SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

January 2022, Vol. 65 112102:1–112102:16
https://doi.org/10.1007/s11432-020-2939-3

Auto-scalable and fault-tolerant load balancing mechanism for cloud computing based on the proof-of-work election

Xiaoqin FENG[1,3], Jianfeng MA[1,2*], Shaobin LIU[1,2], Yinbin MIAO[1,3] & Ximeng LIU[4]

[1]*School of Cyber Engineering, Xidian University, Xi'an* 710071*, China;*
[2]*Shaanxi Key Laboratory of Network and System Security, Xidian University, Xi'an* 710071*, China;*
[3]*State Key Laboratory of Cryptology, Beijing* 100878*, China;*
[4]*College of Mathematics and Computer Science, Fuzhou University, Fuzhou* 350108*, China*

Received 8 January 2020/Revised 8 May 2020/Accepted 4 June 2020/Published online 16 December 2021

Abstract Load balancing mechanism in technologies such as cloud computing has provided a huge opportunity for the development of large-scale projects. Although the conventional view is to build mechanisms that adopt a dynamic load balancing strategy, existing strategies cannot automatically scale platform (network) servers (nodes) to adapt for dynamic requests, but only guarantee load balancing for the pre-deployed nodes, thereby increasing resource consumption and decreasing networks' efficiency. We contend that existing load balancing mechanisms are inadequate for deploying dynamic applications. In this regard, we first adopt both load balancing and cloud computing virtualization technologies to modularly design a load balancing mechanism that provides a dynamically auto-scalable solution for large-scale and dynamic computing scenarios. Furthermore, we adopt the proof-of-work consensus, for a novel use during the lifecycle of master nodes in case of system failure caused by a failed master node, to demonstrate a fault-tolerant load balancing mechanism. We theoretically evaluate the security requirement of our mechanism and analyze its performance. Experimental results show that the mechanism supports auto-scalability and has a better performance compared to existing mechanisms such as the ordinary cluster.

Keywords cloud computing, load balancing, auto-scalable, proof-of-work, fault-tolerant

Citation Feng X Q, Ma J F, Liu S B, et al. Auto-scalable and fault-tolerant load balancing mechanism for cloud computing based on the proof-of-work election. Sci China Inf Sci, 2022, 65(1): 112102, https://doi.org/10.1007/s11432-020-2939-3

1 Introduction

Load balancing mechanisms are promising concepts for large-scale infrastructures, such as cloud computing platform [1]. Figure 1 describes operating principles of load balancing technology, which essentially assigns virtual resources to requests. Traditionally, existing load balancing mechanisms are mechanized — particularly incapable of auto-scalability — and several illustrative examples are described in this study. The honey bee optimization (HBO)-based load balancing scheme [2] used the foraging behavior of honey bees to find the optimal solutions for transaction scheduling, but the system load was balanced in advance. The hash-based traffic steering on softswitches (HATS) method [3] was designed for chaining virtualized network functions (VNFs) [4] to mitigate control and data plane overheads. However, hashing weights of VNFs and network paths have to be periodically updated with load status. The load balancing method in [5] observes the intensity of incoming load and the proportion of total requests that go to each firewall; this method was designed for dynamic cloud firewall services. But the number of firewall services remains constant even though the number of requests keeps changing. Moreover, several load balancing mechanisms proposed in [6–9] have new problems such as fault-intolerance, where the crash of master nodes may cause the entire system to be inoperative.

* Corresponding author (email: jfma@mail.xidian.edu.cn)

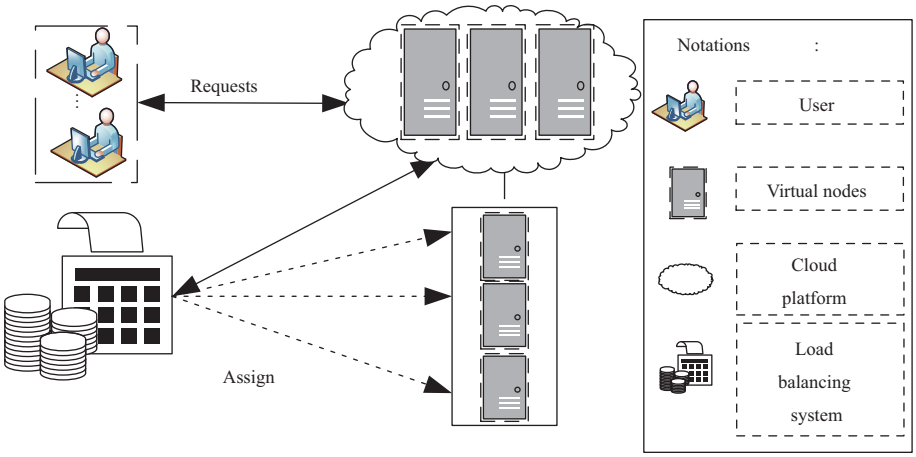

Figure 1 (Color online) Load balancing technology.

In recent years, the emergence of Blockchain [10–12] has spurred the surprising success of Bitcoin [13]. Blockchain network challenges our conventional belief on fault tolerance through the efficient use of hybrid storage and a consensus mechanism. Unlike existing load balancing mechanisms, Blockchain has shown a new application of consensus mechanisms over a public or private network, and among a large number of mutually distrustful and independent nodes. Thus, this paper introduces a novel design idea for fault-tolerant load balancing mechanism.

Fault tolerance is an essential consideration. Blockchain demonstrates the use of individual node to achieve fault tolerance by adopting a consensus, even at the expense of computing. The consensus nodes in many classic Blockchain protocols are confronted with huge computing demands: each participating node performs complex mathematical calculations, and the complex design protects the system from 51% attack and keeps a stable block production. The cloud computing nodes are designed with the capacity to compute [14], and the operation of this platform is similar to the proof-of-work (PoW)-based [15, 16] system, unlike other systems such as proof-of-stake (PoS), delegated proof-of-stake (DPoS), thus, PoW would be a suitable candidate for cloud nodes consensus. We note that the high demand for fault tolerance load balancing mechanisms with the concept of master nodes is often because of the demand for a consensus paradigm, since a consensus paradigm proposes a novel solution for voting-based rule and would essentially be void of the total system failure caused by malfunctioned centralized network nodes. Also, the pressing demand for an efficient load balancing mechanism and a private cloud platform with essential security just require a rather simple mathematical computation in PoW. Besides, the simple solution verification and mutual management of platform nodes support an effective and robust maintenance mechanism for consensus nodes.

1.1 Our contributions

Mechanization and resources-wasting of the conventional load balancing mechanism are considered unpractical.

• Previous work on load balancing mechanisms, even the so-called "automatic" mechanisms, only balance load for the existing nodes deployed on corresponding business applications, where the creation and deployment of new nodes on business applications are manually done, and the configuration of load balancing components are also modified manually. We contend that such mechanisms are unpractical and not suitable for scenarios with dynamic requests. To demonstrate this, we explicitly construct an auto-scalable load balancing mechanism, a system that is sufficiently efficient for dynamic scenarios.

• Load balancing mechanisms are highly demanded in large-scale infrastructures. Therefore, resources-wasting significantly degrades the effectiveness of such infrastructures, and a way of deactivating idle nodes when the system is not at full operation is urgently needed.

Auto-scalable and fault-tolerant load balancing mechanism. We propose a proof-of-work based load balancing mechanism (PWB-LB), which provides automatic and fault-tolerant service in a private cloud computing environment.

We improve on extant mechanisms to realize an automatic load balancing mechanism. The mechanization of extant mechanisms is caused by their inability to automatically deploy new nodes. We propose a

novel load balancing mechanism to overcome this problem by using the cloud computing virtualization technology to timely adjust cloud computing resources, automatically build new task processing nodes through mirroring, and dynamically scale the load balancing capacity in networks.

The major bottleneck in the master nodes' settings is their crash tendency or malware threat. It is worth noting that the PoW consensus mainly focuses on the maintenance of master nodes and the competitions among them. The confirmation of mater nodes' agreement is achieved by solving the moderately difficult mathematical problem and simple hash verifications in PoW among the whole cloud platform. With the necessary computing consumptions during competitions, the malicious competitors lose more than what they can acquire, then they will not misbehave. Besides, the whole cloud platform maintains all master nodes by simple hash verifications, which further prevents malicious tendencies from master nodes and initiates the new master nodes voting once the current master node is down. Different from common voting algorithms whose responsibility simply is voting based on different rules in competition models, reliability and agreement of master nodes considered in this paper show that the proposed load balancing mechanism can guarantee fault-tolerance in systems.

Furthermore, the execution of PoW consensus for frequent master nodes' voting is a waste of time and will delay the processing of users' requests. Therefore, the idea of redundant masters is adopted to select a node from the masters' queue to sequentially work as the current master node. Then, the fault-tolerant load balancing mechanism also provides high efficiency.

