The Journal of Supercomputing (2025) 81:469
https://doi.org/10.1007/s11227-025-06922-7

Decentralized dynamic load balancing for virtual machines in cloud computing: a blockchain-enabled system with state channel optimization

J. Roselin[1] · **Israelin J. Insulata**[1]

Accepted: 7 January 2025
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2025

Abstract
This paper introduces an innovative load balancing algorithm that utilizes blockchain-enabled cloud computing environments. The proposed scheme leverages blockchain technology's decentralized architecture to dynamically and efficiently distribute workloads across virtual machines (VMs). This approach optimizes resource utilization and enhances the performance of cloud services. By integrating smart contracts and employing a meticulous VM selection process, our method effectively addresses the challenges associated with traditional load balancing techniques, which often struggle to adapt to dynamic, heterogeneous workloads. Furthermore, our algorithm promotes transparency and security in task allocation and execution, capitalizing on blockchain's inherent features of immutability and consensus. The effectiveness of the proposed scheme is demonstrated through rigorous simulation using the CloudSim toolkit, showcasing significant improvements over existing methods in terms of makespan, execution time, resource utilization, and throughput. These results underline the potential of our proposed solution to revolutionize cloud computing infrastructure management, making it more adaptable, efficient, and resilient to varying computing demands.

Keywords Load balancing · Blockchain · Cloud computing · Virtual machines · State channels

1 Introduction

In recent years, cloud computing has emerged as a pivotal infrastructure for delivering a wide array of services, ranging from simple storage solutions to complex computational resources [1–3]. At the heart of cloud computing lies the data center,

✉ J. Roselin
roselin.js@auttvl.ac.in

[1] Anna University Regional Campus, Tirunelveli, India

Published online: 07 February 2025

a networked collection of servers that store, process, and manage vast amounts of data. As the demand for cloud services continues to grow, efficiently managing these data centers to ensure high availability, reliability, and performance has become a critical challenge [4–6]. One of the primary issues confronting data centers in cloud computing environments is load balancing [7–9]. Effective load balancing ensures that the workload is evenly distributed across servers or virtual machines (VMs), preventing any single resource from becoming a bottleneck [10–12]. This distribution is crucial for optimizing resource utilization, minimizing response times, and enhancing the overall user experience [13–15]. However, traditional load balancing algorithms often struggle to cope with the dynamic and heterogeneous nature of cloud workloads. These struggles lead to issues such as increased makespan and execution times, resource over-utilization or under-utilization, limited scalability and adaptability, and overall resource wastage [16–18]. A notable example of these challenges occurred on 27 June 2021, when Google Cloud Load Balancing experienced a significant disruption due to increased TLS handshake errors and connection terminations for HTTPS traffic in Western Europe. The incident, which lasted over two hours, was traced back to a configuration change that inadvertently overloaded an internal component critical for traffic handling [19]. This event highlighted the difficulties that current load balancing methods face in complex and changing cloud environments. To address these challenges, the proposed scheme introduces a novel load balancing algorithm that utilizes a blockchain network. This scheme leverages the inherent advantages of blockchain technology, such as its decentralized architecture, immutable ledger, and smart contract capabilities, to foster a more dynamic, efficient, and secure load balancing mechanism. Unlike centralized and static algorithms that struggle with the diverse demands of cloud environments, our algorithm dynamically adjusts to changes in workload patterns and resource availability in blockchain-enabled cloud computing environments. By integrating the principles of blockchain with advanced load balancing strategies, this approach aims to optimize resource utilization across the network. This integration not only improves the overall scalability, reliability, and performance of cloud computing applications but also enhances transparency and security in task allocation and execution. This innovative approach positions the proposed scheme as a robust solution for managing the complex and dynamic nature of modern cloud computing environments, ultimately leading to more adaptable, efficient, and resilient cloud services.

2 Related work

This section offers an in-depth examination of previous research efforts, focusing on significant contributions and uncovering gaps that motivate our current study. Despite considerable efforts to improve load balancing, existing solutions still face significant challenges. Zhao et al. [20] have proposed a Heuristic Clustering-based load balancing scheme. This scheme combines a heuristic clustering-based task deployment technique with Bayes Theorem for load balancing. This scheme might suffer from increased makespan due to its emphasis on long-term load distribution over immediate task execution efficiency. Inorder to avoid this issue, Duan et al. [21]

have proposed a virtualization framework leveraging distributed virtual switches and OpenFlow for load balancing in data centers. It aims to manage network communication efficiently but might face scalability and adaptability challenges. To overcome the challenges of scalability and adaptability, Kumar et al. [22] have proposed a dynamic load balancing algorithm in which the tasks are sorted using bubble sort, then allocated to Virtual Machines in a First-Come-First-Serve order. This scheme exhibits a longer task transfer times during reallocation, potentially lowering efficiency in environments with frequent task shifts or substantial data movements. To avoid this issue, Tang et al. [23] have proposed a dynamical load-balanced scheduling (DLBS) approach for big data centers in clouds using OpenFlow. The Static route initialization in this scheme limits adaptability to changing network conditions, and the DLBS approach lacks consideration for crucial Quality of Service (QoS) parameters like deadlines and priority, restricting its suitability in scenarios where meeting specific deadlines or prioritizing tasks is crucial. Inorder to address these issues, Gamal et al. [24] have proposed a bio-inspired load balancing scheme. This scheme combines ant colony optimization (ACO) and artificial bee colony (ABC) principles for load balancing. The sensitivity of this algorithm to parameter selection, particularly the evaporation rate and importance weights, may impact its performance, potentially leading to suboptimal load balancing. To eliminate this sensitivity issue, Junaid et al. [25] have proposed hybrid model for load balancing in cloud which combines support vector machine (SVM) and ant colony optimization (ACO) metaheuristic. In this scheme, inaccurate pheromone update strategies could result in suboptimal task assignments, either underutilizing resources by being overly cautious or overloading them by not accounting for current loads. Inorder to overcome this issue, Hung et al. [26] have proposed a load balancing scheme which utilizes gene expression programming (GEP) and genetic algorithm. This scheme could suffer from increased execution times due to the utilization of complex genetic algorithms and the overhead associated with VM migrations. To avoid this drawback, Sohani et al. [27] have proposed a scheme called Predictive Priority-based Modified Heterogeneous Earliest Finish Time (PMHEFT). This scheme utilizes a predictive model for workload demand rate and the emergency of cloud requests. The algorithm involves steps such as calculating load demand rates, adding load requests to the prediction priority queue, determining loading thresholds, and selecting load requests based on priority. This algorithm's performance is sensitive to threshold changes, potentially limiting its adaptability to rapid load variations and diverse conditions. To address the adaptability issue, Kruekaew et al. [28] have proposed a load balancing scheme which combines the artificial bee colony algorithm (ABC) with Q-learning. This scheme faces potential drawbacks in resource utilization due to its complex and computationally intensive nature, dependence on accurate predictive modeling, and the challenges associated with balancing multiple optimization objectives in real-time. Inorder to overcome this problem, Reshan et al. [29] have proposed a load balancing scheme which combines grey wolf optimization (GWO) and particle swarm optimization (PSO). The effectiveness of this scheme in large-scale cloud environments, with numerous resources and complex workloads is limited. As the system scales, the algorithm's ability to efficiently balance loads and allocate resources might decrease. These existing load balancing schemes fail

to address real-time adaptability, comprehensive QoS management, and efficient resource utilization. Moreover, they fail to incorporate predictive analytics and suffer from scalability challenges. Furthermore, high computational overhead and inadequate security mechanisms compromise their practicality and transparency, particularly in multi-tenant cloud systems. The proposed scheme effectively bridges these critical gaps in existing methods by integrating dynamic dual-load assessment (HRU and CRU) for real-time adaptability and urgency-based task prioritization to enhance Quality of Service (QoS) management. Moreover, the scheme optimizes resource utilization, effectively mitigating risks of overloading or under-utilization. Furthermore, predictive analytics are leveraged to proactively manage workloads, ensuring efficient task allocation. Additionally, scalability is achieved through the integration of blockchain and state channels, which minimize latency and computational overhead. Real-time task reassignment further reduces delays, while blockchain's immutable ledger enhances security and transparency. Collectively, these features establish the proposed scheme as a robust, scalable, and efficient solution for dynamic cloud environments.

3 Preliminaries

In this section, we present the fundamental concepts and technologies that underpin the proposed scheme, providing the necessary background for understanding the subsequent sections. We introduce the necessary definitions and notations used throughout our proposed scheme. These foundational concepts will aid in understanding the mechanisms and operations involved in our scheme.

Definitions

Definition 1 Blockchain: Blockchain is a decentralized digital ledger technology that records transactions across many computers so that the record cannot be altered retroactively without the alteration of all subsequent blocks and the consensus of the network. This technology ensures transparency, security, and immutability of data, which is particularly valuable in environments requiring high trust and accountability. Blockchain's decentralized nature eliminates the need for a central authority, thereby reducing the risk of single points of failure and enhancing the system's resilience.

Definition 2 Virtual Machine (VM): A virtual machine (VM) is a software emulation of a physical computer that runs an operating system and applications just like a physical computer. VMs provide the ability to run multiple operating systems on a single physical machine, which allows for better resource utilization, flexibility in deploying applications, and isolation between different computing environments. In cloud computing, VMs are essential for scalability and efficient resource management.

