

Online Elastic Resource Provisioning With QoS Guarantee in Container-Based Cloud Computing

Shuaibing Lu ¹, Member, IEEE, Ran Yan, Jie Wu ², Fellow, IEEE, Jackson Yang, Xinyu Deng, Shen Wu ³, Zhi Cai ⁴, Member, IEEE, and Juan Fang ⁵, Member, IEEE

Abstract—In cloud data centers, the exponential growth of data places increasing demands on computing, storage, and network resources, especially in multi-tenant environments. While this growth is crucial for ensuring Quality of Service (QoS), it also introduces challenges such as fluctuating resource requirements and static container configurations, which can lead to resource underutilization and high energy consumption. This article addresses online resource provisioning and efficient scheduling for multi-tenant environments, aiming to minimize energy consumption while balancing elasticity and QoS requirements. To address this, we propose a novel optimization framework that reformulates the resource provisioning problem into a more manageable form. By reducing the original multi-constraint optimization to a container placement problem, we apply the interior-point barrier method to simplify the optimization, integrating constraints directly into the objective function for efficient computation. We also introduce elasticity as a key parameter to balance energy consumption with autonomous resource scaling, ensuring that resource consolidation does not compromise system flexibility. The proposed Energy-Efficient and Elastic Resource Provisioning (EEP) framework comprises three main modules: a distributed resource management module that employs vertical partitioning and dynamic leader election for adaptive resource allocation; a prediction module using ω -step prediction for accurate resource demand forecasting; and an elastic scheduling module that dynamically adjusts to tenant scaling needs, optimizing resource allocation and minimizing energy consumption. Extensive experiments across diverse cloud scenarios demonstrate that the EEP framework significantly improves energy efficiency and resource utilization compared to established baselines, supporting sustainable cloud management practices.

Index Terms—Data center network, energy-efficient, elastic scheduling, QoS, resource provisioning.

Received 23 July 2024; revised 23 November 2024; accepted 20 December 2024. Date of publication 24 December 2024; date of current version 20 January 2025. This work was supported in part by the Fundamental Research Funds for the Central Universities under Grant 2021RC258, in part by the China Postdoctoral Science Foundation under Grant 2021M700366, in part by the National Natural Science Foundation under Grant 92267107, and in part by the National Natural Science Foundation under Grant 62072016. Recommended for acceptance by R. Prodan. (Corresponding author: Zhi Cai.)

Shuaibing Lu, Ran Yan, Shen Wu, Zhi Cai, and Juan Fang are with the College of Computer Science, Beijing University of Technology, Beijing 100124, China (e-mail: lushuaibing@bjut.edu.cn; yanran@emails.bjut.edu.cn; wushen@emails.bjut.edu.cn; caiz@bjut.edu.cn; fangjuan@bjut.edu.cn).

Jie Wu is with the Center for Networked Computing, Temple University, Philadelphia, PA 19122 USA (e-mail: jiewu@temple.edu).

Jackson Yang is with the School of Software Engineering, Beijing Jiaotong University, Beijing 100044, China (e-mail: 21309001@bjtu.edu.cn).

Xinyu Deng is with the University of Southern California Viterbi School of Engineering, Los Angeles, CA 90089-1450 USA (e-mail: xinyud@usc.edu).

Digital Object Identifier 10.1109/TPDS.2024.3522085

I. INTRODUCTION

WITH the rapid advancement of information technology and the advent of cloud computing, data centers have emerged as a fundamental component of the infrastructure underpinning modern society. These facilities are pivotal in hosting critical workloads for diverse applications, including cloud services, Big Data analytics, and artificial intelligence. Moreover, they are required to provide both individuals and enterprises with seamless access to essential resources, namely storage, computational power, and networking capabilities. In response to the escalating resource demands in these environments, containerization has emerged as a highly effective solution. By encapsulating applications within isolated, lightweight containers, this paradigm enables expedited deployment, enhanced scalability, and more efficient resource utilization. Unlike traditional virtual machines, which each require a complete operating system, containers share the host system's kernel, optimizing both performance and resource management. This methodology has significantly augmented the capacity of data centers to dynamically allocate resources, ensuring they can adaptively support the burgeoning requirements of cloud services and applications. To better comprehend the resource provisioning problem, we utilize virtual clusters to illustrate multi-tenant requests. Each virtual cluster serves as a conceptual model of a group of containers from a single tenant, each with distinct requirements for both communication and computing resources [1], [2]. However, tenant deployment sizes exhibit extreme volatility with respect to resource demands, compelling cloud data centers to dynamically scale container instances to meet performance imperatives amid fluctuating workloads. Therefore, the development of an efficient provisioning mechanism is imperative for optimizing resource utilization and minimizing energy consumption, particularly within multi-tenant environments in large-scale data centers.

A. Motivation and Challenges

In this paper, we focus on optimizing energy efficiency and elasticity during the resource provisioning process with QoS guarantee in the context of container-based cloud computing for multi-tenants. By relaxing certain constraints and focusing exclusively on minimizing energy consumption, this complex problem can be approximated as a virtual machine placement problem across multiple servers, which represents a well-established NP-hard problem and is comparable to

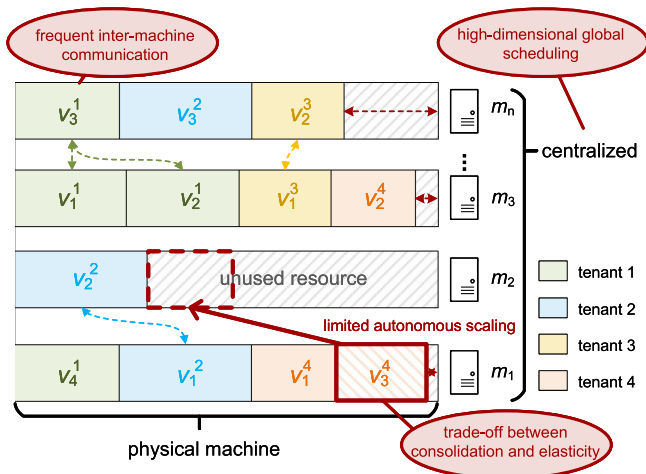


Fig. 1. Illustrate the challenges of resource provisioning in cloud data centers, addressing elastic scaling, bottleneck mitigation, and multi-tenant resource allocation constraints. Each tenant's requests are represented by a set of containers; for instance, $\{v_1^1, v_2^1, v_3^1\}$ represent containers from tenant 1.

the multiple knapsack problem. However, unlike traditional placement problems, this scenario involves online resource provisioning in container-based environments, introducing additional complexity due to its large scale and the dynamic variability of user demands. Therefore, achieving efficient resource provisioning in container-based cloud environments presents several key challenges (see Fig. 1).

First and foremost, while containerization offers flexibility, improper placement and scheduling of container instances can cause resource fragmentation across servers, leading to suboptimal utilization and higher energy consumption. Furthermore, ineffective placement decisions may cause containers within the same virtual cluster to be spread across different servers, leading to frequent inter-machine communication. Proper consolidation and placement can reduce resource fragmentation among servers, prevent resources from being dispersed across different servers and underutilized, enhance resource utilization efficiency, and lower energy consumption. However, excessive consolidation of containers onto fewer servers inevitably reduces elasticity, resulting in limited autonomous scaling capabilities. Therefore, effectively managing resources in cloud data centers involves a complex trade-off between consolidation and elasticity to optimize energy consumption and improve overall performance, which is non-trivial.

Additionally, the fluctuating resource demands of multi-tenant workloads often result in sudden spikes, complicating resource allocation and scaling strategies. While Kubernetes schedulers can automatically allocate workloads based on resource demands and cluster load [3], [4], [5], scaling is often limited by the physical resources available on the allocated nodes, leading to inefficiencies during resource shortages. Existing approaches, such as static provisioning models with pre-allocated resources [1], [2], [6], [7], [11], [12], [13], frequently result in resource underutilization and increased energy consumption, as they struggle to adapt to dynamic conditions. Consequently, a key challenge lies in

ensuring elastic scaling for multiple tenants with diverse requirements, while minimizing manual intervention and avoiding frequent reconfiguration of resource provisioning.

Furthermore, the dynamic nature of resource requirements in large-scale cloud data centers necessitates an agile and adaptive provisioning mechanism that responds in real time to workload fluctuations while maintaining Quality of Service (QoS) for all tenants. However, addressing dynamic resource requests on a global data center scale introduces high-dimensional complexity, making it computationally expensive to achieve optimal scheduling. In many cases, adjustments made during application deployment, such as function-level scaling, can cause resource changes. These adjustments may sometimes be resolved locally within pods, avoiding the need for a global search across the data center. Given the inherent high dimensionality in large-scale scheduling, an efficient grouping strategy becomes essential to reduce complexity, enabling scalable and responsive resource management. Therefore, this paper explores comprehensive strategies to achieve sustainable management and optimization of data centers. Emphasizing energy efficiency, resilience, and service quality, this research delves into dynamic management and optimization methods to enhance data center operations. Insights derived aim to guide data center operators, businesses, and researchers in improving efficiency, reducing energy consumption, and maintaining high-quality services. The subsequent sections detail the proposed approach and present experimental results to demonstrate its practical feasibility and effectiveness.

B. Contributions and Paper Organization

Distinct from the broader objectives, this paper specifically addresses operational challenges in online resource provisioning and efficient scheduling within cloud data centers, with a dual focus on minimizing energy consumption and integrating elasticity with QoS for multi-tenant environments. The contributions are articulated as follows:

- We model resource provisioning in multi-tenant cloud data centers as a joint optimization problem with resource constraints. This model is designed to reduce overall energy consumption and operational costs while ensuring combinatorial elasticity and QoS for multiple tenants.
- We reformulate the original complex optimization problem into an unconstrained form by applying an effective optimization strategy, demonstrating that the resulting solution adheres to specific performance bounds. This transformation establishes a robust foundation for further optimization, with logical connections within the proofs to ensure practical applicability.
- We propose a framework named EEP for resource provisioning, encompassing three primary modules: vertical partitioning, prediction, and elastic scheduling. A novel vertical partitioning module integrates a leader election mechanism. This is complemented by a prediction module that incorporates ω -step prediction strategies tailored to hierarchical levels. A distributed scheduling module is then designed to accommodate the elastic scaling of multiple tenants.

- We conducted extensive experimental analyses using the GWA-T-12 Bitbrains real-world dataset, encompassing performance metrics from 1,750 virtual machines in a distributed data center, supplemented by tests on the Huawei cloud platform. Additionally, to further validate the effectiveness of EEP, we implemented a prototype on Kubernetes and conducted experiments under practical deployment scenarios. The results, evaluated from multiple perspectives, demonstrate significant improvements in reducing overall energy consumption while ensuring efficient resource utilization.

The rest of the paper is structured as follows: Section II surveys related works. Section III describes the model and then formulates the problem. Section IV investigates the problem transformation. Section V discusses the energy-efficient strategy with elastic resource management. Section VI presents the experiments. Finally, Section VII concludes the paper.

II. RELATED WORK

The exponential growth of data volumes has also brought the issues of energy consumption and elasticity in data centers to the forefront. We classify previous works on resource provisioning in cloud data centers into two aspects.

Energy and QoS: To better conserve data center energy consumption and enhance user QoS, various mechanisms focus on resource provisioning to achieve this goal. These include but are not limited to: 1) consolidation [1], [2], [6], [7], [11], [12], [13], [14], [17], [27], [28], [29], [30], [31], [32]. Wen et al. [1] proposed a joint resource combination and container placement strategy for serverless functions with heterogeneous containers by leveraging the vertical scaling feature of Kubernetes for in-place pods. Hieu et al. [2] proposed a virtual machine consolidation algorithm that incorporates multiple usage predictions to improve energy efficiency in cloud data centers. Khan et al. [6] proposed a heuristic method that employs container migration strategies to reduce energy consumption without compromising the quality of service for data centers. Khemili et al. [7] proposed an energy-aware fuzzy approach to combine the virtual network function in a minimum number of virtual machines to optimize resource utilization and reduce energy consumption. Zeng et al. [15] proposed an adaptive virtual machine consolidation framework based on deep reinforcement learning to reduce energy consumption; 2) reallocation [8], [9], [10], [16]. Li et al. [8] proposed an efficient virtual machine placement method to optimize energy efficiency and resource utilization; Lin et al. [9] proposed a peak efficiency-aware scheduling strategy, to redistribute virtual machines in an online manner dynamically. Zhong et al. [10] proposed a heterogeneous task allocation strategy for cost-efficient container orchestration through resource utilization optimization and elastic instance pricing while reallocating the relevant jobs without losing task progress. Saxena et al. [16] proposed a framework that ensures an energy-efficient distribution of physical resources by reallocating the virtual machines subsequently on the arrival of new requests and deallocating on completion of respective tasks to handle dynamic workload

demands of cloud users. However, they did not consider the elastic scaling for multiple tenants, where fluctuations in resource demand could lead to underutilization of resources.