In a private cloud computing environment, PWB-LB accomplishes the dynamic load balancing task and achieves the fault tolerance of master nodes. Unlike previous "automatic" mechanisms where network load balancing scales inconveniently, PWB-LB can automatically scale the load capacity of networks, including the addition and removal of nodes. This new load balancing mechanism is designed for automatic load balancing strategy and scaling of network load capacity. The load balancing mechanism reasonably allocates tasks based on the current load status of each task processing node in the cluster [17, 18] (the cluster consists of multiple computing nodes deployed on business applications, and these nodes work synchronously to complete tasks), dynamically manages the cloud computing resources, and maintains the load equilibrium for all the processing nodes on the cloud platform to achieve load balancing. Besides, the load balancing mechanism is deployed on a private cloud computing platform, and simple master nodes selection is achieved through PoW consensus to ensure fault tolerance in the platform. We formally validated the security and performance of PWB-LB, and show experimentally that it provides auto-scalability and efficient load balancing service compared to other existing mechanisms, especially in terms of high-concurrency and processing difficult queries– processing these kinds of queries incurs high time complexity.

1.2 Deployment scenarios

PWB-LB is suitable for different deployment scenarios sought after by private cloud platforms [19] such as electronic commerce (E-commerce). Among numerous conceivable genres of E-commerce, we highlight an online shopping platform (website) for illustration purposes.

Shopping websites. Practically, there are two extreme cases. (1) Users may flood the shopping websites with plenty of concurrent requests, thereby increasing traffic on the websites. (2) Users seldom send requests, then, the traffic on the shopping websites is low. In the first case, it is essential to dynamically create new nodes and efficiently balance the network resources to handle these requests, and in the second case, it is highly necessary to release the unused resources. As a result, a shopping website hosted on a private cloud computing platform would require an auto-scalable and fault-tolerant load balancing mechanism to handle concurrent users' requests. In this setting, task processing nodes are composed of virtual nodes available in the private cloud platform, among which the master nodes are managed through the PoW consensus. Concretely, PWB-LB is a suitable candidate to be used in such shopping websites.

The rest of this paper is structured as follows: the background and related work are described in Section 2, and preliminaries are presented in Section 3. Then Section 4 demonstrates a detailed workflow of the PWB-LB mechanism. In addition, Section 5 comprehensively analyzes the security and performance of the PWB-LB mechanism, and Section 6 experimentally validates the auto-scalability, efficiency, and performance of the PWB-LB mechanism. Finally, conclusion for the paper are summarized in Section 7.

2 Background and related work

Cloud computing [20] plays a huge role and contributes much to human life. For example, Google [21] provides Gmail [22], Google Earth [23], Google Analytics [24], and other services. High performance of the cloud computing is mostly needed when highly concurrent users' requests [25] become commonplace. Therefore, our overall goal is to build a load balancing mechanism, where computing nodes in a private cloud network participate in the operation of load balancing mechanism. Abstracting away the algorithm for this load balancing, it suffices to build a dynamic, auto-scalable, and fault-tolerant load balancing mechanism.

Load balancing methods provide high efficiency and security guarantees, allowing a hosted website to provide adequate service regardless of highly concurrent requests (high traffic) or node failure. Several studies have been performed for such methods, for different application use cases, offering varied performance tradeoffs, realizing automation of diverse degrees, and tolerating distinct forms of failures. We will discuss the five most widely used softwares for load balancing.

Comprised of scheduling, server cluster and shared storage layers, LVS [26] realizes a high-performance and available Linux server cluster by LVS's load balancing technology and Linux operating system, which has good reliability, scalability, operability, thereby achieving optimal performance at a low cost. However, it only deals with a few requests at one time. HAProxy provides a highly available application proxy based on TCP and HTTP, and is especially suitable for websites with a heavy load. Nginx [27] is the efficient HTTP, reverse proxy, and IMAP/POP3/SMTP (Internet mail access protocol/post office protocol-version 3/simple mail transfer protocol) services, which have good stability, rich feature set, sample configuration file, and low system resource consumption. Nginx implements dynamic and static page separations through scheduling rules. Moreover, it performs load balancing to back-end servers through different techniques such as polling, IP hash, and URL hash. Although Nginx and HAProxy perform gracefully under highly concurrent requests (high traffic), they have no auto-scalability ability once the number of requests reaches the maximum. Furthermore, there are two commonly used algorithms for load balancing. The Round Robin [28] is a simple scheduling algorithm that used the principle of time slices to assign each node a quantum for processing requests. However, the load is randomly selected, which makes a high traffic condition worse. Throttled [29] is an essential algorithm that fully adopted the use of virtual machine for allocating requests. The virtual machine is found through its index and scheduled based on the maintained states. The load balancing algorithm used in this study is a basic method. Different from all of these solutions, our design modified the load balancing algorithm to be based on the current load status. Thus, an auto-scalable network can be developed to guarantee the high-performance for both concurrent and one-way requests.

3 Preliminaries

This section explains the adopted cloud computing virtualization and PoW technologies.

3.1 Cloud computing virtualization

One of the core technologies in cloud computing is the virtualization technology. As depicted in Figure 2, a computer is virtualized into multiple logical computers for different operating systems, and their software applications can be used in parallel space without interfering with the process on another operating system, thus significantly improving the computer efficiency. VMWare, Xen, and kernel-based virtual machine (KVM) are the most commonly used virtual machine technologies, and clustering is one of the classical applications of these technologies. To achieve better performance, the existing load balancing mechanisms are designed based on the clustering architecture.

As a cloud computing management software, OpenStack supports different virtual machine technologies. OpenStack cloud platform consists of several components that have certain functions and coordinate with each other. Among them, Glance provides the image service. Specifically, it manages the virtual mirroring for creating, modifying, and deleting mirrors. All these processes draw support from the Glance application programming interface (API) and provide convenient mirroring scaling, which makes PWB-LB a perfect candidate for load balancing mechanism.

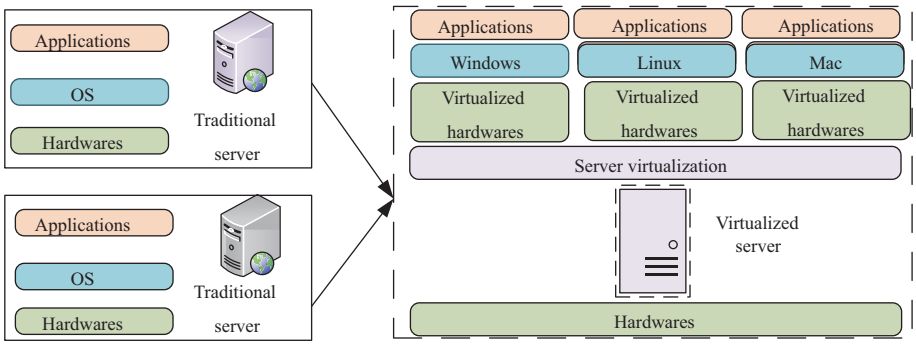

Figure 2 (Color online) Cloud computing virtualization technology.

3.2 PoW

PoW consensus usually works in a Blockchain system where a distributed ledger is maintained by all network nodes through the PoW consensus-based competitive computing. First, system nodes compete for bitcoins by solving a complex mathematical problem. The highest computing consumption in one way provides a standard for bitcoins, and in another way raises limits for malicious nodes to deviate from the rule. Second, network maintenance is done by the network nodes through consensus. We use an improved PoW in the PWB-LB as follows to ensure fault-tolerant and effective master nodes management.

(1) $\mathrm{Ph}(\mathrm{ct}_{i-1}) \rightarrow H_{i-1}$: determining hash of the previous epoch. Let Ver denote the version number of PoW and TS_i denote time stamp. Eq. (1) shows hash value H_{i-1} of the previous voting.

$$H_{i-1} = H(\mathrm{ct}_{i-1}) = H(\mathrm{Ver}, \mathrm{TS}_{i-1}, \mathrm{Diff}_{i-1}, \mathrm{Nonce}_{i-1}). \tag{1}$$

(2) $\mathrm{Ran}(\mathrm{RN}) \rightarrow \mathrm{Nonce}_i$: attempting nonce. As shown in Eq. (2), nodes competitively compute a legal random number Nonce_i so that the SHA-256 algorithm-hashed value of current voting meets the requirement of target TGT_i, and Nonce_i is less than the specific difficulty Diff_i.

$$\mathrm{SHA256}(\mathrm{SHA256}(\mathrm{Ver} + H_{i-1} + \mathrm{TS}_i + \mathrm{Diff}_i + \mathrm{Nonce}_i)) \leqslant \mathrm{TGT}_i. \tag{2}$$

(3) $\mathrm{Val}(H_i, \mathrm{Nonce}_i) \rightarrow N^m$: validating nonce. As Eq. (3), master nodes will be generated and broadcasted to the private cloud computing platform for verification, here are a hand of simple nonce verifications and sortings in a private cloud platform.

$$(H_i \leqslant \mathrm{TGT}_i) \cap (\mathrm{Nonce}_i \leqslant \mathrm{Diff}_i) \longrightarrow \mathrm{Gen}(N^m). \tag{3}$$

(4) $\mathrm{Re}(N^m)_i \rightarrow (N^m)_{i+1}$: performing the next turn of voting. If all masters either crash or kicked for malware threat, computing nodes restarts a new epoch of masters voting.

