user interface which acts as front-end controller.
- User Base: It is nothing but a collection of customers/users at a single place.
- Cloud Data Centre Controller: This object controls the DC.

- Internet Characteristics: Different internet characteristics include execution time between Cloud server and client.
- Service Broker and its executions: It defines the different administration representatives.
- VMLoadBalancer Object: It balances the load by modifying procedures utilized by DC to dole out VM demand.
- DC Base Element: This component tells about DCs and machines for the UI.

2. Simulation Configuration

Configuring simulation tests in CloudAnalyst requires defining five main elements: the cloud DCs and their hardware specifications, the UBs generating the workload, and the policies governing resource allocation, the Service Broker Policy (SBP) and Load Balancing Policies (LBPs). To thoroughly evaluate the proposed method under varying conditions, three heterogeneous simulation scenarios were designed. The simulation models a 24-hour operational period to capture both peak and off-peak demand. To minimize latency, the SBP is configured to route user traffic to the geographically closest DC. Table 2 details the UB configurations for the experimental scenarios, including geographic distribution and load characteristics. The simulation employs multiple, globally distributed DCs, as outlined in Table 3. Each DC hosts two heterogeneous servers with processing capacities of 20,000 and 40,000 MIPS, respectively (Table 4). CloudAnalyst simulation tool, which is built on the CloudSim simulator, is an open-source, event-driven framework implemented in Java [22]. Heterogeneous VMs were defined in the simulation configuration with categories and resource allocations specified in Table 5. VM instances are instantiated in a RR manner: 'Weak', 'Moderate' and 'Strong' based on the user-defined total number of the VMs in GUI.

Thresholds L_1=30 and L_2=60 were derived from pilot studies: L_1 prevents premature queueing while preserving ~30% idle capacity for bursts, and L_2 aligns with manufacturer-recommended concurrency limits for "Strong" VMs. The weighting factor w=0.75 reflects the CPU-bound workload (50–300 byte tasks), prioritizing MIPS over RAM. I acknowledge these values are heuristic and workload-dependent; formal sensitivity analysis is planned as future work. Production deployments should tune these parameters based on their specific VM capacity distributions and workload profiles. The following parameters are configurable in the advanced settings:

- Simulation Duration: 24 hours
- User grouping factor in UBs: 100
- Request grouping factor in DCs: 100
- LBP across VMs in a DC: the Proposed Method, and TLB, RR, and ESCE.
- The executable instruction length: RR manner 50-300 bytes in tests as shown in Table 6.

Table 2. UB configurations

UB	Region	Request Per User Per hour	Data Size Per Request (byte)	Peak Hour Start	Peak Hour End	Average Peak Users	Average Off-Peak Users	Scenario
1	R3	60	100	1	3	120,000	12,000	1
1	R3	70	100	01	03	200,000	20,000	
2	R2	70	100	20	22	200,000	20,000	
3	R4	70	100	21	23	100,000	10,000	
1	R0	75	100	13	15	300,000	30,000	
2	R1	75	100	15	17	200,000	20,000	
3	R2	75	100	20	22	300,000	30,000	
4	R3	75	100	01	03	300,000	30,000	
5	R4	75	100	21	23	150,000	15,000	
6	R5	75	100	09	11	150,000	15,000	

Table 3. DC configurations

Name	Region	Arch	OS	VMM	No. VMs	Scenario
DC1	R3	x86	Linux	Xen	5	1
DC1	R3	x86	Linux	Xen	11	
DC2	R2	x86	Linux	Xen	12	
DC3	R4	x86	Linux	Xen	9	

DC1	R0	x86	Linux	Xen	21	3
DC2	R1	x86	Linux	Xen	15	
DC3	R2	x86	Linux	Xen	20	
DC4	R3	x86	Linux	Xen	16	
DC5	R5	x86	Linux	Xen	11	
DC6	R4	x86	Linux	Xen	12	

Table 4. Physical hardware for each DC

Memory (Mb)	Storage (Mb)	Available Bandwidth	Number of Processors	Processors Speed	VM Policy
204,800	100,000,000	1Gb/s	4 CPU	20,000 MIPS	Time-Shared
204,800	100,000,000	1Gb/s	4 CPU	40,000 MIPS	Time-Shared

Table 5. Configuration of heterogeneous types

Name	Ram	Image Size	V Cores Count	Bandwidth	MIPS
Weak	1,024	10,000	1	1,000	1,000
Moderate	2,048	10,000	2	1,000	2,000
Strong	4,096	10,000	4	1,000	4,000

Table 6. Executable Instruction Lengths in three scenarios.