Definition 3 Load Balancing: Load balancing is the process of distributing workloads across multiple computing resources, such as servers or VMs, to ensure no

single resource is overwhelmed. Effective load balancing optimizes resource use, maximizes throughput, minimizes response time, and prevents system overload. It is crucial for maintaining high availability and reliability of cloud services by dynamically adjusting to varying workload demands.

Definition 4 State Channels: State channels are techniques that allow participants to make multiple transactions off the blockchain while only submitting two on-chain transactions: one to open the channel and another to close it. This approach significantly reduces the load on the blockchain and allows for faster and more scalable interactions. State channels enable real-time transactions and updates, making them ideal for scenarios requiring frequent and rapid exchanges.

Definition 5 Smart Contracts: Smart contracts are self-executing contracts with the terms of the agreement directly written into code. They automatically enforce and execute the agreed-upon terms, facilitating, verifying, or enforcing the negotiation or performance of a contract. Smart contracts are essential in blockchain applications for automating processes and ensuring trust without the need for intermediaries.

Problems

Problem 1 Increased Makespan and Execution Times: Traditional algorithms often struggle to efficiently manage tasks of varying sizes and complexities, leading to prolonged makespan and execution times. This inefficiency results in delays in task completion and a decline in overall system performance.

Problem 2 Inefficient Resource Utilization: Conventional load balancing methods fail to dynamically adjust to workload patterns, causing either over-utilization or under-utilization of resources. Over-utilization leads to bottlenecks and resource contention, while under-utilization results in wasted resources and increased operational costs.

Problem 3 Reduced Throughput: Ineffective load balancing reduces the number of tasks processed per second, negatively impacting throughput. This diminishes the system's ability to handle high task volumes efficiently.

Problem 4 Increased Latency: Inadequate task distribution can increase latency, leading to slower response times and delays in task processing, particularly under heavy workloads.

Problem 5 Imbalanced Load Distribution: Traditional load balancing approaches often fail to evenly distribute tasks across resources, resulting in imbalanced workloads. This imbalance can cause performance degradation in overburdened VMs while underutilized VMs remain idle.

Assumptions

1. *Decentralized Network* The system operates in a decentralized environment where no single node has control over the entire network. This ensures robustness and resilience against failures and attacks, as the system's operation does not rely on a central authority.
2. *Resource Availability* The virtual machines (VMs) in the cloud have varying levels of CPU, memory, and storage resources available. This variation is crucial for the dynamic allocation of tasks based on specific resource requirements, allowing for optimal use of available resources.
3. *Task Heterogeneity* Tasks have different resource requirements and deadlines. This heterogeneity must be accounted for to ensure that the load balancing mechanism can effectively prioritize and allocate tasks according to their specific needs, enhancing overall system efficiency and performance.
4. *Blockchain Integration* The blockchain is integrated into the cloud infrastructure to record transactions and ensure transparency and security. This integration leverages blockchain's immutable and decentralized ledger to enhance trust and accountability in task allocation and execution processes.
5. *Smart Contracts* Smart contracts are used to automate the validation and allocation of tasks to VMs based on predefined rules and conditions. These contracts ensure that tasks are processed according to agreed-upon terms, without the need for manual intervention, thereby reducing the risk of errors and improving efficiency.

Notations and Descriptions

The following notations in Table 1 are used throughout the paper to describe various parameters and processes in the proposed system:

4 Proposed problem and its formulation

4.1 Proposed problem

In the rapidly evolving landscape of cloud computing, efficient management of data centers to ensure high availability, reliability, and performance has emerged as a critical challenge. The primary issue at the core of this challenge is load balancing, which is essential for evenly distributing workloads across servers or virtual machines (VMs) to prevent any single resource from becoming a bottleneck. Effective load balancing optimizes resource utilization, minimizes response times, and enhances the overall user experience. However, traditional load balancing algorithms often fall short in addressing the dynamic and heterogeneous nature of cloud workloads, leading to several key problems:

1. *Increased Makespan and Execution Times* Traditional algorithms often struggle to efficiently manage the varying sizes and complexities of tasks, resulting in

Table 1 Notations and descriptions

Notation	Description
VMs	List of available virtual machines
T	Set of tasks
T	The task to be processed
HRU	Historical resource usage
CRU	Current resource usage
DW	Dynamic weight
UF	Urgency factor
TR	Transaction record
ER	Execution result
VMopt	Optimal virtual machine selected for task execution
R	Maximum retries
RS	Retry status
N	Number of time points
δ	Status of validation
L_i	Load on the ith virtual machine
$\overline{L}$	Average load across all virtual machines
LBI	Load Balance Index
$\text{MakeSpan}_{\text{avg}}$	Average Makespan
$\text{Execution Time}_{\text{avg}}$	Average Execution Time
Avg.RU	Average Resource Utilization
α	Weight assigned to Current Resource Usage (CRU)
β	Weight assigned to Historical Resource Usage (HRU)
$\text{Latency}_{\text{avg}}$	Average Latency

longer makespan and execution times. This inefficiency can lead to delays in task completion and reduced overall system performance.

2. *Resource Over-Utilization or Under-Utilization* Conventional load balancing methods may either over-utilize or under-utilize resources due to their inability to dynamically adjust to changing workload patterns. Over-utilization can cause resource contention and bottlenecks, while under-utilization leads to wasted resources and increased operational costs.
3. *Limited Scalability and Adaptability* Existing load balancing solutions may not scale well with increasing workloads and often lack the adaptability required to handle the diverse and fluctuating demands of cloud environments. This limitation hinders their effectiveness in maintaining optimal performance as the scale of operations grows.
4. *Resource Wastage* Inefficient load balancing leads to suboptimal use of resources, resulting in resource wastage. This not only impacts the cost-effectiveness of cloud services but also affects their environmental sustainability.
5. *Security and Transparency Issues* Traditional centralized load balancing approaches are vulnerable to single points of failure and may lack the transpar-

ency required for secure task allocation and execution. These issues can compromise the reliability and trustworthiness of the system.

4.2 Problem formulation

To address the challenges in load balancing within cloud computing environments, we propose a blockchain-enabled decentralized load balancing scheme. The formulation of the problem involves defining the objectives, components, and constraints necessary to develop and evaluate the proposed solution.

Objectives

1. *Minimize Makespan* The total duration required to complete all tasks should be minimized to enhance system efficiency.
2. *Minimize Execution Time* The time taken to execute a set of tasks should be minimized to ensure rapid task processing.
3. *Maximize Resource Utilization* The utilization of available resources should be maximized to avoid under-utilization and over-utilization.
4. *Maximize Throughput* The rate at which tasks are processed within the system should be maximized to handle more tasks in a given time frame.

Problem Components

1. *Tasks (T)* A set of tasks $T = \{\tau_1, \tau_2, \ldots, \tau_n\}$ with varying computational requirements such as CPU, memory and storage.
2. *Virtual Machines (VMs)* A set of virtual machines VMs = {VM1, VM2,…,VMn} with different resource capacities.
3. *Blockchain Network* Utilizes blockchain technology to record and verify all task allocations and executions, ensuring transparency, security, and decentralization.
4. *State Channels* Employed for off-chain computations to enhance scalability and reduce the load on the main blockchain network.

Constraints

1. Task Assignment Each task must be assigned to exactly one VM.

$$\sum_{\text{VM} \in \text{VMs}} x_{\tau,\text{VM}} = 1 \quad \forall \tau \in T$$

where $x_{\tau,\text{VM}}$ is a binary variable indicating if task τ is assigned to one VM.

2. Resource Capacity The total resource demand of tasks assigned to a VM must not exceed the VM's capacity.

$$\sum_{\tau \in T} (x_{\tau,\text{VM}} \times \text{ Resource Demand } (\tau)) \leq \text{Resource Capacity (VM)} \forall \text{VM} \in \text{VMs}$$

3. Blockchain Recording All task allocation and execution details must be securely recorded on the blockchain for transparency and auditability.

The problem formulation integrates these components and constraints into a coherent scheme to achieve the stated objectives. By formulating the problem in this manner, we aim to develop a load balancing scheme that leverages the advantages of blockchain technology to enhance the performance, scalability, and reliability of cloud computing environments.

5 System model

The proposed load balancing system utilizes blockchain technology to improve performance, security, and transparency in cloud computing. The architecture includes interconnected components that efficiently distribute workloads across virtual machines (VMs) which is depicted in Fig. 1.

Blockchain Layer

Blockchain Network Setup and Consensus Mechanism The blockchain network in the proposed system comprises 10 nodes, each representing a Virtual Machine (VM), with synchronized ledgers maintained through the Practical Byzantine Fault Tolerance (PBFT) consensus algorithm. This setup ensures low latency and fault tolerance for up to $\lfloor (n-1)/3 \rfloor$ faulty nodes, making the system resilient to failures, malicious activity, and network partitions. The system actively monitors connectivity through regular heartbeat messages exchanged between nodes. Nodes failing to respond within a predefined timeout are flagged as unavailable, triggering dynamic reassignment of tasks. Using a smart contract, tasks assigned to these isolated nodes are marked as incomplete and reassigned to candidate VMs selected based on their historical resource usage (HRU), current resource usage (CRU), and task urgency. Once the network recovers, state synchronization protocols update the main blockchain ledger with changes from isolated segments, ensuring consistency and preventing data loss. Moreover, the PBFT algorithm maintains system reliability by achieving consensus through a multi-phase process: In the Pre-Preparation Phase, the leader node proposes a block of validated transactions and broadcasts it to all nodes. In Preparation Phase, replica nodes independently validate the block and broadcast their agreement to the network. In Commit Phase, upon receiving agreements from $2f+1$ nodes, the block is finalized, appended to the ledger, and synchronized across the network. This structured approach ensures valid transactions are added to the blockchain while handling Byzantine failures effectively.