4 PWB-LB: load balancing mechanism construction

In this section we present PWB-LB, the auto-scalable and fault-tolerant load balancing mechanism.

We first define the system model involving a private cloud platform of N computing nodes to work in the load balancing mechanism, vote for master nodes and manage them. Nodes in the deployed cluster work in the load balancing assembly to dynamically give respond to users' requests. Any auto-scalable execution happens in time on the load balancing network to reach the most reasonable resources management and load balancing. Our model particularly matches the private cloud environment, and perfectly fits the hosted shopping websites. Figure 3 shows the system model in terms of its basic entities.

• Users. Users send requests for specific tasks.

• Private cloud platform. The private cloud platform schedules a load balancing mechanism for handling requests.

Our design on load balancing mechanism is with regard to the auto-scalable and fault-tolerant tasks processing system.

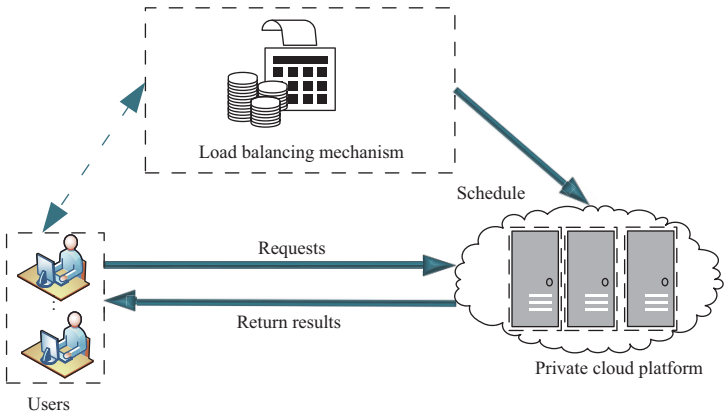

Figure 3 (Color online) System model.

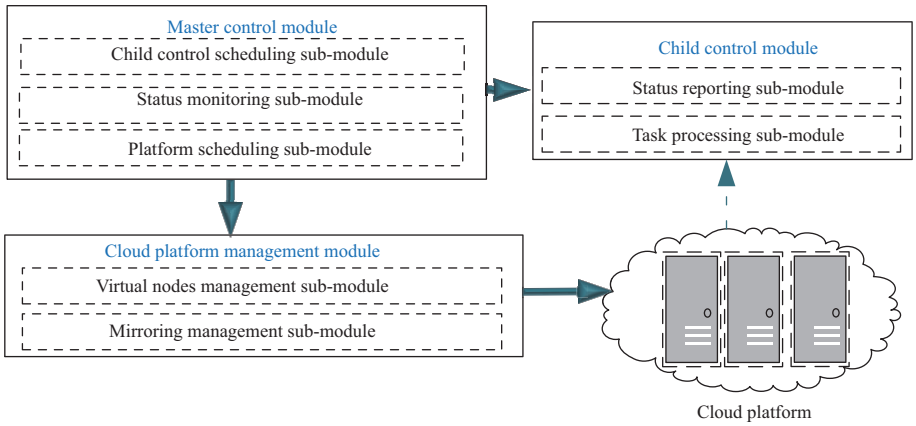

Figure 4 (Color online) The modular load balancing assembly.

4.1 Mechanism overview

The PWB-LB mechanism proceeds in epochs: at beginning computing nodes vote for master nodes based on PoW consensus. Once the current master node crashes, others in the masters' queue operate in turns to receive requests, schedule child nodes and manage cloud computing resources. After each epoch ends, the PWB-LB restarts with the new masters' voting.

Within the PWB-LB, an assembly operates as a mainstream of load balancing, which is modularly designed to reach the aforementioned auto-scalability, and contains three modules as depicted in Figure 4.

• Master control module. It receives requests from users and dispatches them to virtual nodes in child control module according to our load balancing policy, monitors the status of child control, and dynamically manages the virtual nodes through the cloud platform management module and based on the current load.

• Child control module. It processes requests from the master node and timely reports its status.

• Cloud platform management module. It carries out the dynamic management on virtual computing resources and mirrorings in the private cloud platform.

A simple PoW consensus operates several steps for the master nodes management within the PWB-LB.

• Master nodes voting. N performs a simple PoW consensus to compete for the master nodes. Assume that node N_j $(j = 1, \ldots, N)$ acquires its random number Nonce_i^j at time T_i^j, then, N_j acts as the current master node N^m if Eq. (4) is met:

$$\left(H(\mathrm{Nonce}_i^j) \leqslant \mathrm{TGT}_i\right) \cap (\mathrm{Nonce}_i^j \leqslant \mathrm{Diff}_i) \cap (T_i^j = \mathrm{Min}\{T_i^1, \ldots, T_i^N\}). \tag{4}$$

Other master nodes N_c^k $(k = 2, \ldots, K)$ in the queue (except for $N_c^1 = N^m$) are determined as the same way:

$$\left(H(\mathrm{Nonce}_i^{N_c^k}) \leqslant \mathrm{TGT}_i\right) \cap \left(\mathrm{Nonce}_i^{N_c^k} \leqslant \mathrm{Diff}_i\right) \cap \left(T_i^{N_c^1} < T_i^{N_c^2} < \cdots < T_i^{N_c^K} = \mathrm{Min}\{T_i^1, \ldots, T_i^N\}\right). \tag{5}$$

• Agreement on the masters queue. $\text{Nonce}_i^{N_c^k}$ $(k = 1, \ldots, K)$ are broadcasted within the private cloud network for verification. If more than $|N|/2$ of cloud computing nodes catch on an agreement, then $\text{Nonce}_i^{N_c^k}$ will be legal and sorted as the queue of $N_c^1, \ldots, N_c^K$.

• Master nodes state confirmation. N^m is voted through the private cloud network for running state checking. Let $\text{Num}_{\text{vote}}^{N^m}$ denote the number of crashed votings, then N^m will be reset as Eq. (6).

$$\left(\text{Num}_{\text{vote}}^{N^m} \geqslant \frac{1}{2}|N|\right) \rightarrow (N_c^k = N^m = \text{Down}) \cap (N^m = N_c^{k+1}). \tag{6}$$

Before presenting our mechanism formally, we remark on the style of our presentation. We first modularly define the proposed load balancing assembly by running their sub-modules, then build the detailed construction of the PWB-LB mechanism.

4.2 Load balancing assembly

The load balancing assembly works through three modules: the master control, child control, and cloud platform management modules.

4.2.1 $\text{mc}(\text{mc}_{\text{ccs}}, \text{mc}_{\text{sm}}, \text{mc}_{\text{ps}})$

The master control module mc embodies the child control scheduling sub-module mc_{ccs}, the status monitoring sub-module mc_{sm}, and the platform scheduling sub-module mc_{ps}.

• mc_{ccs}. The child control scheduling sub-module has three functions. (1) It dynamically schedules cc to process requests R. (2) It performs load balancing processing based on mc_{sm}-obtained task processing and service running states of child nodes N^{cc}. Specifically, it culls child with crashed running states from the child queue Q_{cc}, and selects the most lightly-loaded and top-located one in Q_{cc} to process R. (3) Besides, it adds new nodes $N_{\text{new}}^{\text{cc}}$ to Q_{cc}.

• mc_{sm}. The status monitoring sub-module focuses on service running and task execution states reported by cc_{sr}. (1) A child is considered out of service if cc_{sr} has not reported its state for a time period. (2) An idle threshold Thr_{id} and overload Thr_{ol} on the system tasks are set. The largest value is set as Thr_{ol} to show some nodes are processing up to Thr_{ol} tasks and the system is overloaded. The Thr_{id} of the smallest task load triggers a timing to count the duration of this condition, and the system is idle when this result is larger than T_{tg}.

• mc_{ps}. The platform scheduling sub-module dynamically scales computing nodes by entrusting cpm to open or close the virtual nodes. It detects the system tasks load based on mc_{sm}, and schedules cpm to close idle nodes $N_{\text{idle}}^{\text{cc}}$ when the system is idle and start new nodes $N_{\text{new}}^{\text{cc}}$ when overloaded.

Moreover, if any sub-module of mc goes wrong it will be checked through the cloud platform and re-set.

4.2.2 $\text{cc}(\text{cc}_{\text{sr}}, \text{cc}_{\text{tp}})$

The child control module cc embodies the sub-modules of status reporting cc_{sr} and task processing cc_{tp}.