	Scenario1			Scenario 2			Scenario 3		
Length (byte)	50	100	150	75	150	225	100	200	300

3. Performance Metrics

Algorithm efficiency, or performance, is evaluated using several key metrics. The most common of these include processing time, response time, and load distribution [25, 26].

Overall Processing Time

Process Time (PT) is the interval a DC takes to handle a request after receiving it from the load balancer. Widely regarded as one of the most reliable performance metrics in cloud computing, it directly measures processing efficiency. The average PT is the time required by the DC to process all the received tasks from the users. Processing time (PT) is calculated using Equation (4) [26]:

$$PT = L_{VM} / C_{VM} \quad (4)$$

In this formulation, L_{VM} represents the aggregate computational load currently distributed across the entire DC, calculated by summing the individual task loads assigned to every active VM. Correspondingly, C_{VM} denotes the total available processing capacity of the environment, derived from the combined hardware potential of all VMs within the cluster. To establish a precise baseline for these metrics in a heterogeneous setting, Equation (5) is employed to determine the specific capacity of each individual VM by integrating its hardware specifications, such as MIPS [26].

This mathematical approach ensures that the load balancer can accurately assess global resource utilization and normalize performance data across various hardware tiers, allowing for a more balanced and intelligent distribution of incoming requests.

$$C_{VM} = p * q \quad (5)$$

Where C_{VM} is the capacity for only one VM, p is the processing speed of the processor (CPU) in MIPS and q is number of busy CPU.

Overall Response Time

Response Time (RT) - Mathematically, it can be stated as the delay between the arrival time and the time when the process initially obtains the CPU. Theoretically, it is the time needed to give response to the user's request. The minimum is the RT, the efficient is the LB algorithm [25, 26].

$$RT = T_{Finish} - T_{Arrival} + T_{Delay} \quad (6)$$

Equation (6) shows RT metric, this metric is derived using the following parameters: request arrival time ($T_{Arrival}$), request completion finish time (T_{Finish}), and transmission delay (T_{Delay}) [26].

Load Distribution

Load distribution, calculated using Equation (7), Load distribution quantifies the percentage of the total workload allocated to each VM [26].

$$Load\ Distribution = \frac{No.\ tasks\ on\ VM_n}{No.\ tasks\ on\ all\ VMs} \times 100 \qquad (7)$$

The Load Distribution metric, formulated in Equation (7), serves as a critical performance indicator by quantifying the specific percentage of the total workload assigned to each individual VM [26]. By calculating the ratio of a specific VM's active load against the cumulative load of the entire cluster, the algorithm provides a normalized view of resource utilization across the heterogeneous environment. This measurement is essential for identifying potential hotspots where a single node might be disproportionately burdened, allowing the load balancer to verify that tasks are being distributed in a manner consistent with the capacity-aware logic established in the preceding phases.

4. *Results*

This section presents a comprehensive evaluation of the four scheduling methods, RR, ESCE, Basic TLB, and the proposed approach, by examining their average response time, DC processing time, and load distribution behavior under a range of workload conditions. I quantify performance differences using statistical summaries and organize the experimental findings into clear tables and figures for ease of comparison. Processing times for the three test scenarios are reported in Table 7 and visualized in Figure 7, while response-time measurements appear in Table 8 and Figure 8. Load distribution outcomes across the heterogeneous VMs are summarized in Table 9. Additionally, Figure 6 highlights the response time result of the proposed method in the third scenario, emphasizing its behavior under the most demanding conditions.