State Channels State channels facilitate off-chain interactions, reducing latency and improving throughput by minimizing on-chain transactions. A state channel is initiated between VM and another VM at the onset of a task. This channel is governed by smart contracts, which define the rules for task validation, execution, and status updates. Task-related operations, such as resource validation, urgency calculations, and execution progress monitoring, are conducted off-chain within the state

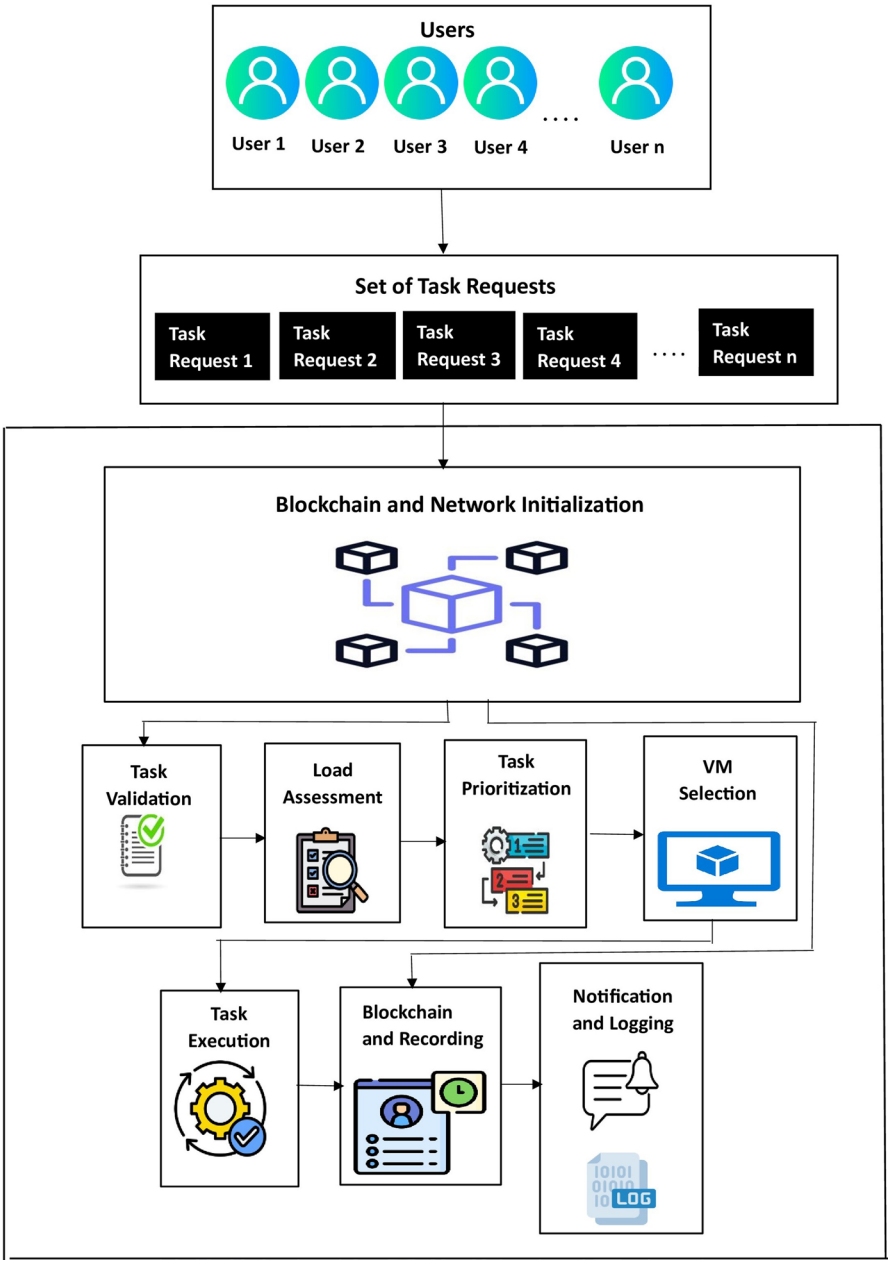

Fig. 1 System model

channel. This ensures rapid response times, free from the delays associated with on-chain consensus mechanisms. Interim updates, such as partial task completion and resource consumption, are recorded within the state channel. Upon task completion,

the state channel is closed. A summary of the task execution, including execution results and resource metrics, is recorded as a final transaction on the blockchain. This step ensures that only essential information is stored on-chain, optimizing both storage and processing overhead.

Task Management

1. *Task Validation* Tasks are validated for completeness and correctness before assignment, ensuring necessary attributes like type, CPU, memory, and storage are checked.
2. *Load Assessment* The system evaluates each VM's load using CPU and memory metrics to calculate a weighted load score, aiding in task distribution.
3. *Task Prioritization and VM Selection* Tasks are prioritized by urgency and resource needs. A scoring system selects the optimal VM, considering load and urgency factors.

Task Execution and Monitoring

1. *Task Execution* Tasks are executed on assigned VMs, with real-time monitoring of resource consumption and execution time.
2. *Blockchain Recording* All task transactions, including assignments, results, and resource usage, are recorded on the blockchain for transparency and accountability.
3. *Notification and Logging* Notifications are sent to stakeholders about task outcomes, and detailed logs are maintained for analysis.

The proposed decentralized system ensures resilience against node failures by dynamically reassigning tasks. State channels enable real-time adaptation to changing workloads and resource availability, ensuring efficient and balanced operations. This system model enhances traditional load balancing through blockchain, offering a robust, efficient, and transparent solution for cloud computing environments.

6 Algorithm of blockchain-enabled decentralized load balancing system

The Algorithm 1 starts by opening a state channel and iterating through each VM to compute its historical resource usage (HRU) and current resource usage (CRU), considering CPU, memory, and I/O cycles. The Dynamic Weight (DW) is then calculated by averaging the HRU and CRU relative to the total CPU, memory, and I/O capacities of each VM. For each task, the algorithm checks if the task's CPU, memory, and I/O requirements exceed the VM's predicted utilizations. If any utilization exceeds 100%, the task is skipped for that VM. Otherwise, the task's urgency factor (UF) is calculated, and the VM is assigned a score based on the UF and DW. After evaluating all tasks and VMs, tasks are sorted by urgency and assigned to the VM with the highest score. The state channel is then closed, and if there are significant updates, the task assignments are recorded on the blockchainnew task assignments.

Algorithm 1 Task Validation and VM Selection

Input: Set of VMs, Set of Tasks (T)
Output: Task Assignments, Updated Blockchain

```
Start
Open State Channel
FOR each VM ∈ VMs DO
Calculate HRU:
VM.HRU ← (Σ (VM.CPU Usage at Time i + VM.Memory Usage at Time i + VM.I/O Usage at Time i)) / (3N)
Calculate CRU:
VM.CRU ← (VM.Current CPU Usage + VM.Current Memory Usage + VM.Current I/O Usage) / 3
Calculate DW:
VM.DW ← (VM.HRU + VM.CRU) / (3 × (VM.Total CPU Capacity + VM.Total Memory Capacity + VM.Total I/O Capacity))
FOR each task τ ∈ T DO
VM.CPU utilization τ ← VM.Current CPU Usage + τ.CpuRequirement
VM.Memory utilization τ ← VM.Current Memory Usage + τ.MemoryRequirement
VM.I/O utilization τ ← VM.Current I/O Usage +
VM.Predicted CPU utilization τ ← VM.CPU utilization τ / VM.TotalCpuCapacity
VM.Predicted Memory Utilization τ ← VM.Memory utilization τ / VM.TotalMemoryCapacity
VM.Predicted I/O Utilization τ ← VM.I/O utilization τ / VM.TotalIOCapacity
IF VM.Predicted CPU utilization τ > 1 OR VM.Predicted Memory Utilization τ > 1 OR VM.Predicted I/O Utilization τ > 1 THEN
VM.Score τ ← 0
CONTINUE // Skip this VM
ELSE
τ.UF ← 1 / (τ.Deadline - CurrentTime)
VM.Score τ ← τ.UF / VM.DW
END IF
END FOR
END FOR
Sort tasks by Urgency Factor (UF)
FOR each task in sorted order
Assign task to VM with the highest score
Remove that assigned VM from the task list
END FOR
Close State Channel
IF significant updates THEN
Update blockchain with new assignments
END IF
End
```

Algorithms 2 and 3 provide crucial mechanisms for managing transactions and task executions in blockchain-based systems. Algorithm 2 updates the blockchain with a new transaction record and notifies relevant parties based on the execution results; it appends the transaction to the blockchain and notifies the submitter of success or logs an error and alerts the admin in case of failure. Algorithm 3 ensures fault tolerance in task allocation using exponential backoff, where the delay for each retry is calculated as $\text{Delay} = 2^i \times \text{Base Delay}$, with i as the attempt number. Tasks are validated and assigned to optimal VMs during each retry. If successful, it returns "Task Assigned"; otherwise, after exhausting retries, it returns "Failed after all retries."