• cc_{sr}. The status reporting sub-module reports the child state to mc_{sm}. The task execution state indicates the load of current tasks. Besides, the service running state can be referred to by the reported conditions.

• cc_{tp}. The task processing sub-module is scheduled by mc_{ccs} to process R. It starts new threads and calls the corresponding service applications to handle the tasks.

4.2.3 $\text{cpm}(\text{cpm}_{\text{vnm}}, \text{cpm}_{\text{mm}})$

The cloud platform management module cpm embodies the virtual nodes management sub-module cpm_{vnm} and the mirroring management sub-module cpm_{mm}.

• cpm_{vnm}. The virtual nodes management sub-module manages virtual resources in the private cloud platform according to uniform resource locator (URL)-formed scheduling results from mc_{ps}. It acquires the resource scheduling contents in this URL request (e.g., parameters of new nodes). Then, it calls API to open (the system is overloaded) or close (the system is idle) virtual nodes.

• cpm_{mm}. The mirroring management sub-module manages cloud platform mirroring to quickly deploy the system. It deploys mc and makes it into a master mirroring M_{mc} that can open a master node, deploys cc and its business applications cc_{ba} to make a child mirroring M_{cc}, schedules API to store and provides mirrorings information for new nodes.

4.3 Constructing the PWB-LB mechanism

The PWB-LB consists of the master nodes voting, child control scheduling and platform scheduling.

4.3.1 *Master node voting*

• $\mathrm{Ph}(\mathrm{ct}_{i-1}) \rightarrow H_{i-1}$. Take the parameters $\mathrm{Ver}, \mathrm{TS}_{i-1}, \mathrm{Diff}_{i-1}, \mathrm{Nonce}_{i-1}$ of $(i-1)$-th epoch and hash value H_{i-2} of $(i-2)$-th epoch as inputs, and H_{i-1} can be calculated through

$$H_{i-1} = H(\mathrm{Ver}, H_{i-2}, \mathrm{TS}_{i-1}, \mathrm{Diff}_{i-1}, \mathrm{Nonce}_{i-1}). \tag{7}$$

• $\mathrm{Ran}(\mathrm{RN}) \rightarrow \mathrm{Nonce}_i$. At epoch beginning, all nodes compete for N^m by taking H_{i-1} and trying a legal Nonce_i meeting Eq. (2), here setting Diff_i quite smaller for a rapid solution attempt.

• $\mathrm{Val}(H_i, \mathrm{Nonce}_i) \rightarrow N^m$. Nodes in the private cloud simply bring in the upcoming Nonce_i^j and calculate its hash. If all situations in Eqs. (4) and (8) are met, then $N_j = N_c^1 = N^m$, where $\mathrm{Num}_v^{N_j}$ denote the agreement votings.

$$\mathrm{Num}_v^{N_j} > \frac{1}{2}|N|. \tag{8}$$

If there exists N_c^k $(k = 2, \ldots, K)$ in $N - N_c^1$ that fulfill the conditions in Eq. (9), then they join the masters' queue by backuping state of their previous one N_c^k $(k = 1, \ldots, K-1)$ in terms of R, child control scheduling, state monitoring and platform management, and timely succeeding the currently invalid master.

$$H\left(\mathrm{Nonce}_i^{N_c^k}\right) \leqslant \mathrm{TGT}_i,\ \mathrm{Nonce}_i^{N_c^k} \leqslant \mathrm{Diff}_i,\ \mathrm{Num}_v^{N_c^k} > \frac{1}{2}|N|. \tag{9}$$

4.3.2 *Child control scheduling*

The child control scheduling process includes several steps.

Step1. $\mathrm{Sta}(\mathrm{Sys}) \rightarrow (\mathrm{mc}, \mathrm{cc}, \mathrm{cpm})$: start the system.

• The PWB-LB-based system S_{lb} deploys mc on N_c^1 and configures it into a master mirroring by $\mathrm{cpm}_{\mathrm{mm}}$, at this point N_c^1 is opened and used as N^m in the assembly, and $\mathrm{mc}_{\mathrm{sm}}$ starts to monitor the child state.

• cc and $\mathrm{cc}_{\mathrm{ba}}$ are then deployed and configured into child mirrorings to initiate several virtual nodes as N^{cc}, at this point, cc could continuously report the service running and tasks execution status to $\mathrm{mc}_{\mathrm{sm}}$.

• For the deployed assembly, mc receives requests from users U, updates states of cc, schedules cc for tasks processing, and adds $N_{\mathrm{new}}^{\mathrm{cc}}$ to the child queue.

Furthermore, the voting process Vot(mc) will be performed if mc goes wrong (e.g., the load is not reasonably balanced, R is not received for a long time, crushed cc is not culled, or idle cc is not removed), otherwise, performing the process $\mathrm{Pro}(R)$.

Step2. $\mathrm{Vot}(\mathrm{mc}) \rightarrow N_c^k / N_c^{k+1}$: check the current master's state.

• Once mc goes wrong in any part, the validation message will be sent through the private cloud to verify the current master node.

• N_c^k will be considered down if Eq. (10) is met, and the next redundant master is activated to work.

$$\mathrm{Num}_{\mathrm{vote}}^{N_c^k} \geqslant \frac{1}{2}|N|. \tag{10}$$

Step3. $\mathrm{Pro}(R) \rightarrow$ results: process the requests.

• $\mathrm{mc}_{\mathrm{sm}}$ analyzes the states of cc and kicks all abnormally operated nodes, furthermore, it chooses the lightest loaded and topmost nodes in Q_{cc} to handle R.

• The scheduled $\mathrm{cc}_{\mathrm{tp}}$ then processes tasks by its business applications.

4.3.3 *Platform scheduling*

The platform scheduling process includes the following steps.

Step1. $\mathrm{Anl}(\mathrm{Sts}) \rightarrow (\mathrm{idle}, \mathrm{ol})$: analyze the system load status.

• Let $\mathrm{Num}_{\mathrm{cc}}^t$ denote the tasks loads of some child and $T_{\mathrm{Thr}_{\mathrm{id}}}$ be the duration time of situation $\mathrm{Num}_{\mathrm{cc}}^t \leqslant \mathrm{Thr}_{\mathrm{id}}$. The system is idle of load and performs the process Pro(Idle) when situations meet Eq. (11). Otherwise, the system is overloaded and initiates the process Pro(Ol) when Eq. (12) is met.

$$\mathrm{Num}_{\mathrm{cc}}^t \leqslant \mathrm{Thr}_{\mathrm{id}},\quad T_{\mathrm{Thr}_{\mathrm{id}}} \geqslant T_{\mathrm{tg}}, \tag{11}$$

$$\mathrm{Num}_{\mathrm{cc}}^{t} \geqslant \mathrm{Thr}_{\mathrm{ol}}. \tag{12}$$

Step2. Pro(Idle) $\rightarrow$ (close N_{idle}): process the idle state.

• Nodes with the smallest load are selected from Q_{cc} as the idle $N_{\mathrm{idle}}^{\mathrm{cc}}$, and no new tasks will be assigned until their loaded tasks are completed.

• $\mathrm{mc}_{\mathrm{ps}}$ then initiates a URL request to $\mathrm{cpm}_{\mathrm{vnm}}$ to indicate the IP information of $N_{\mathrm{idle}}^{\mathrm{cc}}$.

• $\mathrm{cpm}_{\mathrm{vnm}}$ parses the URL request, and invokes API to close the corresponding virtual nodes.

Step3. Pro(Ol) $\rightarrow$ (open N_{new}): process the overload state.

• $\mathrm{mc}_{\mathrm{ccs}}$ selects the mirrorings of new nodes $N_{\mathrm{new}}^{\mathrm{cc}}$, sets resource configurations such as the virtual core number, disk space and memory size, and packs a mirroring request to $\mathrm{cpm}_{\mathrm{vnm}}$.

• $\mathrm{cpm}_{\mathrm{vnm}}$ parses the request, passes the related parameters to API, and starts $N_{\mathrm{new}}^{\mathrm{cc}}$ by their mirrorings.

• After $N_{\mathrm{new}}^{\mathrm{cc}}$ is started, $\mathrm{cc}_{\mathrm{new}}$ reports the state information to $\mathrm{mc}_{\mathrm{sm}}$ and $\mathrm{mc}_{\mathrm{ccs}}$ adds $\mathrm{cc}_{\mathrm{new}}$ to Q_{cc}.

So far, the entire epoch of the PWB-LB mechanism is finished. Section 5 presents the system analysis.

5 Analysis

In this section, we perform the security and performance analysis of PWB-LB.

5.1 Security

The main security requirements of our PWB-LB include the securities of the master node confirmation, the load distribution collection, and the load transfer decision.

• Master node confirmation. PoW is used for masters' voting during this process, therefore, the Sybil attack [30] is the main consideration. In the Sybil attacks, malicious nodes with different identities are the Sybils of some underground big boss.

