Article

Dynamic Resource Aggregation Method Based on Statistical Capacity Distribution

Yuexin Wang [1,2], Jiali You [1,2] and Yang Li [1,2,*]

[1] National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China; wangyx@dsp.ac.cn (Y.W.); youjl@dsp.ac.cn (J.Y.)
[2] School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Beijing 100049, China
[*] Correspondence: liyang@dsp.ac.cn

Abstract: The computing network is a novel architecture that enables resource matching through the network. In distributed computing networks, computing resource management devices collect resource information and report it to network nodes. These nodes then broadcast the information to guide resource matching. One challenge is efficiently aggregating and disseminating computing resource information, as directly reporting fully multi-dimensional data can cause excessive overhead, while overly simplified aggregation may reduce matching accuracy. Existing aggregation methods typically rely on static resource information, overlooking the heterogeneity and dynamics of computing resources that arise from variations in resource capabilities and fluctuations over time, leading to suboptimal matching decisions. In response, this study proposes a dynamic resource aggregation method based on statistical capacity distribution. By modeling the capacity distribution of computing nodes, this method captures dynamic resource information, enabling more precise resource matching. Additionally, constructing resource groups and calculating representative distributions effectively compress the volume of data announcements. Experiments and data analysis demonstrate that, compared to static resource matching methods, the proposed method improves matching accuracy by 48%. Furthermore, it reduces announcement overhead by approximately 77.1% compared to existing dynamic resource allocation methods. These findings provide an efficient solution for resource aggregation in distributed computing networks.

Keywords: computing network; computing resource aggregation; computing resource announcement

Citation: Wang, Y.; You, J.; Li, Y. Dynamic Resource Aggregation Method Based on Statistical Capacity Distribution. *Electronics* **2024**, *13*, 4617. https://doi.org/10.3390/electronics13234617

Academic Editor: Christos J. Bouras

Received: 25 September 2024
Revised: 15 November 2024
Accepted: 20 November 2024
Published: 22 November 2024

1. Introduction

With the rapid advancement and widespread application of technologies such as deep learning and big data, modern society is increasingly embracing digitization and intelligence. These intelligent applications rely on robust computing resources to process vast amounts of data. Existing resource pools and computing-capable terminal devices are deployed in a dispersed manner. There is a lack of effective collaboration mechanisms between edge computing nodes and between edge nodes and cloud computing nodes, resulting in low utilization of computing resources. With the advancement of network technology, especially the maturation of emerging technologies such as Software-Defined Networking (SDN), Network Function Virtualization (NFV), and Information-Centric Networking (ICN), computing resources can be dynamically connected through the network to enable efficient collaboration among cloud, edge, and terminal devices. To unify the perception and management of ubiquitously distributed computing resources, the concept of the "Computing Network" has been proposed [1–5]. Current designs of computing network architectures [6–8] are primarily categorized into centralized and distributed types, with the distributed architecture offering notable advantages in terms of reliability and scalability. In distributed computing networks, resource management devices deployed at the network edge collect and organize computing resource information, which is then reported to network nodes. These nodes distribute the information and create routing tables constrained by both computing and network resources to guide resource allocation.

However, computing resource management within this framework presents significant challenges, particularly regarding the granularity of the resource information collected, which directly affects the efficiency of resource allocation. Reporting detailed, multi-dimensional resource information to the network would lead to excessive data announcements and expansion of routing table entries [9,10]. Conversely, transmitting coarse-grained information, such as simple aggregated information, may reduce the accuracy of resource matching. Therefore, the key challenge of resource management lies in how to effectively aggregate computing resource information to reduce data volume while maintaining accuracy in resource matching.

By unifying the management of widely distributed and diverse computing resources, the computing network provides robust support for modern intelligent applications. However, the heterogeneity and dynamics of these resources present new challenges for resource aggregation. Current research on resource aggregation primarily focuses on multi-cluster management systems such as Liqo and Karmada [11,12], which generally perform simple statistical aggregation based on static, recently observed resource information to guide resource matching. While this approach can be effective in certain scenarios, it demonstrates significant limitations in the context of distributed computing networks.