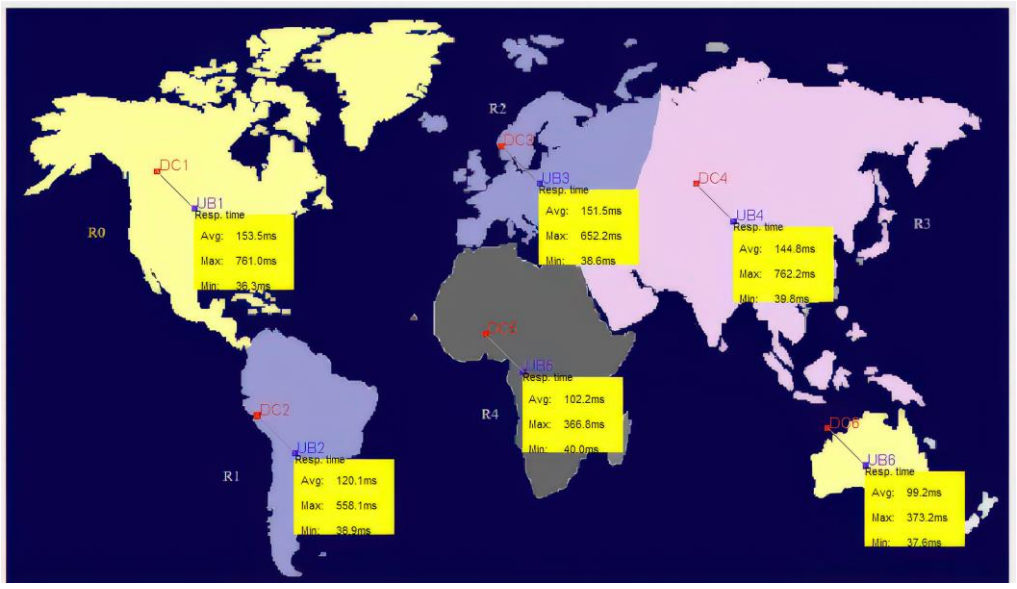

Figure 6. response time result of the proposed method in the third experimental scenario