• Load distribution collection. DDoS attacks [31] against the master and heavy-loaded nodes are usually initiated to destroy the load distribution information.

• Load transfer decision. Attackers may control the master node or report false load information to confuse the load transfer decision.

Proposition 1. The PWB-LB is secure from the Sybil attacks, and N^{m} is reliable.

Proof. A malicious master node N^{mm} in the Sybil attack must execute malicious actions. While in our PWB-LB, the process Vot(mc) will be performed for unregularly behaved N^{mm}. Therefore, N^{mm} will be finally re-set.

Proposition 2. The PWB-LB is secure from DDoS attacks against the master and heavy-loaded nodes, and the load distribution information is reliable.

Proof. It includes two conditions to explain.

• If the master node is attacked, then it is the same case as Proposition 1.

• If a heavy-loaded node in N^{cc} is attacked, then, its service running status is down and it will be culled from Q_{cc}.

In conclusion, the load distribution information is reliable.

Proposition 3. In the PWB-LB, the master node is reliable, and the load transfer decision can be reasonably made.

Proof. It includes two conditions to explain.

• If the master node makes false decisions, then it is the same case as Propositions 1.

• If a malicious child falsely reports its load information to mislead the master, there will be two cases. When it reports that the state is overload, new nodes will be opened for load sharing. When it reports the state that the state is idle, idle nodes will be closed. Hence, this behavior has no influence to the system.

In conclusion, the load transfer decision can be reasonably made.

5.2 Performance

Performance of the PWB-LB is commonly determined by three essential properties: the system load, availability [32], and performability [33], which are explicitly analyzed in the subsequent parts.

5.2.1 *Load*

Definition 1 (load)**.** Let s denote the system state, and t_s be the task. Load is the mean number of tasks $\bar{\mathrm{Num}}_{t_s}^{N_j}$ waiting to be processed in an on-demand computing environment.

Theorem 1 (Little's law)**.** In a stable system, the long-term average number of customers $\bar{\mathrm{Num}}_c$ is equal to the long-term effective arrival rate λ, multiplied by the average waiting time $\bar{T}_w$ of the customer, and it can be expressed by

$$\bar{\mathrm{Num}}_c = \lambda \cdot \bar{T}_w. \tag{13}$$

Proposition 4. The PWB-LB employs the function of load balancing and usually sustains a small level of tasks' load which is up to $\mathrm{Thr}_{\mathrm{ol}}$. Besides, the values of $\mathrm{Thr}_{\mathrm{idle}}$ and $\mathrm{Thr}_{\mathrm{ol}}$ can be dynamically adapted to realize a fascinating load.

Proof. Let T_{t_s} denote the time costs for processing an individual task, λ be the tasks arrival rate, and p presents the task processing rate of a virtual node. There is

$$p = \frac{1}{T_{t_s}}. \tag{14}$$

Then, the average waiting time $\bar{T}_w$ of t_s satisfies

$$\bar{T}_w = \frac{1}{p-\lambda} - \frac{1}{p} = \frac{1}{\frac{1}{T_{t_s}} - \lambda} - \frac{1}{\frac{1}{T_{t_s}}} = \frac{T_{t_s}}{1 - T_{t_s} \cdot \lambda} - T_{t_s}. \tag{15}$$

By Theorem 1, the average waiting number $\bar{\mathrm{Num}}_{t_s}^{N_j}$ of tasks meets:

$$\bar{\mathrm{Num}}_{t_s}^{N_j} = \lambda \cdot \bar{T}_w = \lambda \cdot \frac{T_{t_s}}{1 - T_{t_s} \cdot \lambda} - \lambda \cdot T_{t_s}. \tag{16}$$

We then separately analyze the values of $\bar{\mathrm{Num}}_{t_s}^{N_j}$ under the system states of idle, overloaded and stable.

• Under the condition of $s =$ idle:

Let Num_{t_s} denote the total number of tasks, and the tasks need to be completed within given time ΔT_{t_s}. Assume the system is idle when the actual handling time is controled less half as $\frac{1}{2}\Delta T_{t_s}$, then Eq. (17) is met under this idle case:

$$0 \leqslant \frac{|\mathrm{Num}_{t_s} \cdot T_{t_s}|}{|N^{\mathrm{cc}}|} \leqslant \frac{1}{2}\Delta T_{t_s}. \tag{17}$$

$\mathrm{Thr}_{\mathrm{idle}}$ in this case can be obtained as follows:

$$\mathrm{Thr}_{\mathrm{idle}} = \frac{|\mathrm{Num}_{t_s}|}{|N^{\mathrm{cc}}|} = \frac{\Delta T_{t_s}}{2T_{t_s}}, \quad T_{t_s} = \frac{\Delta T_{t_s}}{2\mathrm{Thr}_{\mathrm{idle}}}, \quad \bar{\mathrm{Num}}_{t_s}^{N_j} \leqslant \mathrm{Thr}_{\mathrm{idle}}. \tag{18}$$

From Eqs. (16) and (18), there is

$$\begin{aligned}\bar{\mathrm{Num}}_{t_s}^{N_j} &= \lambda \cdot \frac{\frac{\Delta T_{t_s}}{2\mathrm{Thr}_{\mathrm{idle}}}}{1 - \frac{\Delta T_{t_s}}{2\mathrm{Thr}_{\mathrm{idle}}} \cdot \lambda} - \lambda \cdot \frac{\Delta T_{t_s}}{2\mathrm{Thr}_{\mathrm{idle}}} = \frac{\lambda \cdot \Delta T_{t_s}^2}{4\mathrm{Thr}_{\mathrm{idle}}(\mathrm{Thr}_{\mathrm{idle}} - \lambda \cdot \Delta T_{t_s})} \\ &= \frac{\lambda \cdot \Delta T_{t_s}^2}{4(\mathrm{Thr}_{\mathrm{idle}} - \frac{1}{2}\lambda \cdot \Delta T_{t_s})^2 - \lambda^2 \cdot \Delta T_{t_s}^2}.\end{aligned} \tag{19}$$

• Under the condition of $s =$ overloaded:

Assume that $\varepsilon \to 0$ ($\varepsilon > 0$) and the system is overloaded when the actual handling time is more than $\Delta T_{t_s} - \varepsilon$, then Eq. (20) is met under this overload case:

$$\frac{|\mathrm{Num}_{t_s} \cdot T_{t_s}|}{|N^{\mathrm{cc}}|} \geqslant \Delta T_{t_s} - \varepsilon. \tag{20}$$

$\mathrm{Thr}_{\mathrm{ol}}$ in this case can be obtained as follows:

$$\mathrm{Thr}_{\mathrm{ol}} = \frac{|\mathrm{Num}_{t_s}|}{|N^{\mathrm{cc}}|} = \frac{(\Delta T_{t_s} - \varepsilon)}{T_{t_s}}, \quad T_{t_s} = \frac{(\Delta T_{t_s} - \varepsilon)}{\mathrm{Thr}_{\mathrm{ol}}}, \quad \bar{\mathrm{Num}}_{t_s}^{N_j} \leqslant \mathrm{Thr}_{\mathrm{ol}}. \tag{21}$$

From Eqs. (16) and (21), there is

$$\begin{aligned}\bar{\mathrm{Num}}_{t_s}^{N_j} &= \lambda \cdot \frac{\frac{(\Delta T_{t_s}-\varepsilon)}{\mathrm{Thr_{ol}}}}{1-\frac{(\Delta T_{t_s}-\varepsilon)}{\mathrm{Thr_{ol}}}\cdot\lambda} - \lambda \cdot \frac{(\Delta T_{t_s}-\varepsilon)}{\mathrm{Thr_{ol}}} = \frac{\lambda(\Delta T_{t_s}-\varepsilon)(\mathrm{Thr_{ol}}-1)}{\mathrm{Thr_{ol}^2}-\mathrm{Thr_{ol}}\cdot\lambda(\Delta T_{t_s}-\varepsilon)} \\ &= \frac{\lambda(\Delta T_{t_s}-\varepsilon)(\mathrm{Thr_{ol}}-1)}{[\mathrm{Thr_{ol}}-\frac{1}{2}\lambda(\Delta T_{t_s}-\varepsilon)]^2-\frac{1}{4}\lambda^2(\Delta T_{t_s}-\varepsilon)^2}. \end{aligned} \tag{22}$$

Based on our load balancing strategy, new computing nodes will be opened to solve the overload problem. We assume that $|N^{\mathrm{cc}}|$ number of computing nodes are started (it could be adapted according to the physical truth), then there is

$$|N_{\mathrm{new}}^{\mathrm{cc}}| = |N^{\mathrm{cc}}|, \quad \bar{\mathrm{Num}}_{t_s}^{N_j} \leqslant \frac{1}{2}\mathrm{Thr_{ol}}. \tag{23}$$

• Under the condition of $s =$ stable:

From the idle and overloaded conditions, the system is stable when Eq. (24) is met:

$$\frac{1}{2}\Delta T_{t_s} < \frac{|\mathrm{Num}_{t_s}\cdot T_{t_s}|}{|N^{\mathrm{cc}}|} < \Delta T_{t_s}-\varepsilon. \tag{24}$$

Let $\mathrm{Thr_{sb}} = \frac{|\mathrm{Num}_{t_s}|}{|N^{\mathrm{cc}}|}$, then there are the following results,

$$\frac{\Delta T_{t_s}}{2T_{t_s}} < \mathrm{Thr_{sb}} < \frac{\Delta T_{t_s}-\varepsilon}{T_{t_s}}, \quad \frac{\Delta T_{t_s}}{2\mathrm{Thr_{sb}}} < T_{t_s} < \frac{\Delta T_{t_s}-\varepsilon}{\mathrm{Thr_{sb}}}, \quad \mathrm{Thr_{idle}} < \bar{\mathrm{Num}}_{t_s}^{N_j} < \mathrm{Thr_{ol}}. \tag{25}$$

From Eqs. (16) and (25), there is

$$\bar{\mathrm{Num}}_{t_s}^{N_j} = \lambda \cdot \frac{T_{t_s}}{1-T_{t_s}\cdot\lambda} - \lambda \cdot T_{t_s}. \tag{26}$$

The aforementioned analysis makes a common way to judge the system load, and the results show that a small load is kept in our method-aided cloud platform. To have an intuitive evaluation of the load distributions, we raise a simple example. We simply defines the tasks arrival rate as $\lambda = 100$/s and the average processing time as $T_{t_s} = 0.004$ s. If the most $\Delta T_{t_s} = 2$ s gives to a platform for any task responding, the determined idle $\mathrm{Thr_{idle}} = 300$ and overload $\mathrm{Thr_{ol}} = 500$ can provide the system of reasonable resources scaling. We bring these inputs to determine the load circumstances under different system statuses.

When the system is idle, there is: $\bar{\mathrm{Num}}_{t_s}^{N_j} = \frac{100\cdot 4}{4\cdot 300(300-100\cdot 2)} = 0.003$.

When the system is overload, there is: $\bar{\mathrm{Num}}_{t_s}^{N_j} = \frac{100(2-\varepsilon)(500-1)}{500^2-500\cdot 100(2-\varepsilon)} \approx 0.66$.

When the system is stable, there is: $\bar{\mathrm{Num}}_{t_s}^{N_j} = 100 \cdot \frac{0.004}{1-0.004\cdot 100} - 100\cdot 0.004 = 0.26$.

Thus through the above results, we have an intuitive recognization of the system loads. We see that the load under idle state is the smallest and that under the overload state is the biggest. The common load under a stable state usually keeps a slower level than the average condition. Therefore, the cloud platform using our load balancing mechanism is usually burdened with light load.

5.2.2 *System availability*

Definition 2 (Availability)**.** Availability is the probability that a required minimum fraction of tasks are finished within a given deadline f. Let ϕ denote the given service quality requirement, $F(T)$ be the response time for a task t_{new} at the time T, then the system is available if Eq. (27) is met:

$$\Pr(F(T) \leqslant f) \leqslant \phi. \tag{27}$$

The available probability av_f^T can be expressed as

$$\mathrm{av}_f^T = \Pr(F(T) \leqslant f). \tag{28}$$

Proposition 5. The PWB-LB employs high availability $\mathrm{av}_{f|s}^{T}$ which is at least equal to $1-\mathrm{e}^{-\frac{4}{T_{t_s}\cdot\Delta T_{t_s}^2}f}$.

Proof. The event "t_s arrives" meets the Poisson distribution as shown in Eq. (29), and the required time for processing tasks meets the exponential distribution.

$$P(x=j)=\frac{(pf)^j}{j!}\mathrm{e}^{-pf}. \tag{29}$$

We then separately analyze the values of $\mathrm{av}_{f|s}^{T}$ under the system states of idle, overloaded and stable.

• Under the condition of $s=$ idle, the number of available virtual nodes $\mathrm{Num}_{\mathrm{av}}^{N^{\mathrm{cc}}}$ meets:

$$\mathrm{Num}_{\mathrm{av}}^{N^{\mathrm{cc}}}=|N^{\mathrm{cc}}|. \tag{30}$$

From Eq. (28) and Exponential distribution of the response time, there is

$$\mathrm{av}_{f|s}^{T}=1-\mathrm{e}^{-pf}, \tag{31}$$

$$p\geqslant\frac{\frac{\mathrm{Num}_{t_s}}{|N^{\mathrm{cc}}|}}{\frac{1}{2}\Delta T_{t_s}}=\frac{2\mathrm{Num}_{t_s}}{|N^{\mathrm{cc}}|\cdot\Delta T_{t_s}}=\frac{2|N^{\mathrm{cc}}|\cdot\mathrm{Thr}_{\mathrm{idle}}}{|N^{\mathrm{cc}}|\cdot\Delta T_{t_s}}=\frac{2\frac{2}{T_{t_s}\cdot\Delta T_{t_s}}}{\Delta T_{t_s}}=\frac{4}{T_{t_s}\cdot\Delta T_{t_s}^2}. \tag{32}$$

• Under the condition of $s=$ overloaded, $\mathrm{Num}_{\mathrm{av}}^{N^{\mathrm{cc}}}$ meets:

$$\mathrm{Num}_{\mathrm{av}}^{N^{\mathrm{cc}}}=0. \tag{33}$$

Let $\otimes$ denote the convolution operation, from Eq. (28) and Exponential distribution of the response time, there is

$$\mathrm{av}_{f|s}^{T}=(1-\mathrm{e}^{-pf})\otimes\left(1-\sum_{j=0}^{|N^{\mathrm{cc}}|-j}\frac{(|N^{\mathrm{cc}}|pf)^j}{j!}\cdot\mathrm{e}^{-|N^{\mathrm{cc}}|pf}\right). \tag{34}$$

As the overload condition in Proposition 4, we add $|N^{\mathrm{cc}}|$ number of new nodes to the child queue. Consider the worst condition and all $|N^{\mathrm{cc}}|$ nodes are applied, there is,

$$\mathrm{av}_{f|s}^{T}=1-\mathrm{e}^{-pf}, \tag{35}$$

$$p\geqslant\frac{\frac{\mathrm{Num}_{t_s}}{|N^{\mathrm{cc}}|}}{\Delta T_{t_s}-\varepsilon}=\frac{\mathrm{Num}_{t_s}}{|N^{\mathrm{cc}}|\cdot(\Delta T_{t_s}-\varepsilon)}=\frac{|N^{\mathrm{cc}}|\cdot\mathrm{Thr}_{\mathrm{ol}}}{|N^{\mathrm{cc}}|\cdot(\Delta T_{t_s}-\varepsilon)}=2\frac{\mathrm{Thr}_{\mathrm{ol}}}{(\Delta T_{t_s}-\varepsilon)}=\frac{2}{T_{t_s}(\Delta T_{t_s}-\varepsilon)^2}. \tag{36}$$

• Under the condition of $s=$ stable, $\mathrm{Num}_{\mathrm{av}}^{N^{\mathrm{cc}}}$ meets:

$$\mathrm{Num}_{\mathrm{av}}^{N^{\mathrm{cc}}}=|N^{\mathrm{cc}}|. \tag{37}$$

From Eq. (28) and Exponential distribution of the response time, there is

$$\mathrm{av}_{f|s}^{T}=1-\mathrm{e}^{-pf}, \tag{38}$$

$$p\geqslant\frac{\frac{\mathrm{Num}_{t_s}}{|N^{\mathrm{cc}}|}}{\Delta T_{t_s}}=\frac{\mathrm{Num}_{t_s}}{|N^{\mathrm{cc}}|\cdot\Delta T_{t_s}}>\frac{|N^{\mathrm{cc}}|\cdot\mathrm{Thr}_{\mathrm{idle}}}{|N^{\mathrm{cc}}|\cdot\Delta T_{t_s}}=\frac{\frac{2}{T_{t_s}\cdot\Delta T_{t_s}}}{\Delta T_{t_s}}=\frac{2}{T_{t_s}\cdot\Delta T_{t_s}^2}. \tag{39}$$

In conclusion from the above three conditions, there is

$$\mathrm{Max}\{p\}=\mathrm{Max}\left\{\frac{4}{T_{t_s}\cdot\Delta T_{t_s}^2},\frac{2}{T_{t_s}(\Delta T_{t_s}-\varepsilon)^2},\frac{2}{T_{t_s}\cdot\Delta T_{t_s}^2}\right\}=\frac{4}{T_{t_s}\cdot\Delta T_{t_s}^2}. \tag{40}$$

Thus, the minimum value of $\mathrm{av}_{f|s}^{T}$ meets

$$\mathrm{Min}\{\mathrm{av}_{f|s}^{T}\}=1-\mathrm{e}^{-pf}=1-\mathrm{e}^{-\frac{4}{T_{t_s}\cdot\Delta T_{t_s}^2}f}. \tag{41}$$

5.2.3 *System performability*

Definition 3 (Performability)**.** Performability refers to the probability $P(t_s)$ that t_s gets successfully completed within limited time ΔT_{t_s}. Let $P(t_s^l)$ denote successful probability of the l-th task t_s^l, then

$$P(t_s) = \Pi_{l=1}^{L} P(t_s^l). \tag{42}$$

Proposition 6. The PWB-LB employs high performability $P(t_s)$ up to 1.