Firstly, differences in hardware configurations and computing capabilities across nodes cause the heterogeneity of resources. Simple statistical aggregation methods, such as summing or averaging resource information, only reflect the overall status of a computing cluster but overlook the differences between individual nodes. This can lead to mismatches between computing requirements and the actual capability of individual nodes, resulting in suboptimal resource allocation decisions. Secondly, research has shown [13–16] that the resource capacity of computing nodes can fluctuate significantly over short periods due to factors such as load variations, user behavior, and network stability. This dynamic behavior makes resource-matching strategies based on static resource information unable to respond quickly to changes in node capacity, potentially leading to issues of resource overload or underutilization. Existing dynamic resource allocation methods [17–19] typically lack in-depth consideration of resource aggregation. Most approaches directly utilize time-varying workloads to represent dynamic resource information. While this approach can effectively capture the dynamic nature of resources, it often results in high data announcement volumes, increasing network communication overhead and impacting the scalability of the system.

To address these challenges, this paper proposes a dynamic resource aggregation method based on statistical capacity distribution, specifically designed for distributed computing networks. Unlike existing dynamic resource allocation methods, this method fully considers both the heterogeneity and dynamics of computing resources by modeling the capacity distribution of nodes to extract dynamic resource information. To minimize information loss during resource aggregation, computing nodes are grouped based on the similarity of their capacity distributions and representative distributions from each group are reported to the network. This approach effectively preserves differences in resource capabilities between nodes. The main advantage of this method lies in its ability to model the capacity distribution of computing nodes accurately and construct appropriate resource groups. By capturing dynamic resource information and significantly compressing data, the method ensures effective resource matching without losing essential details. The main contributions of this paper are as follows:

- Developing a dynamic resource modeling approach: By formalizing resource requirements and using capacity distribution to model computing resources, this method accurately captures the dynamic resource information of computing nodes while compressing the data volume. Compared to static resource allocation methods, it improves the accuracy of resource matching and reduces the risk of resource overload and performance degradation.

- Proposing a resource group construction method: We apply an Expectation-Maximization (EM) clustering algorithm based on a multinomial mixture model to group computing nodes with similar capacity distributions. This method effectively minimizes information loss caused by aggregation.
- Designing an efficient resource aggregation mechanism: By calculating representative capacity distributions for each resource group, this mechanism significantly reduces resource announcement overhead.

The remainder of this paper is organized as follows: Section 2 reviews relevant research on the computing network, existing solutions to the problem of excessive resource information announcements, resource aggregation mechanisms, and clustering algorithms. Section 3 provides a design overview of the proposed method. Section 4 presents a detailed explanation of each component of the dynamic resource aggregation method based on statistical capacity distribution. In Section 5, we carry out simulation experiments and analyze the performance of the proposed scheme. Finally, Section 6 summarizes the method and the simulation results.

2. Related Works

Computing network is an emerging network architecture. Different from traditional network designs, it aims to interconnect ubiquitously distributed computing resources across cloud, edge, and device layers, enabling on-demand and flexible allocation of both computing and network resources [20]. Currently, both academia and industry are actively exploring computing networks, with companies such as China Unicom, China Mobile, and Huawei proposing various computing network architectures [21–24]. These architectures are primarily classified into centralized and distributed types.

In centralized architectures [25–27], resource allocation is managed by a single resource scheduling center. This center maintains a global view of all resources, collects resource information from all computing nodes, and integrates it with network data to orchestrate unified scheduling. In contrast, distributed architectures [28–30] eliminate the need for a central scheduling center. Instead, resource information is exchanged directly between network nodes, and each node generates routing tables constrained by both computing and network resources, enabling integrated scheduling. Compared with centralized architectures, distributed architectures offer greater reliability and scalability [5,31]. However, they also face challenges such as excessive data broadcasts and inflated routing tables due to the large-scale announcement of computing resource information.

To address the issue of excessive computing resource announcements, existing methods can be categorized into two approaches: announcement structure optimization and resource aggregation techniques. In terms of announcement structure optimization, the authors in [32] propose constructing a computing network announcement topology and dynamically adjusting the network structure based on the number of nodes, thus reducing unnecessary network load. However, constructing this topology requires significant computational effort, which increases the burden on network nodes and may backfire due to the additional overhead of topology announcements. Another approach [33] creates a distributed network of scheduling nodes based on the Chord protocol, where a small number of nodes on the Chord ring store computing resource information, effectively reducing the volume of announcements. Nevertheless, maintaining such a scheduling network is complex and presents significant implementation challenges. In contrast to announcement structure optimization, resource aggregation techniques focus on information compression. These methods explore how to achieve precise aggregation of computing resource information through efficient clustering or statistical methods, which are easier to implement for real computing networks. Therefore, this paper conducts an in-depth study of resource aggregation techniques.