Elasticity: There also have been a few recent works on optimizing resource utilization and elasticity of the data center [17], [18], [19], [20], [21], [25], [26]. To improve the performance of the data center, Saxena et al. [18] proposed a load-balancing method based on online resource prediction for elastic resource management in cloud data centers. However, this approach is not suitable for dynamic scaling in use as it incurs additional migration overhead, impacting user QoS. Naji et al. [19] proposed a fault-tolerant elastic resource management framework to reduce performance degradation caused by overload, SLA violations, and excessive power consumption. Yazdanbakhsh et al. [20] proposed a multi-objective strategy for dynamic task scheduling through elastic cloud resources to construct individual non-dominated sets of newly received tasks. Zolfaghari et al. [21] proposed an improved particle swarm optimization-based energy perception method for virtual machine integration problems to optimize energy consumption and ensure the quality of service. Georgios et al. [22] proposed a reactive data-aware dynamic scaling mechanism for cloud resources, which takes into account the data dependencies of workloads to prevent data loss in the data-aware dynamic scaling process. Feng et al. [23] proposed a proactive server elasticity scaling method, which guides server scaling by sensing workload levels, trends, and amplitude variations. Zhu et al. [24] proposed a dynamic fuzzy scheduling framework that collects newly arrived tasks, monitors task execution status, and periodically updates virtual machine availability, generating a fuzzy schedule for the collected tasks. Saxena et al. [25] proposed a fault-tolerant elastic resource management framework that addresses the aforementioned problem from a different perspective by inducing high availability in servers and VMs. Yu et al. [26] proposed a novel framework to automatically identify the scaling-needed services and scale them to meet the service level agreement with an optimal cost for microservice applications. Zakarya et al. [17] showed how virtual machine layout strategies affect energy efficiency, workload performance, and user cost, and proposed a location-based aggregation method for improvement. The majority of the aforementioned investigations employ a centralized virtual machine consolidation approach to optimize energy consumption or resource allocation in cloud data centers. However, the centralized management lacks real-time state capture for servers and faces challenges in scaling for multi-tenant environments. In this paper, we address the challenges of resource scheduling and effective integration for multi-tenant environments in cloud data centers, aiming for energy efficiency and high elasticity. The symbols and definitions used in this paper are summarized in Table I.

III. MODEL AND PROBLEM FORMULATION

This section introduces the models of the data center and virtual clusters. Additionally, we formalize the problem of minimizing energy consumption while considering elasticity and quality of service.

TABLE I
SUMMARY OF SYMBOLS IN THE SYSTEM MODEL

Symbol	Definition
M	set of physical machines, where $M = \{m_i\}$.
L	set of physical links, where $L = \{L_j\}$.
\mathcal{V}	set of virtual clusters, where $\mathcal{V} = \{v^h\}$.
v_k^h	k -th container belongs to virtual cluster h .
ξ	$\xi \in \{o, c, r, b\}$ represent the types.
$U_{m_i}^\xi$	utilization of physical machine m_i .
$b(l_j)$	allocated bandwidth resources on link l_j .
$\hat{b}(l_j)$	utilized bandwidth on link l_j .
$o(m_i)$	number of cores in physical machine m_i .
$\hat{o}(m_i)$	number of utilized cores in physical machine m_i .
$r(m_i)$	storage capacity of physical machine m_i .
$\hat{r}(m_i)$	utilized memory resources of physical machine m_i .
$R^\xi(\cdot)$	raw weight of resource metric ξ .
$\mathcal{P}(m_i)$	power consumption of physical machine m_i .
$\Delta(v_k^h)$	QoS-guarantee consumption of v_k^h .
\mathbf{E}	combinational elasticity of system.
\mathbf{Q}	overall consumption of the cloud data center.

A. Data Center Network

In this paper, we define the substrate topology of the data center network as a fat-tree structure. The set of physical machines is denoted by M , where $M = \{m_i\}$, and $|M|$ represents the total number of physical machines. The model of the physical server in the data center is conceptualized as a collection of resources within an actual physical machine, encompassing cores, CPU usage, and storage. We use $o(m_i)$ to represent the number of cores, and $r(m_i)$ denotes the storage capacity of m_i . Specifically, let $\hat{o}(m_i)$ represent the number of cores in the allocated physical resources, and $\hat{r}(m_i)$ represent the allocated memory resources, respectively. Based on the fat-tree data center topological structure, we use L to denote the set of physical links, where $L = \{L_j\}$. Let $|L|$ represent the total number of physical links. Here, L_j denotes the j -th link, with a capacity represented by $b(l_j)$. For each utilized link, $\hat{b}(l_j)$ signifies the allocated bandwidth resources.

B. Virtual Clusters

To precisely delineate the requirements imposed by multiple tenants in cloud data centers, virtual clusters are utilized to encapsulate the collective requests from individual tenants. In the provisioning process, tenant requests are provisioned in the form of containers, with each virtual cluster consisting of multiple containers that collectively manage the required resources. Our model supports configurations where multiple containers can share a single virtual machine, addressing specific challenges unique to containerized clouds, such as container orchestration and multi-tenancy. Specifically, each container, denoted by v_k from the set $\mathcal{V} = \{v_k\}$, is assigned to a specific virtual cluster, v_k^h , which corresponds to the k -th container belongs to virtual cluster h . The attributes of v_k include $o(v_k)$, representing the number of cores, and $r(v_k)$ specifying the required storage resources. Additionally, $b(v_k^h)$ quantifies the bandwidth necessary for communication within the virtual cluster h . This modeling ensures that all containers requested by a single tenant are

cohesively managed and optimally configured to meet specific service levels.

C. Problem Formulation

1) *Energy Consumption*: Servers constitute the principal source of energy consumption within cloud data centers [34], with their power usage intricately tied to resource utilization. We use $P(m_i)$ to represent the power consumption of the physical machine m_i , which can be expressed as

$$P(m_i) = (P_f(m_i) - P_l(m_i)) \cdot U^c(m_i) \cdot U^o(m_i) + P_l(m_i). \quad (1)$$

Here, $P_f(m_i)$ is the power consumption of m_i under the full load status, and $P_l(m_i)$ is that m_i under the idle status with low power. We use $U^c(m_i)$ to denote the CPU utilization of m_i , and we use $U^o(m_i)$ to denote the core utilization of m_i , where $U^o(m_i) = \hat{o}(m_i)/o(m_i)$. Here, $\hat{o}(m_i)$ represents the number of cores currently occupied by containers, where $\hat{o}(m_i) = \sum_{v_k \in m_i} o(v_k)$, and $o(m_i)$ denotes the total number of cores available on m_i . To assess the overall utilization of the physical machine m_i , we introduce the product $U^c(m_i) \cdot U^o(m_i)$, which captures both the occupied and remaining cores. A detailed explanation of this calculation method is provided in Appendix A, available online. Additionally, let $\mathcal{P}(m_i)$ represent the continuous energy consumption of m_i from time slot t_0 to t_1 , where

$$\mathcal{P}(m_i) = \int_{t_0}^{t_1} P_t(m_i) dt. \quad (2)$$

2) *QoS-Guarantee Consumption*: We aim to decrease the overall energy consumption of data centers by consolidating resources for physical machines with lower utilization. However, this process entails additional consumption due to communication and migration, which significantly impacts the QoS for multi-tenants. The QoS guarantee consumption is defined as follows.

$$\Delta(v_k^h) = \Delta_b(v_k^h) + \Delta_m(v_k^h). \quad (3)$$

We use $\Delta_b(v_k^h)$ to denote the communication cost associated with the bandwidth demand of containers for each cluster, which is defined as $\Delta_b(v_k^h) = b(v_k^h) \cdot \Upsilon_{(v_k^h, v_{k'}^h)} \cdot \eta$. Here, let $b(v_k^h)$ represent the communication demand of v_k^h , and $\Upsilon_{(v_k^h, v_{k'}^h)}$ represent the number of hops in the communication link between container v_k^h and another container $v_{k'}^h$ in the same cluster, η is the unit cost of bandwidth resources. Subsequently, we use $\Delta_m(v_k^h)$ to denote the migration cost of one container that is made up of the remaining portion of the QoS-guarantee consumption, which is defined as $\Delta_m(v_k^h) = r(v_k^h) \cdot \rho(v_k^h)$. Here, we use $\rho(v_k^h)$ to indicate whether v_k^h migrates, which is a boolean value. Since we have already determined the clusters to which each container belongs, we will exclude redundant clusters from the calculation of the final objective function. Instead, we will comprehensively consider the total consumption of virtual clusters to ensure QoS guarantees.

3) *Elasticity*: We quantify the potential growth of multi-tenant computing and communication resources simultaneously using the concept of elasticity, which is the degree to which a system can adjust to changes in workload by automatically

allocating and releasing resources. We take four metrics, which are the core utilization $U^o(m_i)$, CPU utilization $U^c(m_i)$, storage utilization $U^r(m_i)$, and the bandwidth resources utilization $U^b(m_i)$, into consideration for the combinational elasticity. We use $U^\xi(\cdot)$ as a unified representation, where $\xi \in \{o, c, r, b\}$. To differentiate between the attributes of physical machines and links, we categorize their properties into two distinct sets, denoted as $\mathcal{A}_m = \{o, c, r\}$ and $\mathcal{A}_l = \{b\}$. Then each of the metrics will have a raw weight which would be relatively lower when the resource it represents is facing a shortage, and the definition is shown as follows.

Definition 1 (Raw Weight): Let R^ξ indicate the raw weight of resource metric ξ which is relatively lower when the resources it represents are facing a shortage, i.e., $\xi \in \{o, c, r, b\}$,

$$R^\xi(\cdot) = 1 + \ln(1 - \varpi \cdot U^\xi(\cdot)) \quad (4)$$

and o, c, r , and b represent the core, CPU, storage, and bandwidth resources provided by the data center, respectively. Here, $R^\xi(m_i)|_{\xi \in \mathcal{A}_m}$ represents the raw weight of the physical machine m_i . Simultaneously, $R^\xi(l_j)|_{\xi \in \mathcal{A}_l}$ represents the raw weight of the bandwidth resource of physical link l_j .

Here, ϖ is the coefficient of each type of resource on the physical machine, where $\varpi = 1 - 1/e$. Based on that, we use elasticity E_{m_i} to denote the scalability of computation and storage resources on the physical machine m_i , which is defined as $E_{m_i} = \min\{R^\xi(m_i)|_{\xi \in \mathcal{A}_m}\}$. Moreover, the physical machine with the lowest elasticity value determines which machine in the data center will be the bottleneck for scaling the virtual requests of multi-tenants. Consequently, the following formula represents the elasticity of the set of physical machines M , where $\mathbf{E}_M = \min_{m_i \in M}\{E_{m_i}\}$. We use elasticity E_{l_j} to denote the elasticity of the communication resource of the physical link l_j , which is defined as $E_{l_j} = R^\xi(l_j)|_{\xi \in \mathcal{A}_l}$. Similarly, the smallest elastic physical link determines how scalable communication resources can be made. Therefore, the formula for the elasticity of L is as $\mathbf{E}_L = \min_{l_j \in L}\{E_{l_j}\}$. The combinational elasticity considers both computational and communication resources, which is defined as $\mathbf{E} = \min_{M, L \in G}\{\mathbf{E}_M, \mathbf{E}_L\}$.