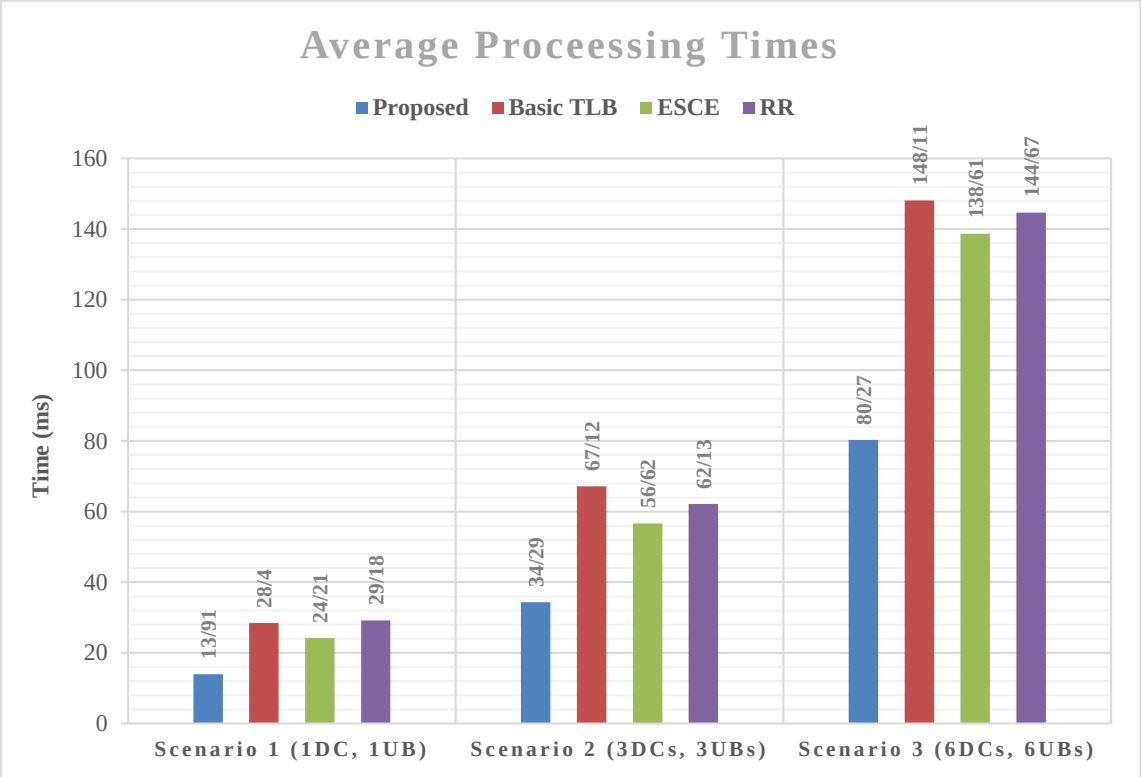

Figure 7. Average processing time in experimental scenario

Table 7. Comparison of average, minimum, and maximum processing times

	Average (ms)	Min (ms)	Max (ms)	Scenario
RR	29.18	0.07	211.95	
ESCE	24.21	0.07	195.32	
TLB	28.40	0.07	138.27	
Proposed Method	**13.91**	**0.04**	**87.46**	
RR	62.13	0.42	54506.85	
ESCE	56.62	0.41	57080.56	
TLB	67.12	0.41	396.19	
Proposed Method	**34.29**	**0.41**	**324.04**	
RR	144.67	0.38	45592.45	
ESCE	138.61	0.39	51834.86	
TLB	148.11	0.56	888.43	
Proposed Method	**80.27**	**0.42**	**691.06**	