In terms of resource aggregation techniques, existing research primarily focuses on multi-cluster management systems. Projects such as ClusterAPI and Liqo [34–38] use simple statistical methods, such as sum, maximum, and average to aggregate resource information across computing nodes. For instance, ClusterAPI [34] aggregates the remaining resources for each dimension (e.g., CPU and memory) by summing them across all nodes within the cluster, providing a total for each resource dimension. However, this approach simply aggregates resources without accounting for fragmentation between nodes, which can lead to resource mismatches. For example, in a cluster of 2000 nodes, if each node has less than one CPU core available, the aggregation method would report 2000 cores as remaining; but in reality, the cluster cannot accommodate any Pod instance requiring more than one CPU core. While these simple statistical aggregation methods offer an overview of the cluster's resource status, they overlook resource differences and fragmentation between nodes, thereby reducing the accuracy of resource matching. Considering fragmented resources, Karmada [39] introduces a "custom resource model" for each cluster. This model categorizes the resources of computing nodes into different levels and counts the number of nodes within each level, offering a more refined aggregation method than simple statistical approaches. However, all these resource aggregation methods rely on static resource information, and there is a notable lack of research on the aggregation of dynamic resource information.

In terms of resource allocation, existing computing resource matching mechanisms can be divided into two categories: static resource matching and dynamic resource matching. Most studies in the current literature base scheduling decisions on static resource information [40–43]. For instance, the authors in [40] propose a placement method for Virtual Network Functions (VNFs) that achieves resource scheduling using the TOPSIS approach. The authors in [41] improve the Kubernetes resource scheduling algorithm by considering CPU, memory, network, and I/O metrics, thus enhancing scheduling efficiency. The authors in [42] introduce a resource allocation method similar to the PageRank algorithm, ranking computing nodes and virtual nodes based on available resources. Some studies [17–19,44,45] focus on dynamic resource allocation. For example, the authors in [17] present a virtual machine load-balancing method based on a genetic algorithm that uses historical data on virtual machine resource usage to achieve load balancing. The authors in [18] develop a heuristic algorithm to maximize resource utilization by considering time-varying resource demands in virtual networks. The authors in [44] investigate a dynamic workload-based VNF placement method to improve network resource utilization. The authors in [45] introduce a scheduling framework called PowerNets, which uses time-varying workloads to reduce energy consumption in data centers. Although these studies consider time-varying workloads, most approaches directly use raw workload historical data for matching, resulting in significant data announcement volumes that are not suitable for more complex distributed computing networks. Compared to these existing dynamic resource allocation methods, our research offers distinct advantages. Firstly, we model the dynamic resource information of computing nodes using capacity distribution, compressing the data volume in the time dimension. Secondly, we employ a clustering algorithm to construct resource groups for computing nodes and select representative capacity distributions for resource matching, thereby reducing the number of computing nodes that need to be announced and further lowering the data volume. This approach not only reduces data announcement overhead but also maximizes the accuracy of resource matching.

To more effectively compress dynamic information of computing nodes, this paper employs a clustering algorithm to group similar nodes into resource groups, leading to a detailed investigation of clustering algorithms. Clustering algorithms can be broadly categorized into partition-based, hierarchical, density-based, and model-based approaches. Partition-based clustering algorithms [46] operate under the assumption that "points within a cluster are close to each other, while points in different clusters are far apart". However, these algorithms are typically limited to simple, low-dimensional data and cannot accurately capture complex relationships between multi-dimensional data, such as those found in resource capacity distributions. Hierarchical clustering algorithms [47] build a cluster hierarchy by gradually merging or splitting clusters, but when faced with dynamic data characterized by complex, multi-dimensional features, these methods often lead to confusion in cluster hierarchy and a decline in accuracy. Density-based clustering algorithms [48] identify clusters and outliers by partitioning high-density regions, but they are primarily suited for low-dimensional spaces and struggle to effectively recognize complex patterns in multi-dimensional dynamic information of computing nodes.

There are existing methods [36,49] for clustering computing nodes. However, these methods cluster based on static information where each computing node is represented as single-dimensional data. In contrast, this paper takes into account the dynamic information of computing nodes and clusters resource capacity distributions with multi-dimensional data. Therefore, the traditional clustering algorithms mentioned above are not applicable to this study. On the other hand, model-based clustering algorithms [50,51] assume that data points are generated by a mixture of models and use statistical methods to estimate the model parameters of these models. Given the special requirements of dynamic resource clustering, this paper adopts a model-based clustering algorithm to group computing nodes with similar resource usage characteristics. These algorithms are better suited for handling complex data distributions and are more effective in capturing the dynamic nature of computing resources.