This study addresses the problem of energy-efficient and elastic resource provisioning within multi-tenant cloud data centers. The total consumption within the cloud data center, incorporating both energy and QoS guarantee costs due to resource provisioning, is denoted by \mathbf{Q} . The primary goal is to minimize \mathbf{Q} by holistically considering the dynamics of combinational elasticity, resource constraints, and the incidence of service level agreement (SLA) violations. The formal problem statement is presented as follows:

$$\mathbf{P}_1 : \text{minimize } \mathbf{Q} = \sum_{i=1}^{|M|} \sum_{k=1}^{|V|} (\mathcal{P}(m_i) + \Delta(v_k)) \quad (5)$$

$$\text{s.t. } \mathbf{E} \geq \tau \quad (6)$$

$$\hat{o}(m_i) \leq o(m_i), \hat{r}(m_i) \leq r(m_i), \forall m_i \in M \quad (7)$$

$$\hat{b}(l_j) \leq b(l_j), \forall l_j \in L \quad (8)$$

Equation (5) specifies the objective function, while (6) through (8) outline the constraints. In (6), τ is used as the threshold for combinational elasticity within the cloud data center. Equations (7) and (8) encapsulate constraints related to resource provisioning, ensuring that the aggregate utilization of computing and networking resources does not surpass the capabilities of the physical machines and communication links.

IV. PROBLEM TRANSFORMATION

This section presents the transformation of the complex optimization problem into an unconstrained framework by employing the interior-point barrier method, which significantly simplifies the optimization process through the integration of constraints into the objective function. However, a direct transformation of the original objective function \mathbf{P}_1 is not feasible due to its composite nature, encompassing both server-oriented energy consumption and the energy needs to be related to tenant service guarantees. These components involve distinct yet inter-dependent variables, rendering joint optimization challenging. To address this issue, the objective function is partitioned as shown in (5). The energy consumption for Quality of Service (QoS) guarantees, denoted as $\Delta(v_k^h)$, is calculated using the Gurobi optimizer to solve for the decision vector $\bar{\rho}$ of the virtual clusters. This vector is then reintegrated into the objective function, transforming it into a function solely dependent on the utilization U^ξ . Subsequently, the interior-point barrier method is applied to this reformulated objective function. Detailed steps for this transformation are provided in Appendix B, available online. Furthermore, elasticity is introduced as a critical parameter to reflect the autonomous resource scaling capability of data centers. Defined as $\mathbf{E} = \min\{\mathbf{E}_M, \mathbf{E}_L\}$ in (6), this parameter requires further transformation, with details also included in Appendix C, available online. These transformations allow for a more efficient optimization of the objective function \mathbf{P}_1 , effectively balancing operational demands and resource utilization. Following that, we introduce an effective optimization method to transform the problem \mathbf{P}_1 , which arises from the consolidation of constraints into an unconstrained problem. The reduction to the standard form of the interior-point barrier algorithm is as follows:

$$\begin{aligned} \mathbf{P}_2 : \text{minimize } & f(x) \\ \text{s.t. } & \zeta_i(x) \leq 0, \quad i = 1, 2, 3. \end{aligned} \quad (9)$$

In the above equation, $x = U_\xi, \xi \in \{o, c, r, b\}$, $f(x) = Q(U_\xi)$, $\zeta_1(x)$, $\zeta_2(x)$, $\zeta_3(x)$ are equivalent to (6), (7) and (8). To address this problem, we apply the interior-point barrier method, which involves introducing a barrier function $\varphi(\eta) = 0(\eta \leq 0)$, otherwise $\varphi(\eta) = +\infty(\eta > 0)$ to transform the initial constrained problem into an unconstrained one. This transformation allows us to work with an unconstrained optimization problem, denoted as \mathbf{P}_3 :

$$\mathbf{P}_3 : \text{minimize } f(x) + \sum_{i=1}^3 \varphi(\zeta_i(x)). \quad (10)$$

When the solution of \mathbf{P}_3 satisfies the two constraints in \mathbf{P}_2 , $\varphi(\zeta_i(x)) = 0$, which implies that searching for the solution of \mathbf{P}_3 is equivalent to seeking a solution for \mathbf{P}_2 . If the solution of \mathbf{P}_3 doesn't satisfy any of the constraints in \mathbf{P}_2 , then \mathbf{P}_3 becomes infinite, making this solution unfeasible. Here, $\varphi(\eta)$ acts as a non-differentiable barrier function, penalizing infeasible solutions. The objective function is enhanced by adding barrier terms to guide the optimization process. For computational convenience, we approximate $\varphi(\eta)$ using the differentiable function $\hat{\varphi}(\eta) = -\frac{1}{\psi} \cdot \log(-\eta)$, and we define $\Phi(x) = -\sum_{i=1}^3 \log(-\zeta_i(x))$, which is the log barrier function. Thus, (10) can be converted to $\frac{1}{\psi}\Phi(x) + f(x)$ that equivalent to $\frac{1}{\psi}(\Phi(x) + \psi \cdot f(x))$. Due to ψ is a defined value that larger than 0, we have:

$$\mathbf{P}_4 : \text{minimize } \Phi(x) + \psi \cdot f(x). \quad (11)$$

As ψ increases, the accuracy improves and $\hat{\varphi}(\eta)$ approximates $\varphi(\eta)$, ensuring that the solution to the transformed problem closely matches the optimal value of the original problem, maintaining both accuracy and alignment with the original objectives. Theorem 1 demonstrates this relationship, and the detailed proof is included in the appendix, available online, for brevity.

Theorem 1: The optimal solution x_{ψ}^* obtained by \mathbf{P}_4 satisfies $p^* \leq f(x_{\psi}^*) \leq p^* + \frac{3}{\psi}$.

Based on that, we analyze the problem in detail and provide a more practical approach for solving subsequent optimization problems. For the unconstrained problem \mathbf{P}_4 , Newton's method is employed to find the optimal solution. At the k -th iteration with the current value of ψ , denoted as x_k , and the optimal solution denoted as x_{ψ}^* , we define $\theta(x) = \Phi(x) + \psi \cdot f(x)$. Newton's method involves approximating the objective function using a quadratic function at the iteration point x_k and determining the minimum point of the quadratic function as the minimum point of the objective function. The second-order Taylor expansion of $\theta(x)$ at x_k is given by $\theta(x_k) + \nabla^T \theta(x_k)(x - x_k) + \frac{1}{2}(x - x_k)^T \nabla^2 \theta(x_k)(x - x_k) + o(\|x - x_k\|)^2$. Neglecting the infinitesimal term yields the approximation. Since $\theta(x)$ is quadratic, at the extremum point, $\nabla \theta(x_{\psi}^*) = 0$. Differentiating $\theta(x)$, we obtain $\nabla \theta(x) = \nabla \theta(x_k) + \nabla^2 \theta(x_k)(x - x_k) = 0$. Thus, $x = x_k - (\nabla^2 \theta(x_k))^{-1} \cdot \nabla \theta(x_k)$, setting $x = x_{k+1}$, and we get:

$$d_k^N = -(\nabla^2 \theta(x_k))^{-1} \cdot \nabla \theta(x_k), \quad (12)$$

where d_k^N is the Newton direction for the k -th iteration. In the Newton method, the iteration step is assumed to be 1 by default as soon as the iteration direction is determined. However, this direction does not necessarily have to be the direction of the decrease of the function value. This is verified by taking the dot product of the current iteration direction and the gradient. If the dot product is negative, it indicates that the iteration direction is the descending direction, i.e., $-\nabla^T \theta(x_k)(\nabla^2 \theta(x_k))^{-1} \nabla \theta(x_k)$. This condition only applies if the Hessian matrix $\nabla^2 \theta(x_k)$ is positive, which is a difficult condition to satisfy. Therefore, the concept of iteration steps is introduced. The linear search method

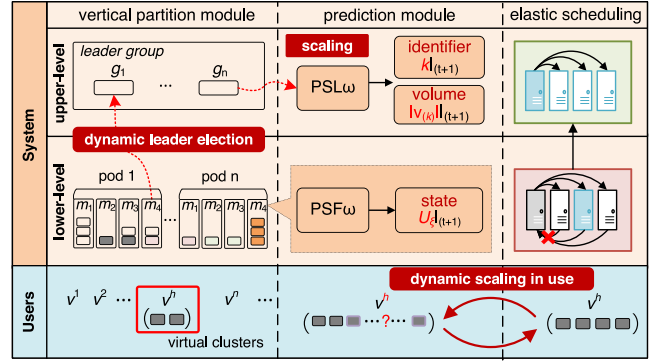


Fig. 2. The overview of EEP framework.

is used as follows:

$$\Delta x_k = \arg \min \theta(x_k + \Delta x_k \cdot d_k^N), \Delta x_k \in \mathbb{R}, \quad (13)$$

where $d\theta(x_k + \Delta x_k \cdot d_k^N)/d\Delta x_k = 0$ is solved directly during the solution process. The iteration ensures a descending direction, gradually approximating the optimal solution through successive steps. When $\|\nabla \theta(x_k)\| < \delta$, it indicates that the optimal point has been reached at the current ψ value.

V. THE EEP DESIGN

In this section, we outline the division of the energy-efficient and elastic resource provisioning process into three crucial steps, culminating in the construction of a comprehensive framework (EEP) as illustrated in Fig. 2. The framework comprises three key steps: vertical partition, prediction, and elastic scheduling. The initial step, vertical partition, utilizes a robust consensus mechanism to ensure fault-tolerant and distributed leadership. Following this, the prediction module employs committed horizon control methods, distinguishing the roles of leaders and followers. Leaders forecast the quantity and type of containers, while followers focus on predicting trends in container utilization on physical machines. The process concludes with elastic scheduling, where a distributed algorithm dynamically integrates resources, adapting to the evolving demands of the system. This strategic combination lays the groundwork for a responsive and adaptive resource provisioning paradigm.

A. Vertical Partition

This subsection details a novel vertical partitioning module enhanced by a leader election mechanism. The primary objective is to enhance communication and decision-making among physical machines within each pod. By electing a leader, the system ensures the consistency and reliability of the system. To increase the efficiency and accuracy of leader elections, we have developed a refined consensus algorithm based on the Raft mechanism. A key innovation is the introduction of a metric, \mathbf{Q} , which is a composite indicator integrating both energy consumption and Quality of Service (QoS). This metric is designed to more precisely assess the idleness of each physical machine and its suitability for leadership. The detailed steps of this approach are outlined in Algorithm 1, providing a clear

roadmap for implementation. First, we initialize the state of all physical machines $m_i \in M$ to the follower. We define $T(m_i)$ as the randomly generated election timeout. Afterward, when a follower m_i has not received a leader heartbeat within the $T(m_i)$, m_i initiates an election and waits for the $T(m_i)$ to expire in lines 2 to 4. Then, it transitions the state to a candidate $m_i^* \in M^*$ in lines 5 to 6. For each candidate m_i^* , we calculate the $\mathbf{Q}_{m_i^*}$ in line 8 and vote for itself in line 9. We also update $\mathbb{Z}(m_i^*)$ to plus one in line 10, assuming that $\mathbb{Z}(m_i^*)$ represents the number of votes, and simultaneously send a vote request with their $\mathbf{Q}_{m_i^*}$ to other physical machines in the pod in lines 11 to 12. In lines 13 to 21, physical machines m_h other than m_i^* determine whether to vote. Each m_h compares their \mathbf{Q} . If the $\mathbf{Q}_{m_i^*}$ obtained is smaller than the local \mathbf{Q}_{m_h} , they cast a vote in line 15 and $\mathbb{Z}(m_i^*)$ plus one in line 16; otherwise they abstain from voting in line 18. In line 19, we elect the candidate m_i^* with the most votes as the leader. If there is no candidate with the most votes in lines 20 to 23, the election timeout $T(m_i)$ is reset and a new election cycle begins, returning to line 4. This sequence of steps ensures that with dynamic workloads and system states, the node that is best suited to take the lead is elected.