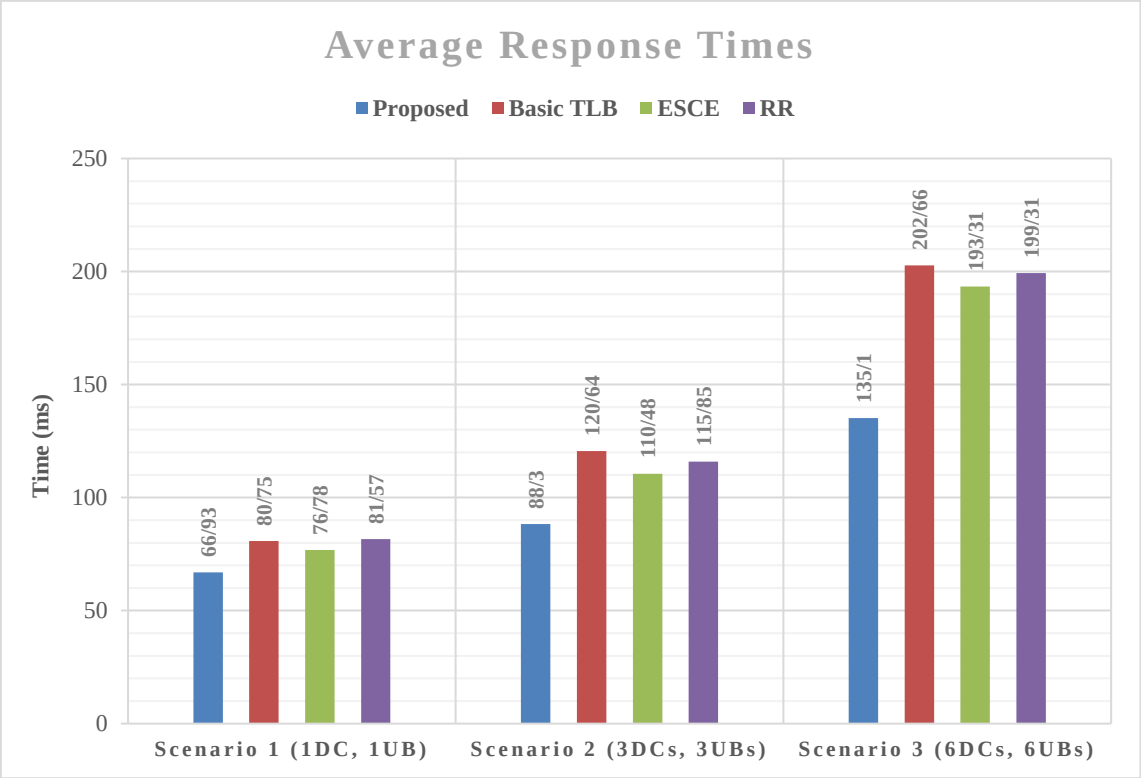

Figure 8. Average response time in experimental scenarios

Table 8. Comparison of average, minimum, and maximum response times

	Average (ms)	Min (ms)	Max (ms)	Scenario
RR	81.57	37.38	273.90	
ESCE	76.78	37.26	254.11	
TLB	80.75	37.26	205.53	
Proposed Method	**66.93**	**37.26**	**148.50**	
RR	115.85	36.78	54562.16	
ESCE	110.48	36.95	57130.70	
TLB	120.64	36.95	461.19	
Proposed Method	**88.30**	**36.94**	**384.84**	
RR	199.31	38.13	45644.80	
ESCE	193.31	36.35	51885.76	
TLB	202.66	36.35	962.43	
Proposed Method	**135.10**	**36.35**	**762.19**	

Table 9. Ratio of Requests Sent to Different VM Types

	Scenario 1			Scenario 2			Scenario 3		
	Strong	Moderate	Weak	Strong	Moderate	Weak	Strong	Moderate	Weak
Proposed	**51.82%**	**24.80%**	**23.38%**	**56.91%**	**25.05%**	**18.04%**	**58.77%**	**23.77%**	**17.46%**

TLB	27.01%	27.58%	45.41%	33.70%	29.51%	36.79%	34.82%	28.34%	36.84%
ESCE	45.38%	24.91%	29.71%	53.42%	20.50%	26.08%	55.70%	18.41%	25.89%
RR	19.99%	39.99%	40.02%	30.88%	34.55%	34.57%	31.53%	33.59%	34.88%

VI. Analysis and Discussion

The simulation results comprehensively demonstrate the superior performance of the proposed hybrid LB method across all evaluation scenarios. Quantitative analysis reveals that the proposed method achieves substantial improvements in both response time and processing time compared to conventional algorithms. Specifically, when compared to the Basic TLB algorithm, the proposed method reduces the average response time by approximately 25.75% and the average processing time by approximately 48.58% across all scenarios. Similarly, against the ESCE algorithm, it achieves reductions of approximately 21.01% in average response time and 41.36% in average processing time. Overall, the proposed method reduces average processing time by 44.97% and average response time by 23.38% compared to traditional algorithms. These improvements highlight the efficacy of the hybrid approach in optimizing system performance.

The load distribution patterns in Table 8 provide mechanistic insights into these performance gains. The proposed method intelligently leverages system heterogeneity by directing 51.82–58.77% of requests to strong VMs, while reducing the burden on weaker ones. This contrasts sharply with TLB's near-equal distribution and RR's uniform allocation, confirming that the proposed method successfully matches workload requirements with appropriate VM capacity. This strategic distribution ensures optimal resource utilization and minimizes bottlenecks.

The convergence of these results validates the hybrid approach's ability to address fundamental limitations of both TLB (through capacity-aware scoring) and ESCE (through load threshold protection). The demonstrated improvements in performance metrics, combined with intelligent resource utilization, establish the proposed method as a robust and efficient solution for cloud LB, delivering enhanced performance without incurring additional operational costs.

VII. Conclusions and Future Work

This paper presented a novel hybrid LB method designed to address the well-documented limitations of the TLB and ESCE algorithms in cloud computing environments. By integrating a two-stage load threshold mechanism with a multi-phase VM selection process, the proposed method effectively balances the load while explicitly accounting for the heterogeneous capacity of VMs. The core achievement of this approach is its ability to prevent VM overloads through a strict load ceiling while intelligently distributing tasks based on processing power, current load, and historical efficiency. Simulation results obtained from the CloudAnalyst simulator provide compelling evidence of the method's superiority, demonstrating a significant 44.97% improvement in DC processing time and a 23.38% reduction in client response time on average against traditional ESCE and TLB algorithms, all achieved without incurring additional operational costs. These results confirm that the method successfully enhances overall system performance, resource utilization, and QoS.

While this study establishes a strong foundation for capacity-aware LB, it opens several promising avenues for future research. First, a comprehensive cost-benefit analysis is warranted. Investigating the trade-offs between employing higher-cost, high-performance VMs and the overall reduction in DC processing time could reveal that the improved efficiency leads to a net reduction in operational costs, making a compelling business case for adoption. Second, the decision model can be extended to incorporate a wider range of VM properties. Factors such as available bandwidth and storage I/O capacity are critical for data-intensive applications. Integrating these metrics into the VM selection score could further optimize task allocation, prevent I/O-based bottlenecks, and improve efficiency for a broader spectrum of cloud workloads, from high-performance computing to big data analytics.