Algorithm 2 Update Blockchain and Notify

Input: TR, τ, ER
Output: Updated Blockchain, Notification and Error Log
1. Update the Blockchain:
2. Blockchain_new ← Blockchain∪{TR}
3. If ER="Success" then
4. Notify τ.Submitter,"Task Completed"
5. Else
6. Log Error("Task Failed", $\tau.ID$)
7. Notify Admin("Task Failure", $\tau.ID$)

Algorithm 3 Handle Retries

Input: τ, VMs
Output: RS
1. Initialize: i←0
2. While τ.Retries > 0:
3. τ.Retries ← τ.Retries – 1
4. Calculate Backoff Time: Delay←$2^i \times$ Base_Delay
5. Wait for Delay
6. (VMopt, δ) ← ValidateTask_SelectVM(τ, VMs)
7. If δ = “Success”:
8. RS ← "Task Assigned"
9. Break
10. Increment i
11. If Retries Exhausted:
12. RS ← "Failed after all retries"
13. Return RS

7 Proposed methodology

To effectively address the challenges of task management in cloud computing environments, we propose a sophisticated system that seamlessly integrates blockchain technology with advanced task prioritization mechanisms. The proposed system aims to enhance system responsiveness and ensure the seamless execution of tasks based on their urgency and resource requirements. Figure 2 shows the Block Diagram of the proposed load balancing Scheme.

Blockchain network initialization

This process involves setting up essential components for efficient operation. It begins with creating a blockchain object, the Transaction Record (TR), responsible for block creation, verification, and storage. Individual nodes, similar to VMs in load balancing, are configured and linked to form a peer-to-peer network, each maintaining a synchronized ledger. State channels are established for off-chain interactions. Smart contracts automate key processes like task management and load balancing.

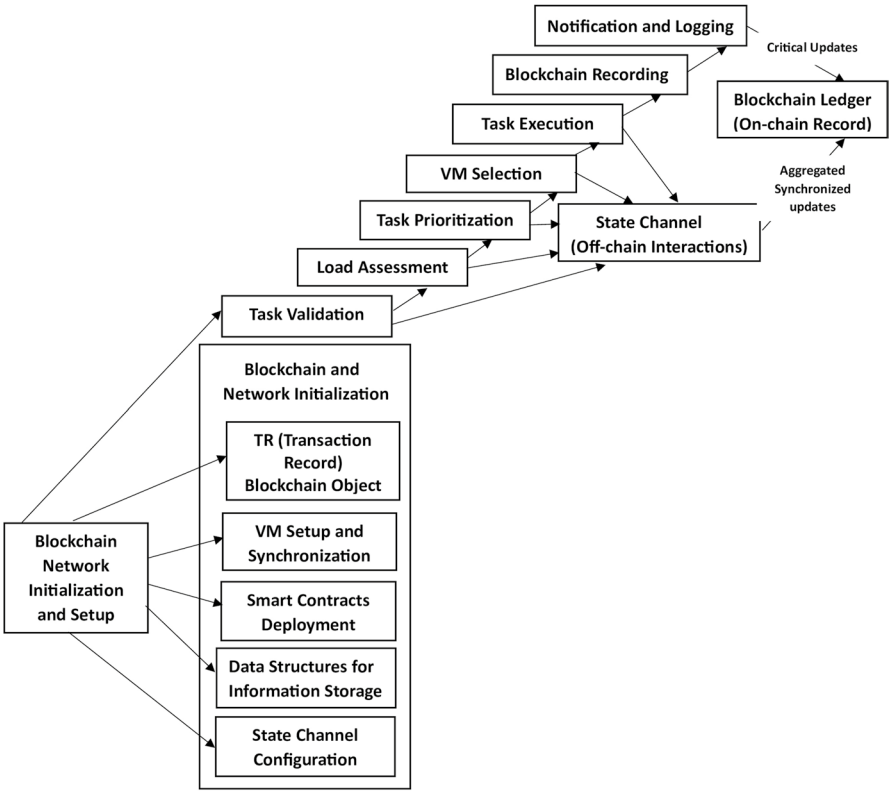

Fig. 2 Block diagram of the proposed load balancing scheme

Load Assessment, Task Prioritization, Predicting Utilization and VM Selection

In cloud services, tasks are typically allocated to virtual machines (VMs) based on their Current Resource Usage (CRU) to achieve load balancing. CRU reflects the current utilization of a VM's resources, such as CPU and memory, ensuring that tasks are assigned to VMs with the most available capacity. Current Resource Usage can be calculated using the formula,

$$\begin{aligned} \text{CRU} = & \frac{\text{Current CPU Usage}}{\text{CPU Capacity}} + \frac{\text{Current Memory Usage}}{\text{Memory Capacity}} \\ & + \frac{\text{Current I/O Usage}}{\text{I/O Capacity}} + \frac{\text{Current Network Bandwidth}}{\text{Network Capacity}} \end{aligned} \tag{1}$$

To ensure efficient task execution, it is important to consider both the Historical Resource Usage (HRU) and Current Resource Usage (CRU) of each VM. Historical Resource Usage (HRU) represents the average load a VM has handled over time, helping to prevent overburdening the consistently utilized VMs. Historical Resource Usage can be calculated using the formula,

$$\text{HRU} = \frac{\sum_{i=0}^{N} (\text{CPU Usage at Time } i + \text{Memory Usage at Time } i + \text{IO Usage at Time } i + \text{Current Network Bandwidth Usage at Time } i)}{4N} \tag{2}$$

where N is the number of time points observed. HRU can be calculated using a window size of $N = 10$ time points, representing the most recent 10 monitoring intervals to provide a balanced representation of recent and long-term resource utilization trends while ensuring computational efficiency.

The proposed scheme considers both Historical Resource Usage (HRU) and Current Resource Usage (CRU) to calculate the Dynamic Weight (DW), ensuring more efficient resource allocation. The system prioritizes the VM with a lower DW value.

The DW is calculated using the following formula:

$$\text{DW} = \frac{\alpha \cdot \text{CRU} + \beta \cdot \text{HRU}}{1} \tag{3}$$

where α and β are constants that represent the weights assigned to CRU and HRU, respectively, and should satisfy $\alpha + \beta = 1$.

However, assigning tasks based on the Dynamic Weight (DW) of virtual machines (VMs) may not always lead to optimal results and can cause certain tasks to overload a VM, leading to inefficiencies in system performance. To achieve effective load balancing, it is crucial to incorporate an analysis of task-related parameters such as resource demand, deadlines of new tasks. By considering both the current state of VMs and the task's demands, the system can ensure more balanced and efficient VM allocation, reducing the risk of overloading and improving overall performance.

To enable more accurate load balancing, the proposed scheme calculates the Predicted CPU and Memory Utilizations of each VM. The CPU and memory requirements are determined as follows:

$$\text{CPU Requirement} = \text{Current CPU Usage} + \text{New Task}'\text{s CPU Requirement} \quad (4)$$

$$\text{Memory Requirement} = \text{Current Memory Usage} + \text{New Task}'\text{s Memory Requirement} \quad (5)$$

$$\begin{aligned} &\text{I/O Requirement} = \text{Current I/O Usage} + \text{New Task}'\text{s I/O Requirement} \\ &\text{Bandwidth Requirement} = \text{Current Bandwidth Usage} + \text{New Task}'\text{s Bandwidth Requirement} \end{aligned} \quad (6)$$

The Predicted CPU and Memory Utilization are then calculated using these formulas:

$$\text{Predicted CPU Utilization} = \frac{\text{CPU Requirement}}{\text{CPU capacity of VM}} \quad (7)$$

$$\text{Predicted Memory Utilization} = \frac{\text{Memory Requirement}}{\text{Memory capacity of VM}} \quad (8)$$

$$\text{Predicted I/O Utilization} = \frac{\text{I/O Requirement}}{\text{I/O capacity of VM}} \quad (9)$$

$$\text{Predicted Bandwidth Utilization} = \frac{\text{Bandwidth Requirement}}{\text{Network capacity of VM}} \quad (10)$$

If either the Predicted CPU Utilization or the Predicted Memory Utilization exceeds 1, the VM is considered overloaded. Conversely, if both utilizations are less than or equal to 1, the task fits well within the VM's capacity, ensuring efficient load distribution.