The computational complexity of Algorithm 1 is determined to be $O(|M|^2)$, where $|M|$ denotes the total number of machines involved. Initially, transitioning all machines to the follower state requires $O(|M|)$ time. During the voting phase, the maximum number of iterations is $O(|M|)$, as each machine can participate in the process. In each round, each machine calculates the voting weight $\mathbf{Q}_{m_i^*}$ for its selected candidate m_i^* and updates its voting status, which requires $O(|M|)$ operations. Subsequently, the request to vote is broadcast to all machines, which then decide whether to vote or abstain based on specific conditions. This step has a time complexity of $O(|M|)$. Identifying the candidate with the highest number of votes also has a complexity of $O(|M|)$. If no candidate achieves the required number of votes, the election process may repeat itself, possibly up to $|M|$ times. Therefore, the overall time complexity for the algorithm is $O(|M|^2)$.

B. Prediction

This section introduces a prediction mechanism into the resource provisioning process, which is primarily driven by the dynamic nature of requirements on containers rented by multi-tenants. Influences such as workload fluctuations and changes in application requirements necessitate an adaptable approach to manage the variability in virtual cluster demands effectively. To adapt more flexibly to this uncertainty, we propose two ω -step prediction methods based on the committed horizon control method for lower-level physical machines (PSF $_{\omega}$) and for upper-level leaders (PSL $_{\omega}$). The main idea of PSF $_{\omega}$ is to use the predictive model to forecast the $\tilde{U}^{\xi}(t+1)$ of M in each time slot through multiple steps and utilize this information to optimize resource provisioning. The specific steps are shown in Algorithm 2. First, we assume that the period from 0 to Γ is the information storage phase, in which the data within this period serves as historical data to support the prediction. Therefore, the information from the previous τ steps is not available. In line 2, we initialize $\tilde{\mathbf{E}} = 1$, which represents the predicted

Algorithm 1: Vertical Partition Algorithm (VPA).

Input: $M, \mathcal{V}, \{U^{\xi}(m_i), P_f(m_i), P_l(m_i), T(m_i)\}_{m_i \in M}$;
Output: LeaderList \mathcal{D} in the upper level;
1: Initialize state of each $m_i \in M$ to follower;
2: **while** each m_i gets no heartbeat from leader in $T(m_i)$ **do**
3: Initiate an election;
4: Wait for the $T(m_i)$ to expire;
5: Convert $m_i \rightarrow m_i^*$;
6: Construct m_i^* into set M^* ;
7: **for** each m_i^* in M^* **do**
8: Calculate $\mathbf{Q}_{m_i^*}$;
9: Self vote to m_i^* ;
10: Update $\mathbb{Z}(m_i^*) = \mathbb{Z}(m_i^*) + 1$;
11: Update $M = M/m_i^*$;
12: Broadcast vote request to all $m_h \in M$ in the pod;
13: **for** each m_h in M **do**
14: **if** $\mathbf{Q}_{m_i^*} < \mathbf{Q}_{m_h}$ **then**
15: m_h vote to m_i^* ;
16: Update $\mathbb{Z}(m_i^*) = \mathbb{Z}(m_i^*) + 1$;
17: **else**
18: m_h abstain from voting;
19: Choose candidate $m_i^* = \arg \max_{m_i^* \in M^*} \{\mathbb{Z}(m_i^*)\}$;
20: **if** $|\mathbf{m}_i^*| > 1$ **then**
21: Reset election timeout T_{m_i} ;
22: Start a new election and go to line 5;
23: **Return D**;

elasticity value for the cloud data center. Following this, the state prediction is performed using an XGBoost regression model, utilizing $\tilde{U}^{\xi}[t, t + \omega]$, as detailed in line 4. Here, $\tilde{U}^{\xi}[t, t + \omega]$ denotes the predicted state content over a step length of ω from the time slot Γ . In line 5, we adjust the time index by setting $\tilde{t} = t - \Gamma$. From lines 6 to 12, we verify whether the predicted step is less than ω . If \tilde{t} is smaller than ω , we set $\tilde{U}^{\xi}|_{(t+1)}$ to the previously predicted content at the subsequent time step, $\tilde{U}^{\xi}(\tilde{t} + 1)$. If enough predictions with a step length of ω have already been accumulated, we update the content of the state prediction $\tilde{U}^{\xi}|_{(t+1)}$ in $A^{(\omega)}$ in line 9, where $A^{(\omega)}$ is the set of prediction contents. In line 10, we use the k -nearest neighbors method to classify the elements of the set $A^{(\omega)}$ into κ groups. In line 11, we choose the group for each physical machine by introducing a new factor for the occurrence of states, namely the potential frequency of states.

Definition 2 (Potential State Frequency): Let ϱ_h indicate the potential state frequency of a_h , where $\varrho_h = \frac{1}{\omega} \sum_{x=0}^{x=\omega-1} f(A^{(x)}, a_h)$.

Here, $f(A^{(x)}, a_h)$ is a function indicating whether \tilde{U}^{ξ} in set $A^{(x)}$ belongs to group a_h . We then select the group a_h by setting $a_h = \arg \max_{h \in 1, \kappa} \varrho_h$. Subsequently, we set $\tilde{U}^{\xi}|_{(t+1)}$ as the maximum element from group a_h in line 12. Finally, we return $\tilde{U}^{\xi}|_{(t+1)}$ of M . We demonstrate that the predicted values under dynamic updating conditions satisfy Lemma 1. The proof is shown as follows.

Algorithm 2: Prediction With ω Steps on Status of Followers (PSF $_{\omega}$).

Input: $M, U^{\xi}|_{(t)}$;
Output: State prediction $\tilde{U}^{\xi}|_{(t+1)}$ of M ;
 1: **for** $t = 0$ to $t = \Gamma$ **do**
 2: Set $\tilde{\mathbf{E}} = 1$;
 3: **for** $t = \Gamma$ to $t = T - 1$ **do**
 4: States prediction $\tilde{U}^{\xi}[t, t + \omega]$ based on XGBoost;
 5: Set $\tilde{t} = t - \Gamma$;
 6: **if** $\tilde{t} < \omega$ **then**
 7: Set $\tilde{U}^{\xi}|_{(t+1)} = \tilde{U}^{\xi}|_{(\tilde{t}+1)}$;
 8: **else**
 9: Update $\tilde{U}^{\xi}|_{(t+1)}$ into $A^{(\omega)}$;
 10: Classify the elements of set $A^{(\omega)}$ into κ groups;
 11: Choose group $a_h = \arg \max_{h \in \{1, \dots, \kappa\}} \{\varrho_h\}$;
 12: Set $\tilde{U}^{\xi}|_{(t+1)} = \arg \max_{\tilde{U}^{\xi}|_{(t+1)} \in a_h} \{\tilde{U}^{\xi}|_{(t+1)}\}$;
 13: **Return** $\tilde{U}^{\xi}|_{(t+1)}$ of M ;

Lemma 1: By applying PSF $_{\omega}$, the value under the updating status according to virtual clusters in use at time slot t , satisfies:

$$f(\tilde{x}) \leq p^* + 2\epsilon + \frac{2}{\omega} \alpha \cdot \sqrt{R} \cdot T. \quad (14)$$

Proof 1: We conduct the proof via introducing $\tilde{F}(t)$ to represent the objective value under the PSF $_{\omega}$ strategy, where $\tilde{F}(t) = f(\tilde{x})|_t$. In addition, we introduce $F(t)$ to represent the value under the decision policy with random frequency, where $F(t) = f(x)|_t$. We use $\delta(t)$ to denote the prediction error at time slot t , which ensures fluctuations under the value of ϵ . Then, we have the average value $\frac{1}{\omega} \sum_{t+1}^{t+\omega} \delta(t) \leq \frac{1}{\omega} \cdot \omega \cdot \epsilon = \epsilon$. Thus, we have

$$\tilde{F}(t) \leq \frac{1}{\omega} \sum_{t+1}^{t+\omega} F(t) \leq p^* + \frac{2}{\omega} \sum_{t+1}^{t+\omega} \delta(t) + \frac{2}{\omega} \alpha \cdot \sum_{t=1}^T \|x_t^a - x^*\|, \quad (15)$$

which can be obtained in [33], and $a = (t - \tau) \bmod \omega$. Here, $W = \max_{m_i \in \mathcal{M}} \{m_i^r - \tilde{m}_i^r\}$, which denotes the maximum available storage resource of physical machines. Since the physical resources of host machines allocated to provisioned virtual clusters within the data center are currently in use, the available resources for dynamic scaling requests should be within the constraint of the remaining capacity of the servers in the data center, where $\|x_t^a - x^*\| \leq \sqrt{W}$. Equation (15) can be converted to

$$\tilde{f}(t) \leq p^* + 2\epsilon + \frac{2}{\omega} \alpha \cdot \sqrt{W} \cdot T, \quad (16)$$

Therefore, the proof of Lemma 1 is complete. ■

Another critical consideration is the possibility of virtual clusters in use requiring dynamic scaling entering the system. The influx of new requests may trigger resource reallocation and migrations. Without a robust prediction mechanism, unplanned container migrations could result in inefficient resource allocation and subsequently increased energy consumption. As highlighted in the challenges described in the first section, unanticipated container migrations may contribute to wasted

energy and performance degradation. Therefore, we propose a ω -step prediction strategy based on the committed horizon control method that prevents unnecessary migrations and energy consumption. The individual steps are shown in Algorithm 3. Similar to the concept of Algorithm 2 is the use of the prediction model, which predicts the number and identifier of the next time slot in several steps and uses this information to optimize resource provisioning. The difference lies in lines 12 to 17, where classification is no longer performed. Instead, the output is determined by selecting the prediction result with the highest probability.

Algorithm 3: Prediction With ω Steps on Status of Leaders (PSL $_{\omega}$).

Input: $\mathcal{V}, h|_{(t)}, N(h)|_{(t)}$;
Output: Scaling prediction $\tilde{h}|_{(t+1)}, N(\tilde{h})|_{(t+1)}$ of \mathcal{V} ;
 1: **for** $t = 0$ to $t = \tau$ **do**
 2: Initialize $N(\tilde{h}) = 0$ and $\tilde{h} = \text{None}$;
 3: **for** $t = \tau$ to $t = T - 1$ **do**
 4: Scaling information prediction of $\tilde{h}[t, t + \omega]$,
 $N(\tilde{h})[t, t + \omega]$ based on XGBoost;
 5: Set $\tilde{t} = t - \tau$;
 6: **if** $\tilde{t} < \omega$ **then**
 7: Set $\tilde{h}|_{(t+1)} = \tilde{h}|_{(\tilde{t}+1)}$;
 8: Set $N(\tilde{h})|_{(t+1)} = N(\tilde{h})|_{(\tilde{t}+1)}$;
 9: **else**
 10: Update $\tilde{h}|_{(t+1)}$ into $B^{(\omega)}$;
 11: Update $N(\tilde{h})|_{(t+1)}$ into $C^{(\omega)}$;
 12: Update the probability p_i of $N(\tilde{h})|_{(t+1)}$ into $\mathbf{N}^{(p)}$;
 13: Update the probability p_n of $\tilde{h}|_{(t+1)}$ into $\mathbf{Y}^{(p)}$;
 14: Set $p_i = \arg \max_{p_i \in \mathbf{N}^{(p)}} \{p_i\}$;
 15: Set $p_n = \arg \max_{p_n \in \mathbf{Y}^{(p)}} \{p_n\}$;
 16: Choose $N(\tilde{h})|_{(t+1)}$ with p_i ;
 17: Choose $\tilde{h}|_{(t+1)}$ with p_n ;
 18: **Return** $\tilde{h}|_{(t+1)}$ and $N(\tilde{h})|_{(t+1)}$ of \mathcal{V} ;

Algorithms 2 and 3 share identical computational complexities; thus, the following analysis applies equally to both. Specifically, the time complexity of Algorithm 2 is $O(T \cdot |M| \cdot \log |M|)$, where T denotes the total number of time steps and $|M|$ represents the number of machines involved. In the initialization phase, the algorithm executes a loop from $t = 0$ to $t = \Gamma$, incurring a computational cost of $O(\Gamma)$. This is followed by the state prediction phase, which iterates from $t = \Gamma$ to $t = T - 1$, contributing an additional complexity of $O(T - \Gamma)$. Within each iteration of this phase, the XGBoost-based state prediction step demands $O(|M| \cdot \log |M|)$ time, attributable to the sorting and tree traversal operations inherent in the algorithm. Subsequent conditional evaluations and state updates are executed in constant time, amounting to $O(1)$ per iteration. During the element classification and group selection stage, the time complexity is $O(|M| \cdot \kappa)$, where κ denotes the number of groups. The final step of returning results also operates in constant time. Therefore, the overall time complexity of the algorithm can be expressed as $O(\Gamma) + O((T - \Gamma) \cdot |M| \cdot \log(|M|)) + O(|M| \cdot \kappa)$. Even in the worst-case scenario, where both T and Γ are

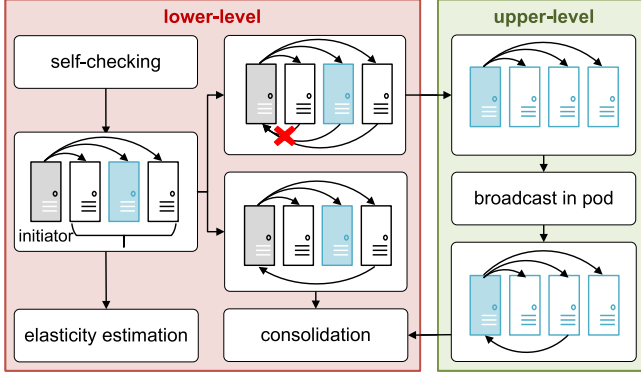


Fig. 3. Elastic scheduling step by step.

large, the overall time complexity remains primarily dominated by the prediction component. Therefore, the complexity can be approximated as $O(T \cdot |M| \cdot \log(|M|))$.