3. Design Overview

3.1. System Architecture

As shown in Figure 1, the main entities in the distributed computing network architecture include the computing node (CN), gateway node (GN), scheduling node (SN), User, and user client. SNs are interconnected through the network, with each SN connected to a GN in a one-to-one connection relationship, while GNs are connected to CNs either one-to-one or one-to-many. Also, SNs are organized in a hierarchical architecture [28]. CNs periodically report their resource capacity information to their respective GNs, which then aggregate this information and forward it to SNs. SNs exchange resource capacity information among themselves. User client collects computing resource requests from User and sends them to SNs, which then match User's resource requirements with the available resource capacities and allocate appropriate CNs accordingly. Based on this architecture, this paper proposes a dynamic resource aggregation method based on statistical capacity distribution.

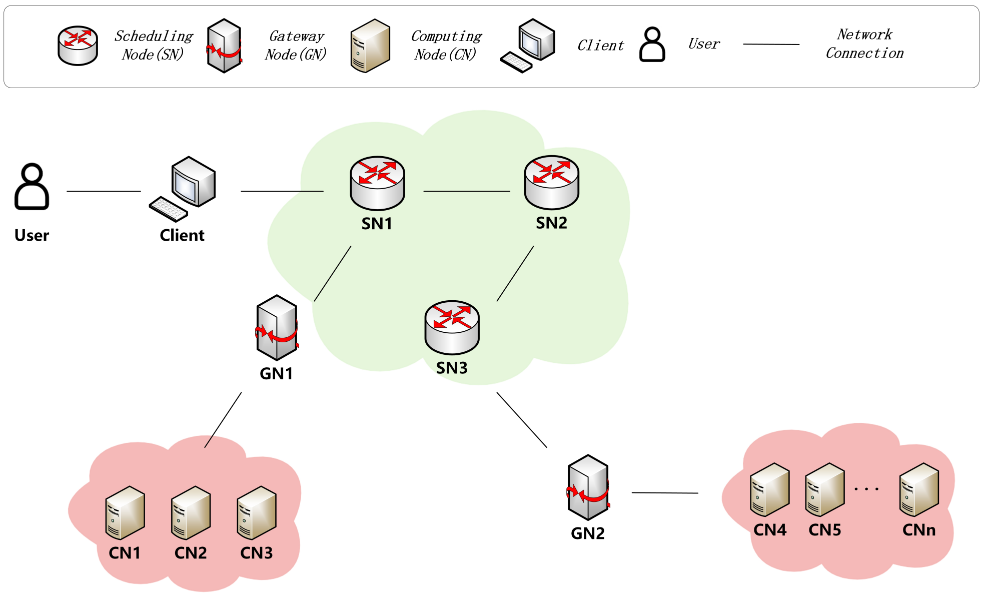


Figure 1. System architecture overview.