References:

[1] *P. Kumar and R. Kumar, "Issues and challenges of load balancing techniques in cloud computing: A survey," ACM Comput. Surv., vol. 51, p. 120, 2019, doi:10.1145/3281010.*

[2] *E. J. Ghomi, A. M. Rahmani, and N. N. Qader, "Load-balancing algorithms in cloud computing: A survey," J. Netw. Comput. Appl., vol. 88, pp. 50–71, 2017, doi:10.1016/j.jnca.2017.04.007.*

[3] *D. A. Shafiq, N. Z. Jhanjhi, and A. Abdullah, "Load balancing techniques in cloud computing environment: A review," J. King Saud Univ. — Comput. Inf. Sci., vol. 34, pp. 3910–3933, 2022, doi:10.1016/j.jksuci.2021.02.007.*

[4] *N. Devi, S. Dalal, K. Solanki et al., "A systematic literature review for load balancing and task scheduling techniques in cloud computing," Artif. Intell. Rev., vol. 57, p. 276, 2024, doi:10.1007/s10462-024-10925-w.*

[5] *S. M. Shetty and S. Shetty, "Analysis of load balancing in cloud data centers," J. Ambient Intell. Humaniz. Comput., vol. 15, pp. 973–981, 2024, doi:10.1007/s12652-018-1106-7.*

[6] *H. Shoja, H. Nahid, and R. Azizi, "A comparative survey on load balancing algorithms in cloud computing," in Proc. 5th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT), Hefei, China, Jul. 2014, pp. 1–5, doi:10.1109/ICCCNT.2014.6963138.*

[7] *J. Laha, S. Pattnaik, and K. Chaudhury, "Dynamic load balancing in cloud computing: A review and a novel approach," EAI Endorsed Trans. Internet Things, vol. 10, 2024, doi:10.4108/eetiot.5387.*

[8] *H. Le and H. Tran, "ITA: The improved throttled algorithm of load balancing on cloud computing," Int. J. Comput. Netw. Commun., vol. 14, pp. 25–39, 2022, doi:10.5121/ijcnc.2022.14102.*

[9] *M. A. Shahid, M. M. Alam, and M. M. Su'ud, "Performance evaluation of load-balancing algorithms with different service broker policies for cloud computing," Appl. Sci., vol. 13, no. 3, p. 1586, 2023, doi:10.3390/app13031586.*

[10] *S. Agrawal and S. Nigam, "Performance testing of a hybrid algorithm based on throttled and ESCE load balancing algorithm," Int. J. Res. Sci. Eng., vol. 2, no. 6, pp. 135–140, 2021.*

[11] *D. Mekonnen, A. Megersa, R. K. Sharma, and D. P. Sharma, "Designing a component-based throttled load balancing algorithm for cloud data centers," Math. Probl. Eng., Art. no. 4640443, 2022, doi:10.1155/2022/4640443.*

[12] *Y. Rashid and C. I. Nakpih, "ReT-ELBa: A novel algorithm for efficient load balancing in cloud computing," Asian J. Res. Comput. Sci., vol. 17, pp. 56–64, 2024, doi:10.9734/ajrcos/2024/v17i5438.*

[13] *T. Matiwure and A. Ndlovu, "Enhancing throttled load balancing algorithm with machine learning for dynamic resource allocation in cloud computing environments," Int. J. Comput. Sci. Mobile Comput., vol. 14, pp. 20–25, 2025, doi:10.47760/ijcsmc.2025.v14i06.003.*

[14] *S. S. Priya and T. Rajendran, "Enhanced weighted round robin: A new paradigm in cloud load balancing," Indian J. Sci. Technol., vol. 18, no. 15, pp. 1220–1228, 2025, doi:10.17485/IJST/v18i15.3976.*

[15] *S. Patel, R. Patel, H. Patel, and S. Vahora, "CloudAnalyst: A survey of load balancing policies," Int. J. Comput. Appl., vol. 117, no. 21, 2015, doi:10.5120/20679-3525.*