C. Scheduling

In this section, the focus is on optimizing overall consumption by introducing an elastic scheduling strategy, with the main steps depicted in Fig. 3. An innovative dynamic distributed approach has been developed to optimize resource utilization and reduce energy consumption. This approach dynamically adjusts the provisioning of virtual clusters for multi-tenants based on their current low utilization states, resource requirements, and the status of the data center network. The main idea is that any physical machine that fulfills the conditions for resource integration can initiate the consolidation request, and can try to search for a suitable provisioning location by broadcasting the request to other physical machines within the pod. In the event that the request is not accepted by the physical machines in the pod, it is then forwarded to the pod leaders and disseminated to the group, facilitating cross-pod consolidation.

The specific steps are shown in Algorithms 4 and 5. The information obtained during the prediction phase and the results of \mathbf{P}_4 are required. Our consolidation approach involves identifying underutilized physical machines and designating them as potential migration sources. We then consider consolidating containers residing on these underutilized physical machines to target machines with higher resource utilization without exceeding the optimal utilization rate. We first consider a scenario where containers can be consolidated within a pod, as outlined in Algorithm 4. The inputs are the current and predicted utilization of m_i , which are $U^\xi(m_i)$ and $\tilde{U}^\xi(m_i)$, respectively. Furthermore, the scaling information $|\tilde{v}^k(t+1)|$ that the number of the containers in the next time slot of virtual cluster k is required as the input. The output is the scheduling strategy \mathbf{X} . We calculate the elastic tendency $\hat{\mathbf{E}}_{m_i}$ for each physical machine m_i in M in line 2, and the definition of $\hat{\mathbf{E}}_{m_i}$ is shown as follows.

Definition 3 (Elastic Tendency): Let $\hat{\mathbf{E}}_{m_i}$ indicate the elastic tendency of the physical machine m_i , where $\hat{\mathbf{E}}_{m_i} = \min\{\mathbf{E}_{m_i}, \tilde{\mathbf{E}}_{m_i}\}$.

Here, \mathbf{E}_{m_i} and $\tilde{\mathbf{E}}_{m_i}$ are the current and predicted elasticity values, respectively. We assume ε is the elasticity upper bound for physical machines. If $\hat{\mathbf{E}}_{m_i} > \varepsilon$, which means that the elasticity value $\hat{\mathbf{E}}_{m_i}$ of a physical machine m_i is greater than the threshold ε , m_i is in a state of extremely lower utilization but still consumes energy. In such cases, the physical machine needs to initiate an integration request $s^{(m_i)}$ to reduce the energy consumption of the physical machine in line 5. Here, we introduce two new definitions of the initiate consolidation set \mathbf{W} and wait state set $\overline{\mathbf{W}}$.

Definition 4 (Initiate Consolidation Set): Let \mathbf{W} indicate the initiate consolidation set which is constructed by the physical machine m_i with a state of extremely lower utilization ($\hat{\mathbf{E}}_{m_i} > \varepsilon$) and with no demand for scaling.

Definition 5 (Wait State Set): Let $\overline{\mathbf{W}}$ indicate the wait state set which is constructed by the physical machine m_i as being either in a state under normal load ($\hat{\mathbf{E}}_{m_i} \leq \varepsilon$) or preparing to be utilized.

Algorithm 4: Elastic Scheduling Strategy (Lower-Level).

Require: $U^\xi|_{(t)}$, $\tilde{U}^\xi|_{(t+1)}$, $N(\tilde{h})|_{(t+1)}$, $\mathbf{E}_{m_i}^*$

Ensure: \mathbf{X}

- 1: **for** each $m_i \in M$ **do**
 - 2: Calculate $\hat{\mathbf{E}}_{m_i}$;
 - 3: **if** $\hat{\mathbf{E}}_{m_i} > \varepsilon$ and no scaling **then**
 - 4: Update m_i into set \mathbf{W} ;
 - 5: Initiate consolidation request $s^{(m_i)}$ and broadcast to physical machines in pod;
 - 6: **else**
 - 7: Update m_i into set $\overline{\mathbf{W}}$;
 - 8: Calculate scalability factor $\Omega(m_i)$;
 - 9: **if** $\Omega(m_i) \leq 0$ **then**
 - 10: Reply with reject;
 - 11: **else**
 - 12: Reply with accepting;
 - 13: Update m_i into set \mathbf{N} ;
 - 14: **for** m_a in \mathbf{N} **do**
 - 15: **if** $x = y|_{\{v_x \in m_i, v_y \in m_a\}}$ **then**
 - 16: consolidate m_i on m_a ;
 - 17: **else**
 - 18: priority with highest scalability factor $\Omega(m_i)$;
 - 19: **Return** \mathbf{X}
-

In lines 7 to 13, the process of determining which physical machines can accept the consolidation requests begins. If $\hat{\mathbf{E}}_{m_i} < \varepsilon$, indicating insufficient elasticity, m_h is moved into a wait state set denoted as $\overline{\mathbf{W}}$. Within this set, for each m_h , the scalability factor $\Omega(m_i)$ is employed in line 8 to assess the updating state of the physical machine. This scalability factor is crucial in determining the capacity of m_h to adapt to additional load without compromising performance, and is defined as follows:

Definition 6 (Scalability Factor): The scalability factor of the physical machine m_i , denoted by $\Omega(m_i)$, signifies its scaling capability when its current elasticity value exceeds that of the optimal location $\mathbf{E}_{m_i}^*$, where $\Omega(m_i) = \min\{\mathbf{E}_{m_i} - \mathbf{E}_{m_i}^*, 0\}$.

If $\Omega(m_i) \leq 0$, it indicates that the physical machine has reached or exceeded the optimal utilization rate and cannot accommodate any more containers, resulting in the rejection of the response in line 10. Otherwise, we update m_i to \mathbf{N} and respond with acceptance in lines 12 to 13. Subsequently, in lines 14 to 18, m_i selects the suitable physical machine to receive $s^{(m_i)}$ and make consolidation decisions \mathbf{X} . If the cluster type of the container v^y on the target physical machine m_a matches that of v^x on the initiating physical machine m_i , then consolidate v^x into m_a in lines 15 to 17 that helps minimize communication costs. Otherwise, we place the container in the physical machine with the maximum $\Omega(m_i)$, aiming to satisfy optimal elasticity $\mathbf{E}_{m_i}^*$ as much as possible. Finally, we return the scheduling decision \mathbf{X} in line 19.

The time complexity of Algorithm 4 is $O(|M| + |\mathbf{N}|)$. Initially, the algorithm iterates over the set of machines M , which has a complexity of $O(|M|)$. During each iteration, the calculations for $\hat{\mathbf{E}}_{m_i}$ and $\Omega(m_i)$ are performed in constant time, resulting in $O(1)$ complexity for these operations. If specific conditions are met, the algorithm updates m_i into different sets or initiates a merge request, which is then broadcast to the physical machines, and this update also occurs in $O(1)$ time. The next step involves a for loop that iterates over the set \mathbf{N} , with a time complexity of $O(|\mathbf{N}|)$. During the merging phase, the algorithm prioritizes machines based on their scalability factor, which may involve sorting or comparing elements within set \mathbf{N} . In the worst-case scenario, this operation has a complexity of $O(|\mathbf{N}|)$, as it requires selecting the machine with the highest scalability factor. Overall, the time complexity is driven by these two main loops over sets M and \mathbf{N} . As a result, the total time complexity is $O(|M| + |\mathbf{N}|)$, reflecting the linear relationship with the sizes of these sets.

Algorithm 5: Elastic Scheduling Strategy (Upper-Level).

Require: $U^\xi|_{(t)}, \tilde{U}^\xi|_{(t+1)}, \tilde{h}, N(\tilde{h})|_{(t+1)}$
Ensure: \mathbf{X}
 1: **if** $\forall m_h \in \overline{\mathbf{W}}$ reject $s^{(m_i)}$ **then**
 2: m_i submit request $s^{(m_i)}$ to pod leader g_n ;
 3: Update $\mathbf{g} = \{\mathbf{g}/g_n\}$;
 4: Leader g_n broadcast $s^{(m_i)}$ to leader group \mathbf{g} ;
 5: **for** each g_l in \mathbf{g} **do**
 6: Broadcast $s^{(m_i)}$ within the pod;
 7: Same as Algorithm 4 in lines 9-13;
 8: Update m_a into \mathbf{N} ;
 9: Send the information of \mathbf{N} to g_n ;
 10: **for** each m_a in \mathbf{N} **do**
 11: Same as Algorithm 4 in lines 15-18;
 12: **Return** \mathbf{X}

The second scenario involves situations where there is no suitable consolidation solution within the pod where the physical machine requesting the consolidation is located. Once this scenario occurs, it indicates that the workload that the consolidation initiator needs to integrate must be integrated into the physical machines in other pods, and the specific procedure is shown in Algorithm 5. We suppose that the initiator m_i belongs to

pod n . If all physical machines m_h in $\overline{\mathbf{W}}$ that belong to pod n refuse to accept the workload, m_i that initiated the request sends $s^{(m_i)}$ to the pod leader g_n in line 2. Then, we remove g_n from the set \mathbf{g} in line 3, where \mathbf{g} represents the group of leaders for all pods. Leader g_n broadcasts $s^{(m_i)}$ to group \mathbf{g} . After converting the cross-pod request into a regular intra-pod request, each pod leader $g_l \in \mathbf{g}$ takes on the role of $s^{(m_i)}$ initiator. It then broadcasts that $s^{(m_i)}$ within its pod on line 6. Consequently, we move on to in-group consolidation, where the consolidation strategy is the same as in Algorithm 4 in lines 9 to 13. Each pod leader g_l sends the information of \mathbf{N} to g_n . In lines 10 to 12, the leader of g_n selects a suitable physical machine to receive $s^{(m_i)}$ in the same way that Algorithm 4 in lines 15 to 18 does. Finally, we return the scheduling decision \mathbf{X} .

The time complexity of Algorithm 5 is $O(|\overline{\mathbf{W}}| + |\mathbf{g}| \cdot |\mathbf{N}|)$. Initially, the algorithm performs a condition check in line 1, where it iterates over all machines in the set $\overline{\mathbf{W}}$. This step has a complexity of $O(|\overline{\mathbf{W}}|)$, where $|\overline{\mathbf{W}}|$ denotes the number of machines in the set. If the condition is met, the algorithm submits a request to the pod leader and updates the set \mathbf{g} , both of which are constant-time operations with a complexity of $O(1)$. Next, the algorithm iterates through all leaders within the group \mathbf{g} , with a complexity of $O(|\mathbf{g}|)$. Each leader broadcasts the request and executes steps similar to those in Algorithm 4, specifically handling the scalability factor $\Omega(m_i)$ of each machine. As in Algorithm 4, these steps have a complexity of $O(|\mathbf{N}|)$, where $|\mathbf{N}|$ is the number of machines involved in the expansion process. Combining these components, the overall time complexity of Algorithm 5 is $O(|\overline{\mathbf{W}}| + |\mathbf{g}| \cdot |\mathbf{N}|)$, reflecting the linear growth in relation to the sizes of sets $\overline{\mathbf{W}}$ and \mathbf{g} , as well as the number of machines $|\mathbf{N}|$ involved. This ensures that the algorithm scales efficiently with the number of machines and leaders it processes.