The proposed scheme computes the Urgency Factor (UF) of each task to prioritize them based on deadlines and time-sensitivity, while assessing the resource requirements of each task. When new tasks arrive, the scheme introduces an Urgency Factor UF(task) to assess their time sensitivity based on the difference between each new task's deadline and the current time. Tasks with a higher urgency are prioritized for completion sooner. The Urgency Factor is calculated using the formula:

$$\text{UF(task)} = \frac{1}{\tau \cdot \text{Deadline} - \text{CurrentTime}} \quad (11)$$

Finally, the system computes a score, which combines Dynamic Weight (DW) and Urgency Factor (UF) to determine the most suitable VM for task execution. Score can be calculated as follows:

$$\text{Score} = \frac{\text{UF(task)}}{\text{DW(VM)}} \quad (12)$$

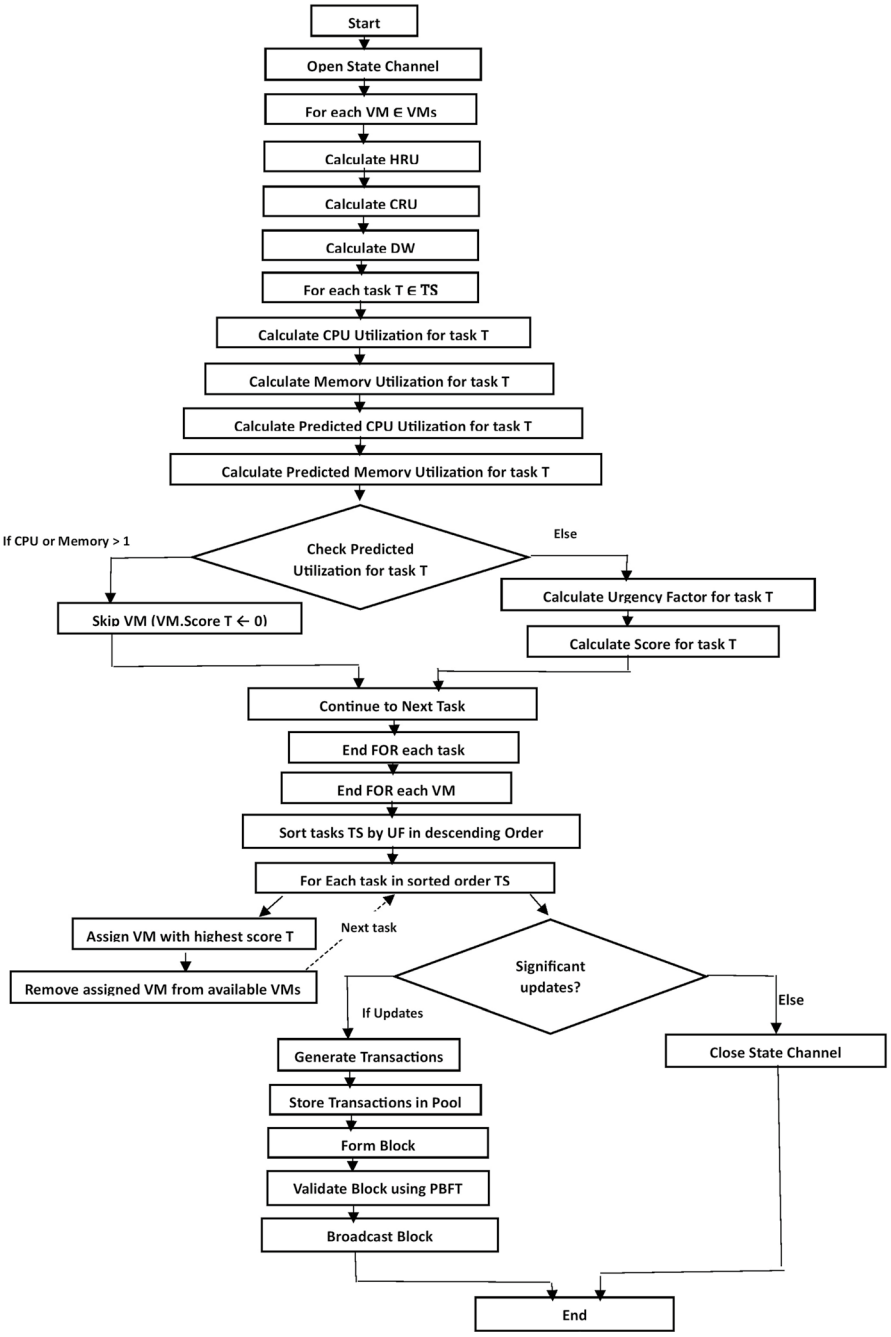

Fig. 3 Flowchart of the proposed load balancing scheme

A higher score suggests that the VM can handle the task more effectively with lower resource contention and greater urgency handling. Hence, tasks are assigned to the VM with the highest score. Figure 3 illustrates the processes of the proposed load balancing scheme.

To elaborate further, consider the following concrete example. Consider a system with three virtual machines (VMs), each with 1000 CPU units, 500 memory units, 600 I/O units, and 60 Mbps bandwidth. Current utilization: VM1 is heavily loaded (800 CPU, 400 memory, 300 I/O, 30 Mbps), VM2 is lightly loaded (300 CPU, 200 memory, 150 I/O, 15 Mbps), and VM3 is moderately loaded (400 CPU, 250 memory, 200 I/O, 20 Mbps). Historical usage trends for each VM show varying resource utilization over three intervals, with VM1's usage peaking at 800 CPU, 400 memory, 300 I/O, and 30 Mbps. Two tasks are to be scheduled: Task 1 (τ1\tau_1τ1) requires 250 CPU, 100 memory, 200 I/O, and 10 Mbps with a 1:00 PM deadline, while Task 2 (τ2\tau_2τ2) requires 300 CPU, 200 memory, 100 I/O, and 15 Mbps with a 12:30 PM deadline. The current time is 12:00 PM.

Dynamic Weight Calculations

By using (1), $CRU_{VM1} = 2.6$, $CRU_{VM2} = 1.2$ and $CRU_{VM3} = 1.566$
By using (2), $HRU_{VM1} = 1.97$, $HRU_{VM2} = 0.709$ and $HRU_{VM3} = 1.036$
By using (3), $DW_{VM1} = 2.285$; $DW_{VM2} = 0.954$; $DW_{VM3} = 1.301$

Task Assignment on VM1

Task 1 (τ1)	Task 2 (τ2)
Predicted CPU Utilization = 1050/1000 = 1.05	Predicted CPU Utilization = 1100/1000 = 1.1
Predicted Memory Utilization = 500/500 = 1.0	Predicted Memory Utilization = 600/500 = 1.2
Predicted I/O Utilization = 500/600 = 0.83	Predicted I/O Utilization = 400/600 = 0.67
Predicted Bandwidth Utilization = 40/60 = 0.5	Predicted Bandwidth Utilization = 45/60 = 0.75

Since the predicted CPU exceeds 1, VM1 is skipped for Task 1. Also, both predicted CPU and memory exceed 1, so VM1 is skipped for Task 2.

Task Assignment on VM2

Task 1 (τ1)	Task 2 (τ2)
Predicted CPU Utilization = 550/1000 = 0.55	Predicted CPU Utilization = 600/1000 = 0.6
Predicted Memory Utilization = 300/500 = 0.6	Predicted Memory Utilization = 400/500 = 0.8
Predicted I/O Utilization = 350/600 = 0.58	Predicted I/O Utilization = 250/600 = 0.42
Predicted Bandwidth Utilization = 25/60 = 0.42	Predicted Bandwidth Utilization = 30/60 = 0.5

Task Assignment on VM3

Task 1 ($\tau 1$)	Task 2 ($\tau 2$)
Predicted CPU Utilization = 650/1000 = 0.65	Predicted CPU Utilization = 700/1000 = 0.7
Predicted Memory Utilization = 350/500 = 0.7	Predicted Memory Utilization = 450/500 = 0.9
Predicted I/O Utilization = 400/600 = 0.67	Predicted I/O Utilization = 300/600 = 0.5
Predicted Bandwidth Utilization = 30/60 = 0.5	Predicted Bandwidth Utilization = 35/60 = 0.58

Urgency Factor and Scores

Task 1 ($\tau 1$): UF = 1.0, Task 2 ($\tau 2$): UF = 2.0. By using (12),

Task 1 ($\tau 1$) Scores: VM1: = 0.438; VM2: = 1.048; VM3: = 0.769

Task 2 ($\tau 2$) Scores: VM1: = 0.876; VM2: = 2.098; VM3: = 1.537

The VM2 has the highest score for both tasks, but Task 2 is prioritized due to its higher urgency. Task 1 is assigned to VM3. After assigning the tasks, the state channel is closed, and any significant updates are recorded on the blockchain. The proposed scheme ensures that VMs close to capacity are skipped, successfully to balancing the load.

Task Execution

The task execution module orchestrates the operational phase within the cloud environment and ensures optimal utilization of resources, maintaining system stability and performance. Once a task is validated and assigned to a VM based on previous load balancing decisions, which include assessments of dynamic weights, predicted utilizations, scores and task urgency, it is ready for execution. The initiation process involves starting the task on the designated VM, which then allocates its computational resources (CPU, memory, etc.) to execute the task's requirements. As the task executes, the VM's resources are utilized according to the task's specific demands. The system actively monitors this resource consumption in real-time to ensure that it does not exceed the available capacity of the VM, which could lead to performance degradation. This is achieved by dynamically adjusting resource allocations based on ongoing consumption rates, thus preventing any potential overload. Concurrently, the module continuously oversees the task's execution status. The system is designed to respond to the issues by logging errors, issuing alerts, and potentially initiating corrective actions such as task retries or reassignments. If a task completes successfully, the event is recorded on the blockchain, creating an immutable transaction record that includes details such as the task ID, VM ID, execution duration, resources used, and the outcome. This blockchain integration ensures that the execution details are transparent and tamper-proof, enhancing trust and accountability within the distributed network. In case of task failure, the system logs the failure, removes the task from the VM's active task list, and retries the task depending on available retries. If all retries are exhausted, it records the task as failed after all attempts. This comprehensive error-handling approach helps maintain system reliability and informs necessary stakeholders about potential issues.

Blockchain Recording, Notification and Logging

This module logs every detail of task executions on VMs to a blockchain, creating a permanent, tamper-proof ledger that enhances trust. For each task executed, it records details such as task identity, the VM used, execution result (success or failure), duration, and resources consumed. The Notification and Logging module performs two main functions: notifying stakeholders of task outcomes and logging system errors. It ensures that all stakeholders, including users and administrators, receive timely updates on task statuses, such as success or failure, along with detailed insights like execution time, resource usage, and outputs. Additionally, the module logs errors, including execution failures and configuration issues, providing detailed information for troubleshooting. Through real-time notifications and error logging, the module enhances operational transparency and strengthens error-handling capabilities in the cloud environment.