[16] *V. D. Kumar, J. Praveenchandar, M. Arif, A. Brezulianu, O. Geman, and A. Ikram, "Efficient cloud resource scheduling with an optimized throttled load balancing approach," Comput. Mater. Continua, vol. 77, pp. 2179–2188, 2023, doi:10.32604/cmc.2023.034764.*

[17] *N. Chauhan, D. G. Thakur, D. A. Joshi, V. Kumar, and A. Kumar, "Review of techniques and algorithms of load balancing in cloud computing," J. Recent Innov. Comput. Sci. Technol., vol. 1, no. 1, pp. 15–26, 2024, doi:10.70454/JRICST.2024.10103.*

[18] *A. H. Zamri, N. S. M. Pakhrudin, S. Saaidin, and M. Kassim, "Equally spread current execution load modelling with optimize response time brokerage policy for cloud computing," Int. J. Adv. Comput. Sci. Appl. (IJACSA), vol. 14, no. 2, 2023, doi:10.14569/IJACSA.2023.0140257.*

[19] *A. Sharma and K. K. Sharma, "A novel approach for load balancing distribution and storage by using cloud computing," in E3S Web Conf., vol. 399, Art. no. 04009, 2023, doi:10.1051/e3sconf/202339904009.*

[20] *D. G. Akinola, E. Adetiba, A. Abayomi, U. Nnaji, S. Thakur, and S. Moyo, "Intra-datacenter load balancing in a federated cloud with throttled algorithm," in 2024 Int. Conf. Sci., Eng. Bus. for Driving Sustain. Dev. Goals (SEB4SDG), Omu-Aran, Nigeria, 2024, pp. 1–9, doi:10.1109/SEB4SDG60871.2024.10629693.*

[21] *M. A. Shahid, M. M. Alam, and M. M. Su'ud, "A systematic parameter analysis of cloud simulation tools in cloud computing environments," Appl. Sci., vol. 13, p. 8785, 2023, doi:10.3390/app13158785.*

[22] *S. Shanmugapriya and N. Priya, "Examination of cloud simulation platforms and implementation of load balancing in CloudAnalyst," Indian J. Sci. Technol., vol. 17, no. 33, pp. 3424–3436, 2024, doi:10.17485/IJST/v17i33.1751.*

[23] *O. Oladimeji, D. Oyeyiola, O. Oladimeji, and P. Oyeyiola, "A comprehensive survey on cloud computing simulators," Sci. J. Informatics, vol. 8, pp. 51–59, 2021, doi:10.15294/sji.v8i1.28878.*

[24] *S. R. Jena, R. Shanmugam, K. Saini, and S. Kumar, "Cloud computing tools: Inside views and analysis," Procedia Comput. Sci., vol. 173, pp. 382–391, 2020, doi:10.1016/j.procs.2020.06.045.*

[25] *Y. Lohumi, D. Gangodkar, P. Srivastava, M. Khan, A. Alahmadia, and A. Alahmadia, "Load balancing in cloud environment: A state-of-the-art review," IEEE Access, 2023, doi:10.1109/ACCESS.2023.3337146.*

[26] *N. Elnagar, G. El-Kabbany, A. Al-Awamry, and M. B. Abdelhalim, "Simulation and performance assessment of a modified throttled load balancing algorithm in cloud computing environment," Int. J. Electr. Comput. Eng., vol. 12, no. 2, pp. 2088–2096, 2022, doi:10.11591/ijece.v12i2.pp2087-2096.*

Author Biography:

Ali Yousefi Choubini is a part-time lecturer in the Department of Computer Engineering at Sepahan Institute of Higher Education Science and Technology, Isfahan, Iran. He received his **B.Sc.** in Computer Engineering from the Sepahan Institute of Higher Education (2018-2022), graduating as the top-ranked student. He is currently pursuing his **M.Sc.** in Computer Engineering at Shahid Ashrafi Esfahani University (2023-2025), where he maintains a first-ranked position in his cohort with a remarkable **GPA of 19.49/20**. His research interests lie in cloud computing and load balancing, and **he aims to transition to a Ph.D. program following his Master's studies**