VI. EVALUATIONS

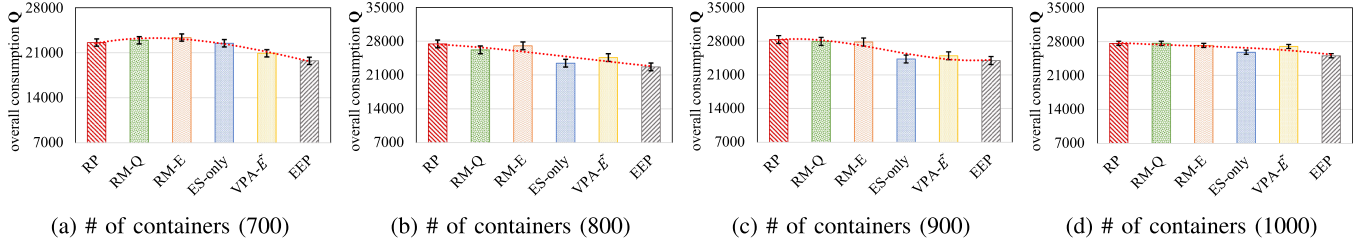
This section details extensive simulations and experiments conducted to explore energy-efficient and elastic resource management within multi-tenant cloud data centers. Initially, the datasets and experimental settings are described, setting the stage for a rigorous analysis. Subsequently, results are presented from multiple perspectives, providing a comprehensive overview of the performance outcomes. These results offer valuable insights and enable the drawing of meaningful conclusions regarding the effectiveness of the proposed strategies.

A. Basic Setting

Our prototype was conducted using Python 3.7 on Windows 10 with an Intel(R) Xeon(R) Silver 4210R CPU, NVIDIA RTX5000 GPU, 128 GB memory, and a 2 TB hard disk. We simulate a fat-tree data center network with 4-port switches and 16 physical machines. Each physical machine has CPU resources of 2.40 GHz, memory resources of 100 GB, 16 cores, and a bandwidth resource range in [3, 7]GB. In our basic setting, we consider deployments and workload conditions based on observations from analyzing public data from GWA-T12 Bitbrains [32]. Specifically, we select 25 tenants, each comprising 4 containers, with the required cores for these containers

TABLE II
 OVERALL CONSUMPTION ACROSS DIFFERENT STRATEGIES UNDER THE 4-PORTS TOPOLOGY OF DIFFERENT ε

$Q(\varepsilon)$	RP	RM-Q	RM-E	ES-only	VPA- \tilde{E}	EEP	Avg
0.2	1574.08 \pm 15.30	1524.46 \pm 112.8	1586.12 \pm 28.75	1437.17 \pm 69.57	1381.17 \pm 206.5	1354.07 \pm 75.16	1476.18
0.3	1611.08 \pm 158.5	1551.68 \pm 45.61	1429.95 \pm 33.26	1374.13 \pm 90.04	1569.11 \pm 128	1291.15 \pm 105.4	1471.18
0.5	1619.44 \pm 127.3	1562.35 \pm 75.6	1486.31 \pm 109	1565.13 \pm 59.07	1586.43 \pm 86.34	1328.13 \pm 89.15	1524.63


 Fig. 4. Overall consumption across different strategies under the 8-ports topology with $\varepsilon = 0.5$.

varying between $[1, 4]$. Subsequently, we conduct tests using various groups of parameters, including scenarios where scaling requests are 10, 20, and 30, and the value of ε ranges from 0.1 to 0.9. To further assess scalability and performance across different data center scales and user request conditions, we also conduct experiments under larger configurations with $\varepsilon = 0.5$. Specifically, we test scenarios with 8-port switches and 128 physical machines, hosting 100 tenants with 8 containers each under the condition where $\varepsilon = 0.5$. We compare our proposed framework with five baselines (RP, RM-Q, RM-E, ES-only, VPA- \tilde{E}).

- **RP**: virtual clusters from multiple tenants are randomly distributed across the cloud data center during the initial provisioning phase, with resources allocated at random locations on physical machines and without any prediction or scheduling.
- **RM-Q**: virtual clusters are provisioned without predictive analytics. Resource scaling decisions are based solely on Q values, with real-time adjustments during scaling requests. The Q values are ranked and selected following the method outlined in [31], ensuring resource allocation aligns with the most critical scaling needs.
- **RM-E**: this strategy provisions virtual clusters without prediction, with scaling decisions based solely on E values to adjust resources during scaling requests.
- **ES-only**: virtual clusters are provisioned without comprehensive initial placement or predictive information, with scaling decisions driven entirely by an elastic scheduling strategy focused on real-time resource adjustments [36].
- **VPA- \tilde{E}** : utilizes a distributed virtual partitioning method enhanced by predictive analytics. Virtual clusters are not fully placed initially; instead, resource scheduling is based on predicted elasticity values \tilde{E} , allowing for strategic resource allocation from the beginning.

B. Experiment Results

We investigate the overall consumption under five strategies with three groups of tenants. The results are shown in Table II

and Fig. 4. We derive the following key observations with Q under different elasticity upper bounds ε . (i) Impact of different strategies: For each algorithm, we computed the average Q across various initial states of the data centers, beginning with 70 to 90 containers and then adding 30 to 10 containers, randomly selected from different virtual clusters in the 4-port topology, while maintaining a consistent elasticity parameter ε . We also calculated the standard deviation to capture the variability across these configurations. Table II (4-port topology) shows the performance of different strategies (RP, RM-Q, RM-E, ES-only, VPA- \tilde{E} , EEP) in terms of overall consumption for different ε values and container counts. The EEP strategy consistently achieves the lowest consumption across all ε values, particularly at $\varepsilon = 0.3$, where it records 1291.15 \pm 105.4. This indicates that EEP is highly adaptable to changes in ε , while other models exhibit erratic behavior likely due to the randomness in placement decisions, leading to suboptimal resource utilization and variations in Q . The RP algorithm shows inconsistent trends with fluctuating Q values when the initial number of running containers varies across tenants. Similarly, RM-Q and RM-E also display fluctuations and, in some cases, an increase in Q , reflecting their limitations in dynamically adapting to changing workloads. These approaches lack elasticity prediction, which hinders their ability to make proactive scaling decisions. In contrast, the EEP algorithm consistently outperforms other methods. By incorporating elasticity prediction, EEP anticipates resource needs, leading to lower Q values and more effective scaling. Its proactive leader election and dynamic consolidation further enhance its ability to handle dynamic workloads efficiently.

For the 8-port topology, we evaluated performance with 700 to 1000 containers distributed across 100 virtual clusters. As illustrated in Fig. 4, EEP consistently demonstrates lower energy consumption compared to other strategies, even as the number of containers scales up. For example, in Fig. 4(d) with 1000 containers, EEP demonstrates one of the lowest consumption values. Across all container count scenarios (Fig. 4(a) to (d)), EEP maintains a stable consumption pattern. In contrast, strategies such as RP and VPA- \tilde{E} show a more significant increase in energy consumption as both the number of containers and

the complexity of the network increase. The consistent performance across both topologies highlights how effectively EEP manages increasing workloads while reducing energy consumption, demonstrating its robustness in both simple and complex network configurations.

(ii) Impact of topologies: We conducted experiments in data centers with different network topologies, specifically 4-port and 8-port configurations. The results show different consumption trends depending on the network configuration. In the 4-port topology (Table II), overall consumption across all strategies is lower than in the 8-port topology (Fig. 4(a) to (d)), as expected due to the simpler network structure and fewer container requests. However, EEP consistently demonstrates superior energy efficiency in both topologies, maintaining lower consumption regardless of the network complexity. For instance, in the 4-port topology with $\varepsilon = 0.5$ (Table II), EEP achieves significantly lower consumption at $1328.13_{\pm 89.15}$ and thus outperforms RP ($1619.44_{\pm 127.3}$) and VPA- \tilde{E} ($1586.43_{\pm 86.34}$). This trend continues in the 8-port topology, where EEP remains the most energy-efficient strategy, especially as the number of containers increases. In Fig. 4(d), EEP with 1000 containers still demonstrates lower consumption compared to RP and VPA- \tilde{E} , which experience significant increases in consumption as the network complexity and container count rise. Across all container count scenarios (Fig. 4(a) to (d)), EEP maintains a stable consumption pattern. In contrast, strategies such as RP and VPA- \tilde{E} show a more significant increase in energy consumption as both the number of containers and the complexity of the network increase. The consistent performance across both topologies highlights how effectively EEP manages increasing workloads while reducing energy consumption, demonstrating its robustness in both simple and complex network configurations.

(iii) Impact of ε : We summarized the results of 9 sets of \mathbf{Q} for different initial placement strategies under various ε for the six algorithms in Table II. As ε increased from 0.2 to 0.5, the \mathbf{Q} value of EEP demonstrated a notable downward trend (from 1354.07 to 1291.15). Although there is an increase in the \mathbf{Q} at $\varepsilon = 0.5$, it is related to the initial placement positions. Inappropriate initial placement positions can cause multiple reschedules. This indicates EEP's proficiency in handling elastic scaling scenarios and aligning resource allocation more efficiently with evolving demands. In contrast, RP consistently yielded higher \mathbf{Q} , underscoring the inefficiency of random placement strategies. Predictive capabilities played a crucial role, with VPA- \tilde{E} showcasing lower \mathbf{Q} at $\varepsilon = 0.2$ emphasizing the importance of predictive modeling. ES-only performs well, with elastic scheduling playing a crucial role. The fluctuation in \mathbf{Q} for RM- \mathbf{Q} and RM- \mathbf{E} highlights the limitations of relying solely on \mathbf{Q} or \mathbf{E} without prediction and elastic scheduling.

We also investigated the overall consumption and consolidation times across nine different elasticity upper bounds ε , ranging from 0.1 to 0.9 in intervals of 0.1. The results are depicted in Fig. 5. Several key observations emerged from the analysis. First, a distinct trend in \mathbf{Q} and migration numbers is evident as ε varies. For ε values between 0.1 and 0.5, \mathbf{Q} shows a decreasing trend, indicating improved efficiency and resource utilization. However,

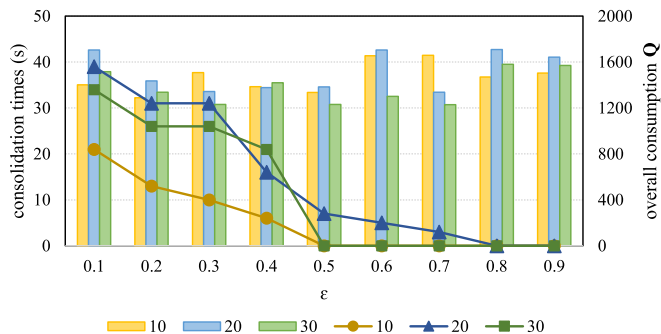


Fig. 5. \mathbf{Q} under different ε and consolidation times of EEP.

for ε values from 0.5 to 0.9, \mathbf{Q} begins to increase. This suggests that $\varepsilon = 0.5$ acts as a relatively optimal upper bound. A low ε can lead to more frequent integrations and higher migration costs, while a high ε results in fewer integrations, increasing energy consumption and communication costs. Second, consolidation times are also affected by ε . As ε increases, the number of migrations decreases, reaching zero at values above 0.5. For example, with 30 scaling requests, $\varepsilon = 0.5$ resulted in zero migrations and a \mathbf{Q} of 1230.14. Furthermore, the consolidation times are not only influenced by ε , but also by the level of initial placement. As the initial number of running containers increases, the ε value at which zero migrations occur shifts. With 70 initial containers scaling by 30, zero migrations are observed at $\varepsilon = 0.5$. However, with 90 initial containers scaling by 10, zero migrations occur at $\varepsilon = 0.4$. This variability is likely due to the randomness in the initial distribution of containers, which affects the ability of ε to optimize the scheduling process without requiring migrations. This highlights a complex interplay between ε , initial container placement, and consolidation times.