8 Complexity analysis

(i) Task Validation and VM Selection Algorithm

Time Complexity and Space Complexity For each VM, the algorithm computes HRU, CRU, and DW, each taking $O(1)$ time. For each task, checking predicted resource utilization for all VMs takes $O(m)$, where m is the number of VMs. Sorting the tasks by urgency requires $O(n \log n)$, where n is the number of tasks. The overall time complexity of Algorithm 1 is: $O(m{\cdot}n) + O(n \log n) = O(m{\cdot}n)$. The space complexity is $O(m+n)$, as it stores data for both tasks and VMs.

Theoretical Bound on Performance The worst-case time complexity of this algorithm is $O(m{\cdot}n)$, where m is the number of VMs and n is the number of tasks, dominated by task validation and urgency-based sorting. The best-case complexity is $\Omega(n \log n)$, occurring when minimal tasks or VMs are involved. The tight bound is $\Theta(m{\cdot}n)$, ensuring scalability for large systems.

(ii) Blockchain Update and Notification Algorithm

Time Complexity and Space Complexity: Updating the blockchain involves appending a transaction, which takes O(1). Notification processes (checking task result and notifying the submitter) also take constant time O(1). Thus, the time complexity for Algorithm 2 is: O(1). The space complexity is O(1) as it only stores the transaction record and task execution results.

Theoretical Bound on Performance This algorithm operates with a constant time complexity O(1) for both updating the blockchain and notifying stakeholders. Its best-case complexity is also $\Omega(1)$, and the tight bound remains $\Theta(1)$.

(iii) Retry Handling Algorithm

Time Complexity and Space Complexity: The retry mechanism ensures robustness by revalidating tasks with exponential backoff delays. It iteratively selects the optimal VM until all retries are exhausted. Let R denote the maximum number of retries. For each retry, task validation and VM selection require O(V).

Theoretical Bound on Performance The worst-case complexity of this algorithm is $O(r{\cdot}m{\cdot}n)$, where r is the retry count, m is the number of VMs, and n is the number of tasks, as it retries task validation and VM selection for a maximum of r retries. The best-case complexity is $\Omega(m{\cdot}n)$, occurring when tasks succeed on the first attempt, and the tight bound remains $O(r{\cdot}m{\cdot}n)$.

The overall system complexity is dominated by the Task Validation and VM Selection Algorithm, resulting in a worst-case complexity of $O(m{\cdot}n)$ and a best-case complexity of $\Omega(n \log n)$. The tight bound remains $\Theta(m{\cdot}n)$, ensuring the system is scalable and efficient for dynamic, large-scale cloud environments. Algorithm 1 scales linearly with VMs (m) and tasks (n) with a time complexity of $O(m{\cdot}n)$. Algorithm 2 operates in constant time O(1), ensuring minimal overhead, while Algorithm 3 efficiently handles failures with $O(r{\cdot}m{\cdot}n)$ complexity. Together, these algorithms ensure predictable performance, minimal delays, and robust scalability under varying workloads.

9 Formal proofs of algorithm correctness

1. Task Validation and VM Selection Algorithm

The correctness of this algorithm is derived from the following properties:

(i) *Task Completeness Validation* The algorithm ensures that every task is evaluated for resource requirements (CPU, memory, and I/O). Tasks exceeding any resource constraint are excluded from being assigned to overloaded VMs.

Proof For each task τ, the algorithm computes: $\text{Predicted CPU Utilization} = \frac{\text{CPU Requirement}}{\text{CPU capacity of VM}}$

Similar calculations are performed for memory and I/O. If any predicted utilization exceeds 111, the VM is skipped, ensuring only feasible assignments are considered. This guarantees that tasks are assigned to VMs with sufficient capacity.

(ii) *Dynamic Weight (DW) Calculation* The algorithm combines historical and current resource usage to ensure balanced task distribution.

Proof Dynamic Weight is calculated by using the formula, $\text{DW} = \frac{\alpha{\cdot}\text{CRU}+\beta{\cdot}\text{HRU}}{1}$

The DW calculation uses weights α and β, ensuring fairness by balancing short-term and long-term resource usage data. This prevents overloading any specific VM and promotes equitable resource allocation.

(iii) *Task Assignment Based on Prioritization* Tasks are sorted by their urgency factor (UF) and assigned first to the most suitable VM.

Proof The urgency factor ensures tasks closer to their deadlines are prioritized.

$$\text{UF (task)} = \frac{1}{\tau \cdot \text{Deadline} - \text{CurrentTime}}, \text{Score} = \frac{\text{UF (task)}}{\text{DW (VM)}}$$

Tasks are assigned to VMs with the highest score. This ensures optimal prioritization and effective load balancing.

2. Blockchain Update and Notification Algorithm

(i) *Secure Recording* Task assignments and results are appended to the blockchain, ensuring transparency and immutability.

Proof The blockchain is updated with each transaction: $\text{Blockchain}_{\text{new}} = \text{Blockchain} \cup \{\text{TR}\}$

The cryptographic integrity of the blockchain guarantees that recorded transactions cannot be altered.

(ii) *Result Notification* Notifications are sent to stakeholders based on the task execution result.

Proof For a successful execution result (ER = Success), the submitter is notified. For failed results, errors are logged, and an admin is alerted. This ensures reliable tracking of all task outcomes and seamless communication.

3. Retry Handling Algorithm

(i) E*xponential Backoff Mechanism* The retry mechanism prevents resource contention during high load periods.

Proof For each retry i, the delay is calculated as: $\text{Delay} = 2^{i}$. Base Delay

This ensures increasing intervals between retries, reducing system load while maintaining retry efficiency.

(ii) *Guaranteed Completion or Failure* The algorithm retries a task up to R times, marking it as failed if all retries are exhausted.

Proof After R retries, if the task still fails, it is recorded as: Retry Status = Failed after all retries

This guarantees that every task is either successfully completed or definitively marked as failed, ensuring reliability.

The formal proofs validate the correctness of the proposed algorithms in achieving their objectives. The Task Validation and VM Selection Algorithm ensures optimal task assignments while balancing resource utilization. The Blockchain Update and Notification Algorithm guarantees transparent, secure, and immutable task tracking. The Retry Handling Algorithm ensures robust handling of failures, avoiding system overload. Together, these algorithms establish an efficient, reliable, and secure framework for decentralized dynamic load balancing in cloud computing.

10 Performance evaluation

To assess the efficiency, effectiveness, and scalability of the proposed blockchain-enabled load balancing scheme, the following performance metrics are employed:

Makespan Makespan refers to the total time required to complete all assigned tasks. A shorter makespan indicates improved system efficiency and faster task processing across varying workloads [30–32]. It is calculated by using the formula,

$$\text{MakeSpan} = \text{Max}(\text{Task Completion Time}) \tag{13}$$

$$\text{MakeSpan}_{\text{avg}} = \left(\sum \frac{\text{Max}(\text{Task Completion Time})}{\text{No. of Virtual Machines}} \right) \tag{14}$$

Execution Time Execution time is defined as the duration required to execute an individual task. Lower execution times indicate efficient task allocation and resource utilization, reflecting the scheme's capability to handle complex workloads promptly [30–32]. It is calculated by using the formula,

$$\text{Execution Time} = \text{Actual CPU Time} \tag{15}$$

$$\text{Execution Time}_{\text{avg}} = \frac{\Sigma(\text{Actual CPU Time})}{\text{No. of Tasks}} \tag{16}$$

Resource Utilization Resource utilization measures how effectively resources are utilized [30]. Higher utilization reflects efficient usage but may indicate overloading if excessively high, potentially causing performance degradation [31, 32]. It is calculated by using the formula,

$$\text{Avg.RU} = \frac{\text{Execution Time}}{\text{Make Span}} \times 100 \tag{17}$$

Throughput Throughput measures the number of tasks successfully processed per unit of time. Higher throughput signifies enhanced operational capacity, enabling the system to efficiently handle high volumes of tasks under varying load conditions [30]. It is calculated by using the formula,

$$\text{Throughput} = \frac{\text{No. of tasks}}{\text{Average Makespan (ms)} \times 0.001} \tag{18}$$

Latency Latency captures the time delay between task initiation and completion. Reduced latency is critical for time-sensitive applications, demonstrating the system's responsiveness and ability to manage workloads with minimal delays. It is calculated by using the formula:

$$\text{Latency} = \text{Task Arrival Time} - \text{Task Completion Time} \tag{19}$$

$$\text{Latency}_{\text{avg}} = \frac{\sum (\text{Task Arrival Time} - \text{Task Completion Time})}{\text{No. of Tasks}} \tag{20}$$

Load Balance Index The load balance index quantifies the uniformity of task distribution across virtual machines (VMs). A lower LBI value indicates a balanced workload distribution, preventing resource bottlenecks and promoting consistent performance across the system. The formula for load balance index (LBI) is:

$$\text{LBI} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(L_i - \overline{L})^2} \tag{21}$$

$$\overline{L} = \frac{1}{N}\sum_{i=1}^{N} L_i \tag{22}$$

where N is the number of Virtual Machines, L_i is the load on the ith virtual machine, $\overline{L}$ is the average load across all virtual machines.