Proof. Let $\mathrm{Num}_{t_s}^{\mathrm{ms}}$ denote the number of unfinished tasks within ΔT_{t_s}. Then, there is

$$P(t_s) = \frac{\mathrm{Num}_{t_s}^{\mathrm{ms}}}{\mathrm{Num}_{t_s}} \cdot 100\%. \tag{43}$$

For $s =$ idle/stable, no tasks missed their deadlines, then $P(t_{\mathrm{idle}}) \approx P(t_{\mathrm{sb}}) \approx 1$. For $s =$ overloaded, $N_{\mathrm{new}}^{\mathrm{cc}}$ is opened, then, there is

$$P(t_{\mathrm{idle}}) \approx P(t_{\mathrm{overload}}) \approx P(t_{\mathrm{stable}}) \approx 1. \tag{44}$$

6 Experimental evaluation

In this section, we perform several experiments and performance evaluation of the PWB-LB mechanism. Unless stated otherwise, results reported in this section are by design for the case where the master node may crash.

First, we demonstrate that PWB-LB is indeed auto-scalable by experimenting with different dynamic tasks. Besides, it is evident that even under dynamic conditions, PWB-LB reaches high efficiency of hundreds of tasks per second. Furthermore, compared to the existing mechanisms, PWB-LB outperforms them in terms of more auto-scalability and efficiency. Finally, we demonstrate PWB-LB performance over massive requests, i.e., high traffic.

Implementation details. We developed a prototype of PWB-LB using JAVA programming language, on the OpenStack cloud computing platform for task processing. Specifically, there we assume that it takes 30–50 ms for a virtual node to process a difficult task, besides, the idle threshold is 20 tasks and the overload threshold is 50 tasks. The complete evaluation results are plotted in Figure 5.

6.1 Auto-scalability

We analyze the auto-scalability of the PWB-LB under dynamic environments where we set the number of tasks ranging between 1000 and 10000. The auto-scalability is reflected by timely and dynamically active or passive nodes, hence the maximum number of child nodes at a time can act as a measurement metric for auto-scalability. Besides, time costs under this circumstance are evaluated to validate the mechanism's efficiency. The purpose of this experiment is to compare PWB-LB to other existing methods namely ordinary cluster, classical Round Robin, and Throttled previously describe in Section 2.

• Auto-scalability. The PWB-LB allows auto-scalability by dynamically adding and removing computing nodes through the load balancing mechanism based on system load status. As shown in Figure 5(a), the higher the number of tasks, the more child nodes instantiated, and vice versa, thereby ensuring the prudent management of virtual resources. For example, the peak number of child nodes was 13 for 5000 tasks, and 21 for 10000 tasks. The continuous curve change indicates the rapid nodes instantiation or removal. While from the comparison results, we see that the PWB-LB effectively performs automatic scaling than the existing methods, where the Round Robin and Throttled algorithm left created nodes active most of the time.

• Efficiency. The efficiency of PWB-LB is measured by the time costs for processing tasks. As shown in Figure 5(b), it takes less time for the PWB-LB to handle large-scale tasks than the ordinary cluster, while the Round Robin and Throttled take much more time. For example, it takes 22.07 s to process 5000 tasks and 32.85 s for 10000 tasks using our PWB-LB, but it takes almost 400 s using Round Robin to process 10000 tasks.

In conclusion, the PWB-LB mechanism allows for effective auto-scalability and high efficiency.

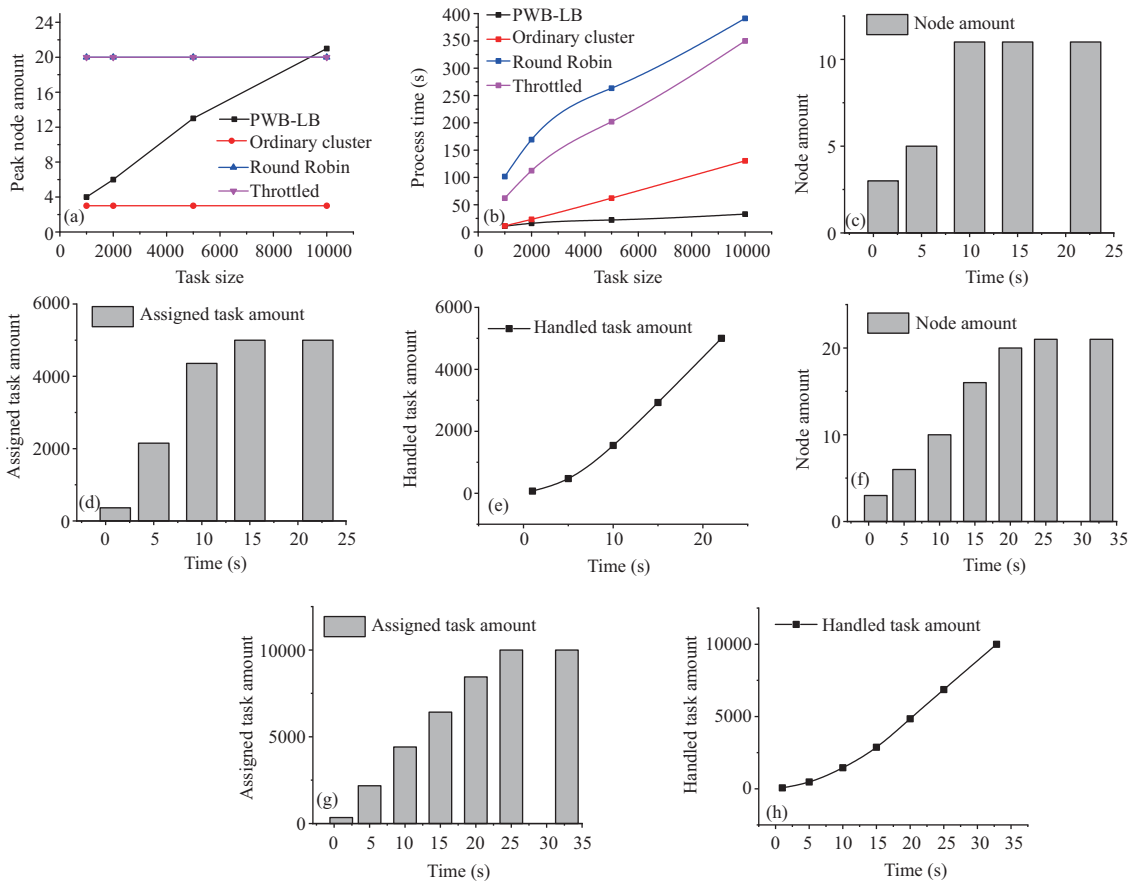

Figure 5 (Color online) Auto-scalability and performance evaluation. (a) Auto-scalability comparison; (b) comparison of efficiency; (c) child nodes under task 5000; (d) task assignment under 5000; (e) task processing under 5000; (f) child nodes under 10000; (g) task allocation under 10000; (h) task processing under 10000

6.2 Performance

The performance of PWB-LB is mainly determined using three basic metrics: system load, availability, and performance. As they have been defined, a simulated request processing scenario could give an intuitive view of these metrics. Thus, we simulate a system running process to evaluate it in three main phases: the system starts new nodes to join $Q_{\rm cc}$ on receiving new tasks, assigns tasks to different child nodes through the load balancing strategy, and schedules child nodes to handle the tasks. In our experiment, we use the "number of computing nodes", "assigned tasks", and "processed tasks" at different times as our measuring units to evaluate the PWB-LB's performance. Figures 5(c)–(e) show the experiment results from the time 1 to 22.07 s for 5000 tasks.

• Node amount. As shown in Figure 5(c), new computing nodes are timely instantiated to balance the system load, besides, few nodes are activated to ensure judicious use of resources. For example, 5 child nodes are activated at the time 5 s, and 11 nodes are activated at the time 10 s until the whole tasks are processed.

• Assigned task amount. As shown in Figure 5(d), it takes rather less time to assign all tasks to child nodes. For example, 4359 tasks have been assigned at the time 10 s, and all tasks at the time 15 s.

• Handled task amount. As shown in Figure 5(e), the number of tasks processed almost exponentially grows with processing time, and it takes few time to complete all tasks. For example, 472 tasks are processed in 5 s, 2930 tasks are processed in 15 s, and it takes 22.07 s to process all 5000 tasks.