C. Evaluations on Cloud Testbed

Testbed Configuration: The experimental environment is anchored by a Huawei Cloud c7.4xlarge.4, serving as our primary information server. A cloud data center infrastructure is constructed, comprising 8 servers organized into two pods, with each pod housing four servers. Specifically, Pod 1 includes servers m_1, m_2, m_3, m_4 and Pod 2 consists of m_5, m_6, m_7, m_8 . Each server within the setup is provisioned with 16 virtual CPUs (vCPUs) and 64 GiB of memory, ensuring robust computational and data handling capabilities. Furthermore, the configuration includes 10 virtual clusters, designed to address diverse operational requirements and workload distributions. The allocation and detailed specifications of these virtual clusters within the data center are comprehensively outlined in Table III, offering a clear overview of the resource distribution and utilization strategies employed in the testbed setup.

Testbed Results: We investigated the overall consumption under three different strategies: RP, RM- \mathbf{Q} , and EEP with different utilization levels. Each strategy was sampled every 30 seconds and the examination spanned 300 seconds. The results are shown in Fig. 6. We have the following observations: (i) EEP consistently maintains lower \mathbf{Q} values. As shown in Fig. 6(a), we

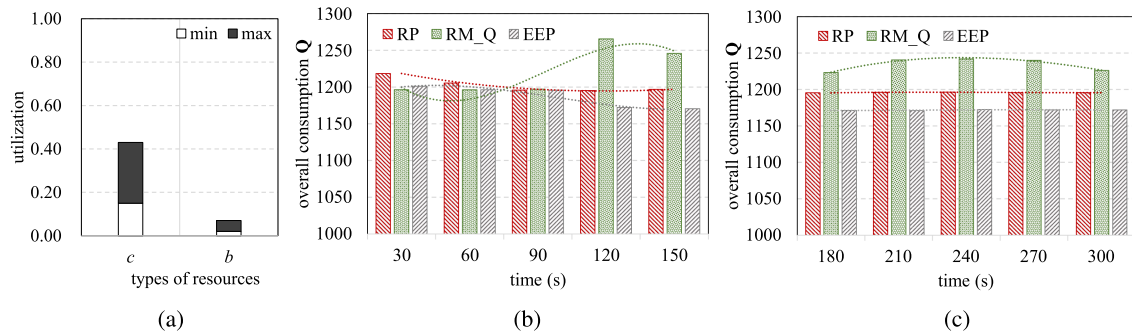


Fig. 6. Overall consumption with lower initial utilization on a real cloud test-bed: (a) shows modest bandwidth (b) and CPU (c) usage; (b) and (c) illustrate stable consumption over 30 to 300 seconds with minor fluctuations.

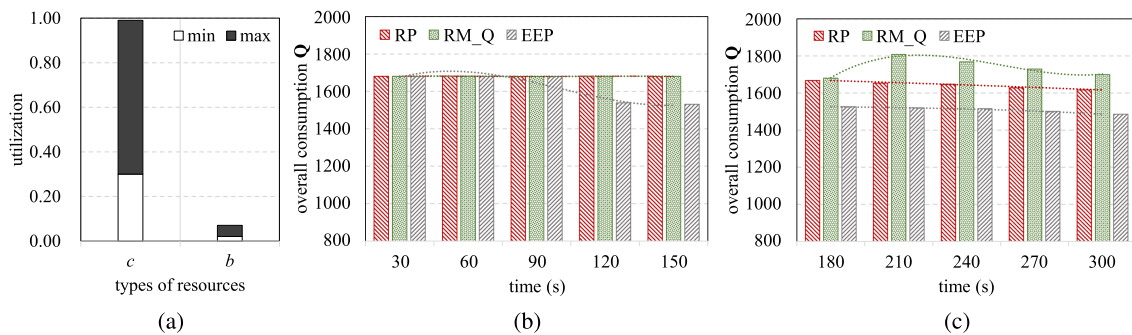


Fig. 7. Overall consumption with higher initial utilization on a real cloud test-bed: (a) depicts elevated bandwidth (b) and CPU (c) usage; (b) and (c) demonstrate variable consumption over 30 to 300 seconds with notable fluctuations.

TABLE III
INITIAL PLACEMENT OF VIRTUAL CLUSTERS WITHIN THE HUAWEI CLOUD TESTBED, ILLUSTRATING THE DISTRIBUTION ACROSS DIFFERENT SERVERS WITHIN EACH POD

pod 1				pod 2			
m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8
-	-	-	-	-	-	-	-
-	-	-	v_8^1	-	-	-	-
v_2^1	v_4^1	-	v_7^1	-	-	-	-
v_1^1	v_3^1	v_5^1	v_6^1	v_9^1	v_5^2	v_{10}^1	v_{10}^2

first discuss the overall consumption at low server utilization due to processing non-computationally intensive tasks. The results are shown in Fig. 6(b) and (c). Compared to RP and RM-Q, EEP demonstrates excellent overall performance, while RM-Q performs poorly, exhibiting a lower Q value than the initial provisioning of RP. The reason for these results is that the RM-Q method focuses solely on reducing the utilization of physical machines by putting them into low-power states while ignoring migration costs. In the RM-Q, containers are consolidated on physical machines to keep them idle as much as possible. For example, containers from m_6 , m_7 , and m_8 in Pod 2 are migrated to m_5 , and the container from m_3 in Pod 1 is migrated to m_1 . This approach reduces the energy consumption of physical machines but introduces ineffective migrations, leading to increased migration costs. Additionally, communication costs

among containers still exist after migration. EEP not only considers prediction but also prioritizes integration within the same virtual cluster during deployment. For example, the container on m_7 belongs to the same cluster as the container on m_8 . Through inter-group container integration, the containers on m_7 are integrated into m_8 . Similarly, the container on the physical machine m_6 belongs to the same cluster as the container on m_3 . In this case, the container on m_6 can be integrated into m_3 through inter-group container integration. By employing this method, EEP minimizes communication costs as much as possible. Compared to the five integrations of RM-Q, EEP integrates only twice. Therefore, EEP effectively reduces the costs compared to RM-Q.

(ii) With increasing utilization of the physical machines, EEP performs better, as shown in Fig. 7(b) and (c). On the cloud experimentation platform, server utilization is elevated by executing computationally intensive tasks, as shown in Fig. 7(a). When the utilization is low, the difference between EEP and RP is small, except for the worse performance of RM-Q. This is due to the lower utilization, which results in less communication between the containers. Consequently, the reduced communication overhead counteracts the migration costs incurred by the consolidation. Therefore, in such cases, a rational migration strategy is required instead of blind integration that could increase consumption. When server utilization is high, a significant amount of communication takes place between containers. By integrating containers within the same cluster,

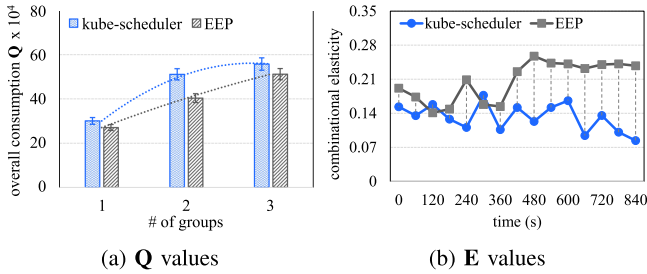


Fig. 8. Overall consumption (Q) and elasticity (E).

the communication overhead and energy consumption of the physical machine can be effectively reduced, thereby reducing the Q value and optimizing the effect.

D. Evaluations on Kubernetes

Testbed Configuration: To further validate the effectiveness of EEP, we implemented a prototype on Kubernetes. The experimental setup utilized Python 3.11-slim on CentOS 7.6, simulating a fat-tree data center network with 2 port switches and 4 physical machines. The deployment involved a total of 5 nodes, comprising 4 worker nodes and 1 master node responsible for managing and orchestrating container operations. Four of the machines were configured as worker nodes, each with Intel(R) Xeon(R) Gold 6348 CPU, 16 GB memory, 8 cores, and bandwidth resources between 3 and 15 GB. The controller was configured as a master node and equipped with an Intel(R) Xeon(R) Gold 6266 C CPU, 4 GB of memory, 2 cores, and bandwidth resources of 1.2 and 4 GB. To reflect the operating conditions of the real system, we gradually increased the workload for a randomly chosen half of the containers every 60 seconds to simulate simultaneous demand for services. During the experiment, we recorded the utilization of the nodes in response to the variations in tenant requests and the communication volume between the nodes related to the execution of containers from virtual clusters. We select the default scheduling strategy kube-scheduler as a baseline for the comparative analysis [35].

Testbed Results: Three groups were randomly selected for comparison under time-series data collection. As illustrated in Fig. 8(a), we compared the overall consumption (Q) between the kube-scheduler and the proposed EEP framework. The experimental results demonstrate that EEP achieves lower overall consumption compared to the kube-scheduler. During the scheduling process, EEP involved four container migrations following the initial placement, whereas Kubernetes performed none. Although these migrations introduce some additional overhead, they allow EEP to adapt dynamically to changing system conditions, ultimately leading to optimized container placement. Therefore, the superior performance of EEP is primarily attributed to its adaptive placement and migration strategy, which contrasts significantly with the kube-scheduler as observed in the results. During scheduling and placement, the kube-scheduler does not account for the physical proximity of containers belonging to the same tenant. This limitation results in containers from the same user request being distributed

across distant nodes. This leads to excessive inter-container communication and subsequently increases the overall Q value due to heightened communication costs across nodes. In contrast, EEP optimizes container placement by strategically migrating containers, reducing cross-node traffic, and enhancing communication efficiency. The adaptive migration mechanism employed by EEP ensures dynamic responsiveness to fluctuating system conditions, effectively balancing container placement with network efficiency. As a result, EEP not only achieves lower overall consumption but also maintains superior communication efficiency, particularly under varying workload demands.

Furthermore, we analyzed elasticity to evaluate the scaling capabilities of both strategies over time, as shown in Fig. 8(b). The EEP method consistently outperforms the kube-scheduler in terms of the E value, albeit with some initial fluctuations. During the first 120 seconds of the scheduling process, the E value for EEP is slightly lower than that of the kube-scheduler at one or two time points. This behavior is attributed to the overhead from initial migrations required for planning and the instantaneous elasticity adjustments during the system's dynamic adaptation. However, as the process continues, EEP's elasticity value stabilizes and consistently exceeds that of the kube-scheduler, which continues to exhibit significant fluctuations. This trend highlights EEP's dynamic adaptability in resource allocation and container consolidation, enabling it to maintain superior elasticity and system stability compared to the static scheduling approach of the kube-scheduler. Consequently, EEP achieves superior energy efficiency by dynamically balancing the load and consolidating resources, effectively reducing the number of active servers and optimizing energy consumption. In contrast to the kube-scheduler's static scheduling approach, EEP leverages adaptive scheduling and controlled migrations, significantly improving overall consumption while maintaining elasticity at a consistently high level, ensuring efficient resource utilization and minimizing energy consumption.

VII. CONCLUSION

This paper focuses on solving the problem of resource scheduling and efficient integration for multiple tenants in cloud data centers, and the objective is to minimize energy consumption by jointly considering combinational elasticity and QoS. We explore the optimization method for energy-efficient and elastic resource provisioning with QoS guarantee by proposing a distributed framework EEP. The method of energy-efficient and elastic resource provisioning refers to the decision-making process of the positions of containers belonging to a certain tenant to balance the utilization and elasticity to achieve the greatest efficiency of overall energy consumption for the cloud data centers. Across multiple experimental results, EEP consistently outperforms the baseline, demonstrating its adaptability and effectiveness in handling dynamic workloads, reducing migration time, and ensuring stable performance in various scenarios.

In this work, we focus on the issue of online resource provisioning and efficient scheduling for multiple tenants in cloud data centers. However, the current problem structure is based on a fat-tree topology, which is a specific type of tree structure

that divides and completes grouping. In our future work, we will discuss more general situations and study how to address user QoS problems in data centers based on a general tree structure.