Evaluation with Real-World Cloud Workload Traces and Varied Scenarios

The evaluation of the proposed blockchain-enabled load balancing scheme involved testing under real-world cloud workload traces and varied scenarios to ensure comprehensive validation. Specifically, the system was tested using a mix of real-world workloads, including CPU-bound, memory-bound, and I/O-bound tasks, simulating typical cloud environments. A total of 100 tasks were distributed across 10 virtual machines (VMs) with heterogeneous resource capacities (CPU, memory, and I/O), reflecting a realistic cloud infrastructure. To further assess the robustness of the proposed scheme, it was tested under various network conditions (high latency, low bandwidth, packet loss) as well as workload patterns such as constant load and burst load. The proposed scheme consistently demonstrated low latency and efficient resource utilization, regardless of network

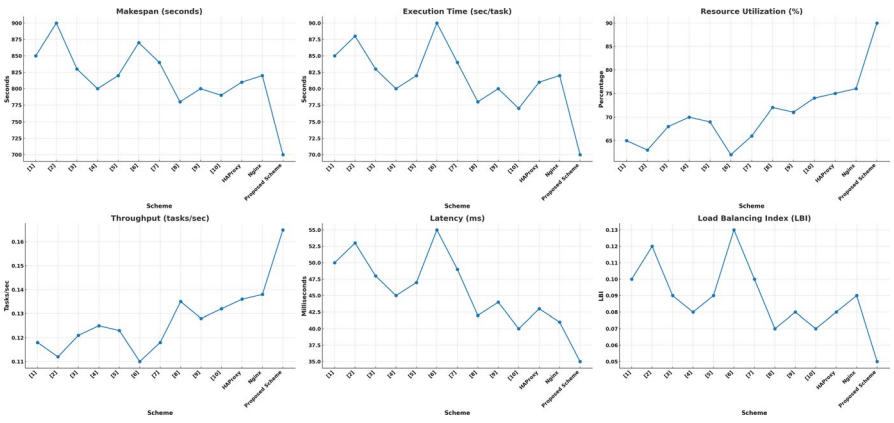

Fig. 4 Sudden traffic spikes

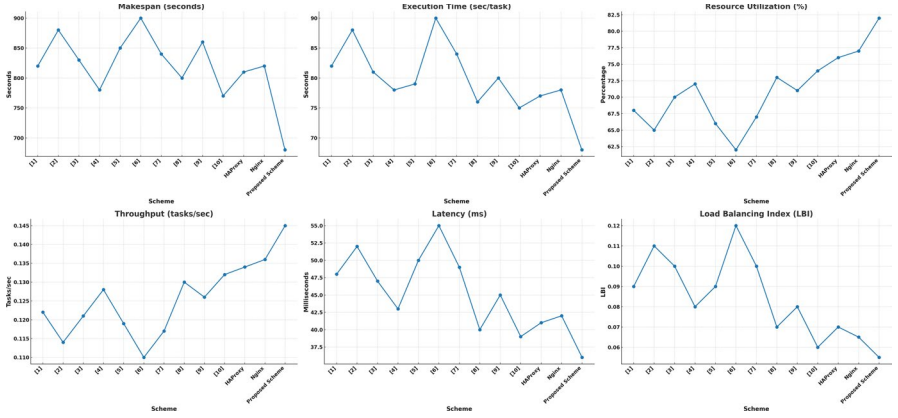

Fig. 5 Heterogeneous workloads

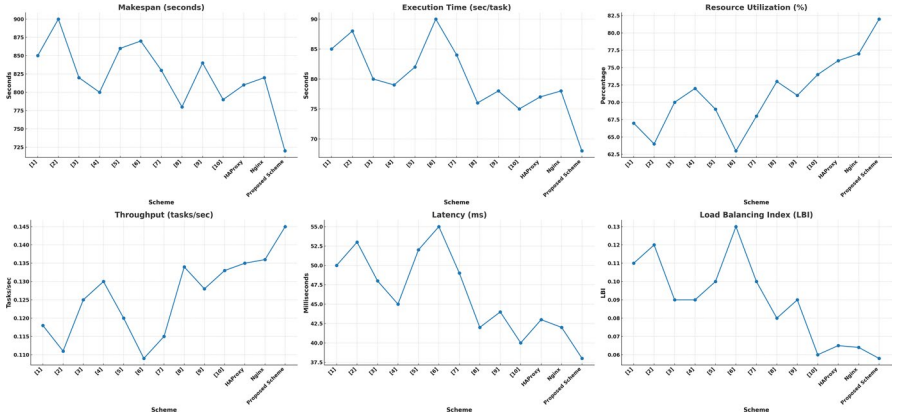

Fig. 6 Geographically distributed cloud

variability, showcasing its adaptability to real-time cloud environments. The system was also tested under the following conditions:

1. *Sudden Traffic Spikes* The system effectively handled unexpected load surges maintaining low latency and efficiently managing task assignments, even when blockchain recording was involved.
2. *Heterogeneous Workloads* The mix of different resource-bound tasks tested the system's adaptability to diverse workload types. The proposed scheme achieved superior task assignment efficiency, reducing latency.
3. *Geographically Distributed Cloud* The proposed scheme was evaluated in a geographically distributed cloud, where latency can increase due to network delays. Even in this scenario, the scheme minimized latency through optimized task scheduling and resource allocation.

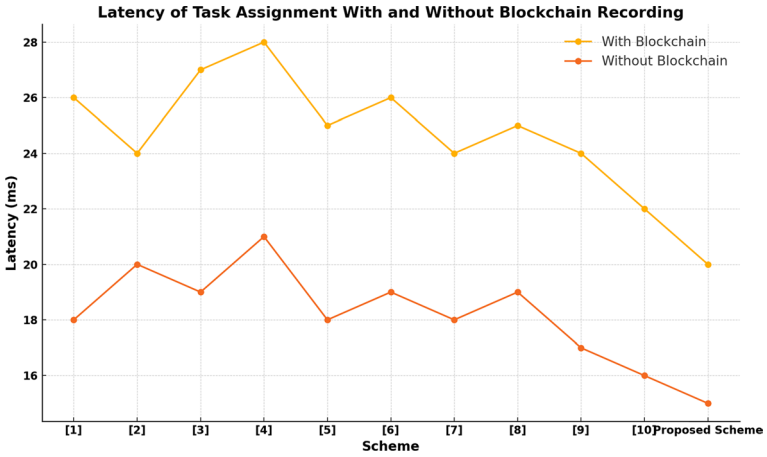

Fig. 7 Latency of task assignment with and without blockchain recording

Figures 4, 5 and 6 represent the performance evaluation results during Sudden Traffic Spikes, Heterogeneous Workloads and in Geographically Distributed Cloud. In Fig. 7, the proposed scheme is tested under two conditions: with and without blockchain recording. The results show that blockchain introduces additional latency due to the consensus and recording processes required for blockchain transactions. However, the proposed scheme manages to minimize this impact, as shown by the lower latency compared to other schemes in both conditions.

This comprehensive testing, involving real-world traces, varied scenarios, and challenging network conditions, confirms the effectiveness of the proposed scheme in minimizing latency while leveraging blockchain for transparency and security. Thus, the proposed blockchain-enabled load balancing scheme outperforms existing approaches by consistently delivering low latency, high resource utilization, and efficient task assignment.

Blockchain-Specific Metrics Evaluation

The proposed blockchain-enabled decentralized load balancing system has been evaluated using real-world cloud workload traces from Google cluster data, providing insights into its efficiency and scalability. Key blockchain-specific performance metrics are analyzed to assess the impact of blockchain integration on task allocation and execution.

1. Block Time Block time refers to the average time required to validate and append a block containing task-related transactions to the blockchain. Lower block time ensures minimal delays in blockchain updates for task assignments and completions. Block time can be calculated by using the formula,

$$\text{BlockTime}_{\text{avg}} = \left(\sum_{i=1}^{N} \text{BlockTime}_i \right) / N$$

where BlockTime_i represents the time required for the ith block validation, and N is the total number of blocks validated.

2. Gas Costs Gas costs measure the computational resources consumed for executing blockchain transactions, such as recording task assignments and resource usage. Efficient task management minimizes redundant transactions, reducing overall gas costs. Gas Costs can be calculated by using the formula,

$$\text{GasCost}_{\text{total}} = \sum_{i=1}^{M} \text{Gas}_i$$

where Gas_i is the gas consumed by the ith transaction, and M is the total number of transactions recorded during evaluation.

3. Transaction Throughput Transaction throughput measures the number of task-related transactions processed and recorded by the blockchain per unit of time. High throughput ensures the blockchain can handle frequent task updates, even during sudden traffic spikes, without becoming a bottleneck. Transaction throughput can be calculated by,

$\text{Throughput}_{\text{blockchain}} = \frac{\text{Total Transactions}}{\text{Total Time}}$Where Total Transactions is the number of blockchain transactions processed, and Total Time is the duration of the evaluation period.