Figures 5(f)–(h) show the experimental results from time 1 to 32.85 s for 10000 tasks. Following the same conclusions from Figures 5(c)–(e), we have the following conclusions:

• Node amount. As shown in Figures 5(c) and (f), there are more child nodes and time costs when increasing the number of tasks. However, less processing time and new child nodes are needed compared with the number of extra new tasks. Thus, our design is efficient and resource-saving.

• Assigned task amount. As shown in Figures 5(d) and (g), it takes more time to assign all tasks. However, the assigning rate becomes higher with more new tasks, and the time consumption is low.

• Handled task amount. As shown in Figures 5(e) and (h), more time is required for handling all tasks with an increasing number of tasks. However, the processing rate greatly increases and less time is needed with the addition of new tasks, hence, the time consumption is low.

In conclusion, the experimental evaluation confirms that the PWB-LB mechanism is auto-scalable, more efficient, and well-performed, which is more suitable for processing highly concurrent and complex tasks.

7 Conclusion

We have presented the PoW-based auto-scalable and fault-tolerant PWB-LB. Since the load balancing strategy and new virtual nodes are rapidly established by mirroring, and the idle nodes are dynamically deactivated, our PWB-LB has high efficiency and less resource consumption compared with existing methods. By taking advantage of the simple PoW consensus used for master nodes voting and management, PWB-LB can ensure fault-tolerance for master nodes. Furthermore, the modular design for different working functions of the load balancing assembly and the applied mirroring for managing computing resources make PWB-LB an intelligent and automatic load balancing mechanism. Finally, this mechanism can be widely applied to various cloud computing platforms. Through our experimental results, we demonstrate that PWB-LB is a suitable component to be used in cloud platforms, such as E-commerce to ensure dynamic processing of requests and fault-tolerance for participating nodes.

Acknowledgements This work was supported in part by Key Program of National Natural Science Foundation of China (NSFC) (Grant No. U1405255), in part by Shaanxi Science & Technology Coordination & Innovation Project (Grant No. 2016KTZDGY05-06), in part by Fundamental Research Funds for the Central Universities (Grant No. SA-ZD161504), in part by National Natural Science Foundation of China (Grant No. 61702404), in part by Fundamental Research Funds for the Central Universities (Grant No. JB171504), in part by Project Funded by China Postdoctoral Science Foundation (Grant No. 2017M613080), in part by Major Nature Science Foundation of China (Grant Nos. 61370078, 61309016), in part by Key Research and Development Plan of Jiangxi Province (Grant No. 0181ACE5002).

References

1 Miao Y B, Liu X M, Choo K K R, et al. Fair and dynamic data sharing framework in cloud-assisted internet of everything. IEEE Int Things J, 2019, 6: 7201–7212
2 Mahato D P, Singh R S. Load balanced transaction scheduling using honey bee optimization considering performability in on-demand computing system. Concurr Comput-Pract Exper, 2017, 29: 4253
3 Thai M T, Lin Y D, Lin P C, et al. Towards load-balanced service chaining by Hash-based traffic steering on softswitches. J Netw Comput Appl, 2018, 109: 1–10
4 Kouchaksaraei H R, Dräxler S, Peuster M, et al. Programmable and flexible management and orchestration of virtualized network functions. In: Proceedings of European Conference on Networks and Communications, Ljubljana, 2018
5 Dezhabad N, Sharifian S. Learning-based dynamic scalable load-balanced firewall as a service in network function-virtualized cloud computing environments. J Supercomput, 2018, 74: 3329–3358
6 Wang Q, Zhu M J, Yu Y R, et al. A load balance mechanism in heterogeneous network based on utility function. J Commun, 2016, 11: 871–878
7 Mohanty S, Patra P K, Ray M, et al. An approach for load balancing in cloud computing using JAYA algorithm. Int J Inf Tech Web Eng, 2019, 14: 27–41
8 Yang J P. Intelligent offload detection for achieving approximately optimal load balancing. IEEE Access, 2018, 6: 58609–58618
9 Youssef F, Habib B L E, Hamza R, et al. A new conception of load balancing in cloud computing using tasks classification levels. Int J Cloud Appl Comput, 2018, 8: 118–133
10 Judmayer A, Stifter N, Krombholz K, et al. Blocks and chains: introduction to bitcoin, cryptocurrencies, and their consensus mechanisms. Synth Lect Inf Secur Priv Trust, 2017, 9: 1–123
11 Zhang Y H, Deng R H, Liu X M, et al. Blockchain based efficient and robust fair payment for outsourcing services in cloud computing. Inf Sci, 2018, 462: 262–277
12 He Y H, Li H, Cheng X Z, et al. A blockchain based truthful incentive mechanism for distributed P2P applications. IEEE Access, 2018, 6: 27324–27335
13 Nakamoto S. Bitcoin: a peer-to-peer electronic cash system. 2008. http://www.spacepirates.com/bitcoin.pdf
14 Liu X M, Qin B D, Deng R H, et al. A privacy-preserving outsourced functional computation framework across large-scale multiple encrypted domains. IEEE Trans Comput, 2016, 65: 3567–3579
15 Chen L, Xu L, Gao Z M, et al. Protecting early stage proof-of-work based public blockchain. In: Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, Luxembourg, 2018. 122–127
16 Bentov I, Lee C, Mizrahi A, et al. Proof of activity: extending bitcoin's proof of work via proof of stake [extended abstract]y. SIGMETRICS Perform Eval Rev, 2014, 42: 34–37
17 Huang W H, Ma Z, Dai X F, et al. Fuzzy clustering with feature weight preferences for load balancing in cloud. Int J Soft Eng Knowl Eng, 2018, 28: 593–617
18 Zhao J, Yang K, Wei X H, et al. A heuristic clustering-based task deployment approach for load balancing using Bayes theorem in cloud environment. IEEE Trans Parallel Distrib Syst, 2016, 27: 305–316
19 Miao Y B, Ma J F, Liu X M, et al. Practical attribute-based multi-keyword search scheme in mobile crowdsourcing. IEEE Int Things J, 2018, 5: 3008–3018

20 Mohiuddin I, Almogren A. Workload aware VM consolidation method in edge/cloud computing for IoT applications. J Parallel Distrib Comput, 2019, 123: 204–214

21 Carneiro T, da Nobrega R V M, Nepomuceno T, et al. Performance analysis of Google Colaboratory as a tool for accelerating deep learning applications. IEEE Access, 2018, 6: 61677–61685

22 Grevet C, Choi D, Kumar D, et al. Overload is overloaded: email in the age of Gmail. In: Proceedings of SIGCHI Conference on Human Factors in Computing Systems, Toronto, 2014. 793–802

23 Kumar L, Mutanga O. Google earth engine applications since inception: usage, trends, and potential. Remote Sens, 2018, 10: 1509

24 O'Brien P, Young S W H, Arlitsch K, et al. Protecting privacy on the web: a study of HTTPS and google analytics implementation in academic library websites. Online Inform Rev, 2018, 42: 734–751

25 Tan W A, Lu G Z, Sun Y, et al. A concurrent level based scheduling for workflow applications within cloud computing environment. In: Proceedings of Joint International Conference on Pervasive Computing and the Networked World, Vina del Mar, 2013. 400–411

26 Chen J, Wang Q G, Zhu S N. An improvement on load-balancing on Linux virtual server for internet-based laboratory. In: Proceedings of the 12th IEEE International Conference on Control and Automation, Kathmandu, 2016. 685–689

27 Chi X N, Liu B C, Niu Q, et al. Web load balance and cache optimization design based Nginx under high-concurrency environment. In: Proceedings of the 3rd International Conference on Digital Manufacturing and Automation, Guilin, 2012. 1029–1032

28 Sahana S, Mukherjee T, Sarddar D. A conceptual framework towards implementing a cloud-based dynamic load balancer using a weighted round-robin algorithm. Int J Cloud Appl Comput, 2020, 10: 22–35

29 Hussein W, Peng T, Wang G J. A weighted throttled load balancing approach for virtual machines in cloud environment. Int J Comput Sci Eng, 2015, 11: 402–408

30 Vukolic M. The quest for scalable blockchain fabric: proof-of-work vs. BFT replication. In: Proceedings of International Workshop on Open Problems in Network Security, Zurich, 2015. 112–125

31 Naoumov N, Ross K. Exploiting P2P systems for DDoS attacks. In: Proceedings of the 1st International Conference on Scalable Information Systems, Hong Kong, 2006

32 Mainkar V. Availability analysis of transaction processing systems based on user-perceived performance. In: Proceeding of the 16th Symposium on Reliable Distributed Systems, North Carolina, 1997. 10–16

33 Kavi K M, Youn H Y, Shirazi B A, et al. A performability model for soft real-time systems. In: Proceeding of the 27th Annual Hawaii International Conference on System Sciences, Maui, 1994. 571–580