REFERENCES

- [1] L. Helali and M. N. Omri, "Software license consolidation and resource optimization in container-based virtualized data centers," *J. Grid Comput.*, vol. 20, no. 2, 2022, Art. no. 13.
- [2] N. T. Hieu, M. Di Francesco, and A. Ylä-Jääski, "Virtual machine consolidation with multiple usage prediction for energy-efficient cloud data centers," *IEEE Trans. Serv. Comput.*, vol. 13, no. 1, pp. 186–199, Jan./Feb. 2020.
- [3] Z. Wen, Q. Chen, Q. Deng, Y. Niu, Z. Song, and F. Liu, "ComboFunc: Joint resource combination and container placement for serverless function scaling with heterogeneous container," *IEEE Trans. Parallel Distrib. Syst.*, vol. 35, no. 11, pp. 1989–2005, Nov. 2024.
- [4] B. Burns, J. Beda, K. Hightower, and L. Evenson, *Kubernetes: Up and Running*. Sebastopol, CA, USA: O'Reilly Media, 2022.
- [5] T. Kubernetes, "Kubernetes," vol. 24, May 2019. [Online]. Available: <https://dzone.com/storage/attachments/14131598-dzone-kubernetesbundle.pdf>
- [6] A. Khan, M. Zakarya, R. Buyya, R. Khan, M. Khan, and O. Rana, "An energy and performance aware consolidation technique for containerized datacenters," *IEEE Trans. Cloud Comput.*, vol. 9, no. 4, pp. 1305–1322, Fourth Quarter 2021.
- [7] W. Khemili, J. E. Hajlaoui, and M. N. Omri, "Energy aware fuzzy approach for placement and consolidation in cloud data centers," *J. Parallel Distrib. Comput.*, vol. 161, pp. 130–142, 2022.
- [8] Z. Li, K. Lin, S. Cheng, L. Yu, and J. Qian, "Energy-efficient and load-aware VM placement in cloud data centers," *J. Grid Comput.*, vol. 20, no. 4, 2022, Art. no. 39.
- [9] W. Lin, W. Wu, and L. He, "An online virtual machine consolidation strategy for dual improvement in performance and energy conservation of server clusters in cloud data centers," *IEEE Trans. Serv. Comput.*, vol. 15, no. 2, pp. 766–777, Mar./Apr. 2022.
- [10] Z. Zhong and R. Buyya, "A cost-efficient container orchestration strategy in Kubernetes-based cloud computing infrastructures with heterogeneous resources," *ACM Trans. Internet Technol.*, vol. 20, no. 2, pp. 1–24, 2020.
- [11] H. M. Naeen, E. Zeinali, and A. T. Haghghat, "Adaptive Markov-based approach for dynamic virtual machine consolidation in cloud data centers with quality-of-service constraints," *Softw. Pract. Experience*, vol. 50, no. 2, pp. 161–183, 2020.
- [12] M. Reza khani, N. Sarrafzadeh-Ghadimi, R. Entezari-Maleki, L. Sousa, and A. Movaghar, "Energy-aware QoS-based dynamic virtual machine consolidation approach based on RL and ANN," *Cluster Comput.*, vol. 27, pp. 827–843, 2024.
- [13] J. Wang, H. Gu, J. Yu, Y. Song, X. He, and Y. Song, "Research on virtual machine consolidation strategy based on combined prediction and energy-aware in cloud computing platform," *J. Cloud Comput.*, vol. 11, no. 1, pp. 1–18, 2022.
- [14] H. Wu, Y. Chen, C. Zhang, J. Dong, and Y. Wang, "Loads prediction and consolidation of virtual machines in cloud," *Concurrency Computation: Pract. Experience*, vol. 35, 2023, Art. no. e7760.
- [15] J. Zeng, D. Ding, K. Kang, H. Xie, and Q. Yin, "Adaptive DRL-based virtual machine consolidation in energy-efficient cloud data center," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2991–3002, Nov. 2022.
- [16] D. Saxena, I. Gupta, J. Kumar, A. K. Singh, and X. Wen, "A secure and multiobjective virtual machine placement framework for cloud data center," *IEEE Syst. J.*, vol. 16, no. 2, pp. 3163–3174, Jun. 2022.
- [17] R. Zolfaghari, A. Sahafi, A. M. Rahmani, and R. Rezaei, "An energy-aware virtual machines consolidation method for cloud computing: Simulation and verification," *Softw. Pract. Experience*, vol. 52, no. 1, pp. 194–235, 2022.
- [18] D. Saxena, A. K. Singh, and R. Buyya, "OP-MLB: An online VM prediction-based multi-objective load balancing framework for resource management at cloud data center," *IEEE Trans. Cloud Comput.*, vol. 10, no. 4, pp. 2804–2816, Fourth Quarter 2022.
- [19] M. A. N. Saif, S. K. N. Aradhya, B. A. H. Murshed, O. A. M. F. Alnaggar, and I. M. S. Ali, "Multi-objective container scheduling and multi-path routing for elastic business process management in autonomic multi-tenant cloud," *Concurrency Computation Pract. Experience*, vol. 35, no. 6, pp. 1–1, 2023.
- [20] M. Yazdanbakhsh, R. Isfahani, and M. Ramezanzpour, "MODE: A multi-objective strategy for dynamic task scheduling through elastic cloud resources," *Majlesi J. Elect. Eng.*, vol. 14, no. 2, pp. 128–141, 2020.
- [21] R. Karthikeyan, V. Balamurugan, R. Cyriac, and B. Sundaravadivazhagan, "COSCO2: AI-augmented evolutionary algorithm based workload prediction framework for sustainable cloud data centers," *Trans. Emerg. Telecommun. Technol.*, vol. 34, no. 1, 2023, Art. no. e4652.
- [22] G. L. Stavrinides and H. D. Karatza, "Scheduling real-time IoT workflows in a fog computing environment utilizing cloud resources with data-aware elasticity," in *Proc. 6th Int. Conf. Fog Mobile Edge Comput.*, 2021, pp. 1–8.
- [23] B. Feng, Z. Ding, X. Zhou, and C. Jiang, "Heterogeneity-aware proactive elastic resource allocation for serverless applications," *IEEE Trans. Serv. Comput.*, vol. 17, no. 5, pp. 2473–2487, Sep./Oct. 2024.
- [24] J. Zhu, X. Li, R. Ruiz, W. Li, H. Huang, and A. Y. Zomaya, "Scheduling periodical multi-stage jobs with fuzziness to elastic cloud resources," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 12, pp. 2819–2833, Dec. 2020.
- [25] D. Saxena, I. Gupta, A. K. Singh, and C. N. Lee, "A fault tolerant elastic resource management framework toward high availability of cloud services," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 3, pp. 3048–3061, Sep. 2022.
- [26] G. Yu, P. Chen, and Z. Zheng, "Microscaler: Cost-effective scaling for microservice applications in the cloud with an online learning approach," *IEEE Trans. Cloud Comput.*, vol. 10, no. 2, pp. 1100–1116, Second Quarter 2022.
- [27] U. Arshad, M. Aleem, G. Srivastava, and J. C. W. Lin, "Utilizing power consumption and SLA violations using dynamic VM consolidation in cloud data centers," *Renewable Sustain. Energy Rev.*, vol. 167, 2022, Art. no. 112782.
- [28] L. Wesolowski et al., "Datacenter-scale analysis and optimization of GPU machine learning workloads," *IEEE Micro*, vol. 41, no. 5, pp. 101–112, Sep./Oct. 2021.
- [29] X. Ma, H. Xu, H. Gao, M. Bian, and W. Hussain, "Real-time virtual machine scheduling in industry IoT network: A reinforcement learning method," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 2129–2139, Feb. 2023.
- [30] S. Singh and R. Kumar, "Energy efficient optimization with threshold based workflow scheduling and virtual machine consolidation in cloud environment," *Wireless Pers. Commun.*, vol. 128, no. 4, pp. 2419–2440, 2023.
- [31] M. Zakarya, L. Gillam, K. Salah, O. Rana, S. Tirunagari, and R. Buyya, "CoLocateMe: Aggregation-based, energy, performance and cost aware VM placement and consolidation in heterogeneous IaaS clouds," *IEEE Trans. Serv. Comput.*, vol. 16, no. 2, pp. 1023–1038, Mar./Apr. 2023.
- [32] S. Shen, V. Van Beek, and A. Iosup, "Statistical characterization of business-critical workloads hosted in cloud datacenters," in *Proc. 15th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2015, pp. 465–474.
- [33] J. Comden, S. Yao, N. Chen, H. Xing, and Z. Liu, "Online optimization in cloud resource provisioning: Predictions, regrets, and algorithms," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 1, pp. 1–30, 2019.
- [34] A. Berl et al., "Energy-efficient cloud computing," *Comput. J.*, vol. 53, no. 7, pp. 1045–1051, 2010.
- [35] Kubernetes Scheduler, "Kubernetes Documentation," 2024. [Online]. Available: <https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/>
- [36] S. Lu, J. Wu, J. Shi, J. Fang, J. Zhang, and H. Liu, "Towards dynamic request updating with elastic scheduling for multi-tenant cloud-based data center network," *IEEE Trans. Netw. Sci. Eng.*, vol. 11, no. 2, pp. 2223–2237, Mar./Apr. 2024.



Shuaibing Lu (Member, IEEE) received the PhD degree in computer science and technology from Jilin University, Changchun, in 2019. She is currently a lecturer with the College of Computer Science, Beijing University of Technology. She is supported by the China Scholarship Council as a visiting scholar supervised by Prof. Jie Wu with the Department of Computer and Information Sciences, Temple University (2016–2018). Her current research focuses on distributed computing, cloud computing, and edge computing.



Ran Yan received the BSc degree in network engineering from Beijing Information Science and Technology University. She is currently working toward the MSc degree with the College of Computer Science, Beijing University of Technology. Her research interests include cloud computing and edge computing.



Xinyu Deng received the BSc degree in network computer science and technology from the Beijing University of Technology. He is currently working toward the MSc degree in computer science with the Viterbi School of Engineering, University of Southern California. His research interests include the distributed system and the storage system.



Jie Wu (Fellow, IEEE) is the director of the Center for Networked Computing and Laura H. Carnell professor with Temple University. He also serves as the director of International Affairs at the College of Science and Technology. He served as chair with the Department of Computer and Information Sciences from the summer of 2009 to the summer of 2016 and associate vice provost for International Affairs from the fall of 2015 to the summer of 2017. Prior to joining Temple University, he was a program director with the National Science Foundation and was a distinguished professor with Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. He regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including the IEEE TRANSACTIONS ON SERVICE COMPUTING and JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING. He was general co-chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, ACM MobiHoc 2014, ICPP 2016, and IEEE CNS 2016, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society distinguished visitor, ACM distinguished speaker, and chair of the IEEE Technical Committee on Distributed Processing (TCDP). He is a CCF distinguished speaker. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.



Shen Wu received the BSc degree in computer science and technology from Yanshan University. He is currently working toward the MSc degree in electronic information with the College of Computer Science, Beijing University of Technology. He is a student member of the CCF.



Zhi Cai (Member, IEEE) received the MSc degree from the School of Computer Science, University of Manchester, in 2007, and the PhD degree from the Department of Computing and Mathematics, Manchester Metropolitan University, U.K., in 2011. He is an associate professor with the College of Computer Science, Beijing University of Technology, China. His research interests include information retrieval, ranking in relational databases, keyword search, and intelligent transportation systems.



Jackson Yang is currently working toward the undergraduate degree with the School of Software, Beijing Jiaotong University, specializing in software engineering. He has participated in several projects focusing on the development of VR systems for psychological treatments, edge computing, and cloud computing.



Juan Fang (Member, IEEE) received the MS degree from the Jilin University of Technology, Changchun, China, in 1997, and the PhD degree from the College of Computer Science, Beijing University of Technology, Beijing, China, in 2005. In 1997, she joined the College of Computer Science, Beijing University of Technology. Since 2015, she has been a professor with the Beijing University of Technology. Her research interests include high performance computing, edge computing, and Big Data technology.