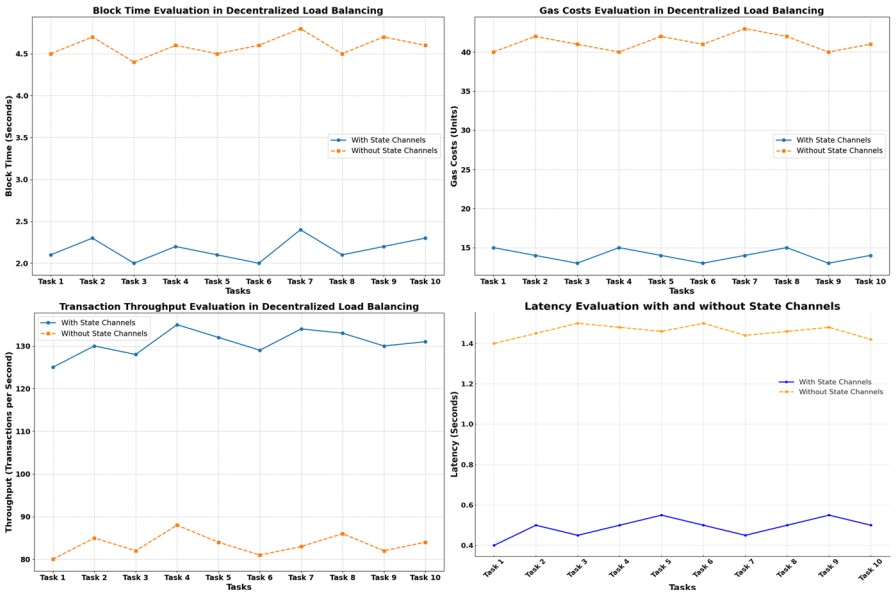

Fig. 8 Evaluation of blockchain metrics with and without state channels

4. Latency Reduction Percentage Latency reduction percentage measures the improvement in transaction processing times when optimization mechanisms, such as state channels, are employed. A higher latency reduction percentage indicates the effectiveness of optimization techniques like state channels in minimizing delays, enhancing the overall performance of the load balancing system.

$$\text{Latency Reduction}\% = \frac{\text{Latency}_{\text{wo}} - \text{Latency}_{\text{w}}}{\text{Latency}_{\text{wo}}} \times 100$$

where $\text{Latency}_{\text{wo}}$ and $\text{Latency}_{\text{w}}$ represent the blockchain transaction latency without and with state channels, respectively (Fig. 8).

The evaluation of the proposed decentralized load balancing system highlights significant improvements with state channels, including a 52% reduction in block time from 4.6 to 2.2 s, a 66% reduction in gas costs from 41 to 14 units, and a 55% increase in transaction throughput from 84 to 130 transactions per second. Latency decreased by 66%, averaging 0.490 s with state channels compared to 1.459 s without. These results demonstrate the system's efficiency, cost-effectiveness, and ability to handle high-demand cloud workloads.

11 Conclusion and future works

The research presented a novel blockchain-enabled load balancing algorithm that significantly enhances the efficiency and effectiveness of cloud computing operations. The proposed solution effectively integrates blockchain's immutable and decentralized features to improve task distribution and execution within cloud data centers. Performance evaluations highlight the proposed algorithm's superiority in reducing makespan and execution times while achieving higher resource utilization and throughput compared to traditional schemes. This advancement not only supports more robust cloud operations but also contributes to the scalability and adaptability of cloud services to meet the demands of increasingly complex workloads. For future work, we plan to assess the applicability of our blockchain-enabled load balancing algorithm in edge computing environments. This exploration will aim to extend the benefits of our approach, such as reduced latency and improved data traffic management, beyond centralized data centers to decentralized networks. This shift is expected to enhance service delivery and operational efficiency at the network's edge, where data processing demands are rapidly increasing.

Author contribution JR and JII are the authors of the manuscript, drafting sections related to the introduction, methodology, results, and discussion. JR provided the foundational conceptual framework and strategic guidance for integrating blockchain technology with load balancing techniques. Her insights were pivotal in shaping the direction of the research. She supervised the development of the methodology, ensuring that the proposed approaches were robust, innovative, and aligned with current advancements in cloud computing and blockchain technology. She provided critical feedback and support, fostering a rigorous and comprehensive research process. She reviewed and edited the manuscript, ensuring the academic rigor, clarity, and coherence of the research narrative. Her feedback was essential in refining the

research findings and their presentation. JII implemented and refined the algorithms for task validation, VM selection, and task prioritization, optimizing them for efficiency and performance. She conducted extensive simulations using the CloudSim toolkit, analyzing the performance of the proposed scheme against existing methods. Her work demonstrated significant improvements in makespan, execution time, resource utilization, and throughput. She meticulously documented the results, ensuring their accuracy and reliability. She conducted a comprehensive literature review, identifying gaps in existing research and positioning the proposed scheme within the broader context of cloud computing and blockchain technology.

Data availability No datasets were generated or analyzed during the current study.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Rehman AU et al (2020) Dynamic energy efficient resource allocation strategy for load balancing in fog environment. IEEE Access 8:199829–199839
2. Sharma S, Singh S, Sharma M (2008) Performance analysis of load balancing algorithms. Int J Civ Environ Eng 2(2):367–370
3. Devine KD et al (2005) New challenges in dynamic load balancing. Appl Numer Math 52(2–3):133–152
4. Aslanpour MS et al (2024) Load balancing for heterogeneous serverless edge computing: a performance-driven and empirical approach. Future Gener Comput Syst 154:266–280
5. Javadi SA, Gandhi A (2019) User-centric interference-aware load balancing for cloud-deployed applications. IEEE Trans Cloud Comput 10(1):736–748
6. Ghomi EJ, Rahmani AM, Qader NN (2017) Load-balancing algorithms in cloud computing: a survey. J Netw Comput Appl 88:50–71
7. Asghar A et al (2021) Fog based architecture and load balancing methodology for health monitoring systems. IEEE Access 9:96189–96200
8. Babou CSM et al (2020) Hierarchical load balancing and clustering technique for home edge computing. IEEE Access 8:127593–127607
9. Watts J, Taylor S (1998) A practical approach to dynamic load balancing. IEEE Trans Parallel Distrib Syst 9(3):235–248
10. Mishra SK, Sahoo B, Parida PP (2020) Load balancing in cloud computing: a big picture. J King Saud Univ Comput Inf Sci 32(2):149–158
11. Al Nuaimi K et al (2012) A survey of load balancing in cloud computing: Challenges and algorithms. In: 2012 second symposium on network cloud computing and applications. IEEE (2012)
12. Devi N et al (2024) A systematic literature review for load balancing and task scheduling techniques in cloud computing. Artif Intell Rev 57(10):276
13. Mattia GP, Pietrabissa A, Beraldi R (2023) A load balancing algorithm for equalising latency across fog or edge computing nodes. IEEE Trans Serv Comput 16:3129–3140
14. Nayyer MZ et al (2022) LBRO: load balancing for resource optimization in edge computing. IEEE Access 10:97439–97449
15. Khiyaita A et al (2012) Load balancing cloud computing: state of art. Natl Days Netw Secur Syst 25(2012):106–109
16. Semmoud A et al (2020) Load balancing in cloud computing environments based on adaptive starvation threshold. Concurr Comput Pract Exp 32(11):e5652
17. Subrata R, Zomaya AY, Landfeldt B (2007) Game-theoretic approach for load balancing in computational grids. IEEE Trans Parallel Distrib Syst 19(1):66–76
18. Sthapit S et al (2018) Computational load balancing on the edge in absence of cloud and fog. IEEE Trans Mob Comput 18(7):1499–1512
19. https://status.cloud.google.com/incidents/UPG5wxRnLGjqqVFMW7Kq

20. Zhao J et al (2015) A heuristic clustering-based task deployment approach for load balancing using Bayes theorem in cloud environment. IEEE Trans Parallel Distrib Syst 27(2):305–316
21. Duan J, Yang Y (2017) A load balancing and multi-tenancy-oriented data center virtualization framework. IEEE Trans Parallel Distrib Syst 28(8):2131–2144
22. Kumar M, Sharma SC (2020) Dynamic load balancing algorithm to minimize the makespan time and utilize the resources effectively in cloud environment. Int J Comput Appl 42(1):108–117
23. Tang F et al (2016) A dynamical and load-balanced flow scheduling approach for big data centers in clouds. IEEE Trans Cloud Comput 6(4):915–928
24. Gamal M et al (2019) Osmotic bio-inspired load balancing algorithm in cloud computing. IEEE Access 7:42735–42744
25. Junaid M et al (2020) A hybrid model for load balancing in cloud using file type formatting. IEEE Access 8:118135–118155
26. Hung L-H et al (2021) Migration-based load balance of virtual machine servers in cloud computing by load prediction using genetic-based methods. IEEE Access 9:49760–49773
27. Sohani M, Jain SC (2021) A predictive priority-based dynamic resource provisioning scheme with load balancing in heterogeneous cloud computing. IEEE Access 9:62653–62664
28. Kruekaew B, Kimpan W (2022) Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning. IEEE Access 10:17803–17818
29. Reshan Al, Saleh M et al (2023) A fast converging and globally optimized approach for load balancing in cloud computing. IEEE Access 11:11390–11404
30. Shafiq DA et al (2021) A load balancing algorithm for the data centres. IEEE Access 9:41731–41744
31. Alboaneen D et al (2021) A metaheuristic method for joint task scheduling and virtual machine placement in cloud data centers. Future Gener Comput Syst 115:201–212
32. Konjaang JK, Murphy J, Murphy L (2022) Energy-efficient virtual-machine mapping algorithm (EViMA) for workflow tasks with deadlines in a cloud environment. J Netw Comput Appl 203:103400

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.