3.2. Resource Aggregation Idea Based on Statistical Capacity Distribution

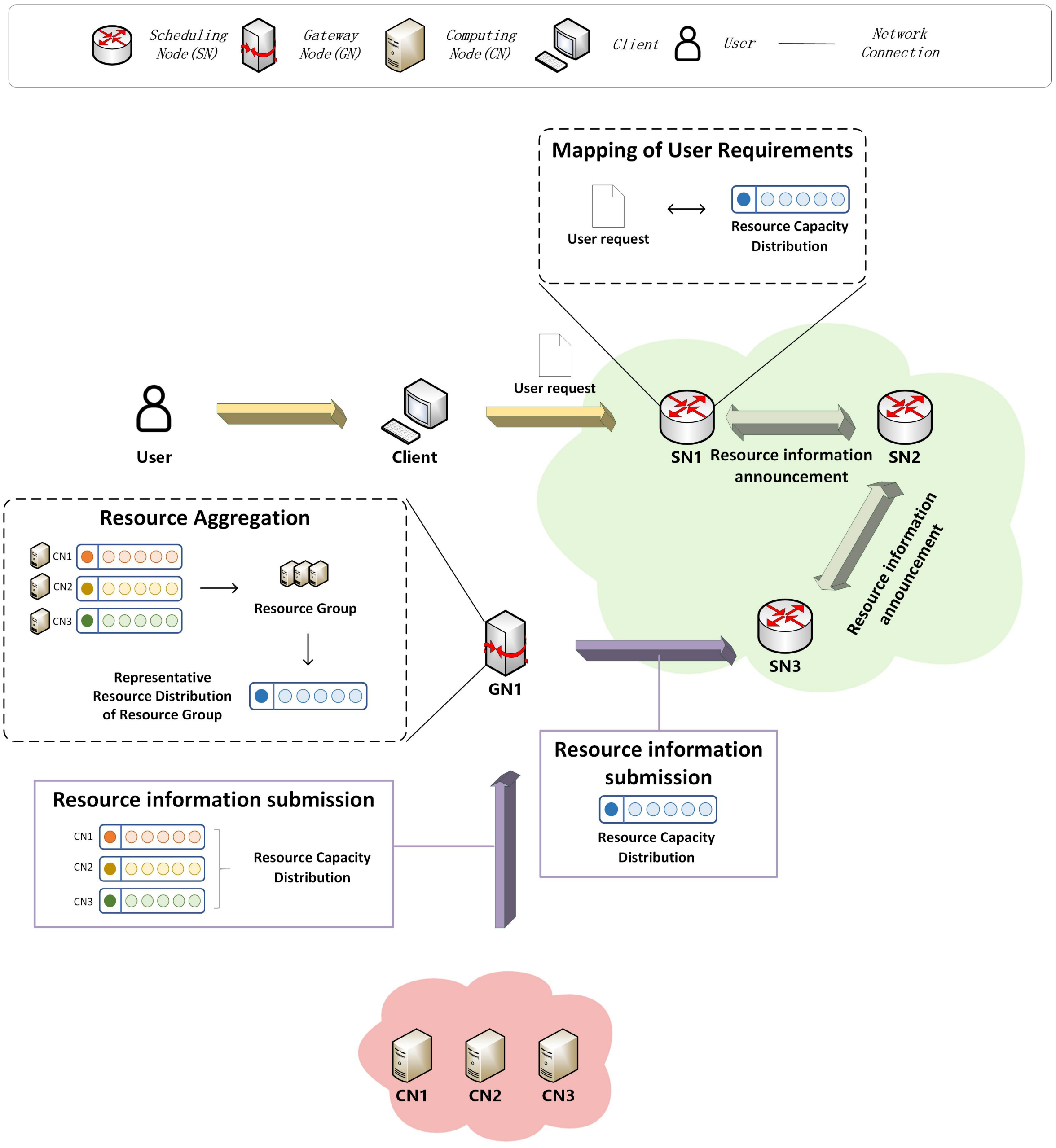
In existing computing network resource scheduling systems, the resource requirement of an application is typically defined as the minimum capacity needed for a certain resource. These systems rely on static resource capacity information for scheduling. However, since the resource requirements of applications and resource capacities of computing nodes can fluctuate significantly over time, relying solely on the minimum resource requirement for scheduling can easily lead to resource overload, resulting in performance degradation or system failures. To address this challenge, we propose introducing a tolerance factor into the definition of resource requirements to provide a more flexible scheduling mechanism.

In current resource scheduling scenarios, application tolerance to resource overload [44] is often used to guide resource matching, allowing systems to better adapt to the dynamic nature of computing resources. The authors in [52] conducted a study on dynamic resource allocation for virtual machines in cloud computing environments. They demonstrated that incorporating tolerance enables scheduling systems to flexibly respond to resource fluctuations, thereby avoiding overload issues. The authors in [53] further emphasized the significance of dynamic resource allocation through tolerance mechanisms in data stream processing systems. Building on these insights, this paper formalizes resource requirements as a triplet $\{R, C_d, P\}$, where R represents the resource type (including CPU, memory, storage, etc.), C_d denotes the minimum required capacity, and P indicates the application's tolerance to resource overload. Tolerance P is defined as the proportion of time within a specific period during which the application can accept a resource capacity falling below C_d.

To accurately capture the dynamic resource usage of computing nodes, we introduce the concept of resource capacity distribution. The resource capacity distribution $\boldsymbol{X}(\boldsymbol{C_r})$ represents the probability distribution of resource usage at each capacity level within a given time period for a computing node. By leveraging this distribution, we can directly map application resource requirements within it, ensuring their reliable execution on computing nodes. Additionally, resource capacity distribution comprehensively preserves the dynamic resource information of nodes, facilitating more precise resource scheduling and allocation. Researches have shown that once applications stabilize on computing nodes, their resource capacity distributions often exhibit distinct periodic characteristics. Studies by Chen et al. [54], Mishra et al. [55], and Benson et al. [56] reveal the periodic characteristics of enterprise workloads and data center resource usage, further emphasizing the significance of using periodic resource capacity distributions for scheduling decisions.

However, broadcasting the resource capacity distribution of each computing node to scheduling nodes would generate significant data announcement overhead. To mitigate this, we propose a resource aggregation method that clusters computing nodes based on the similarity of their resource capacity distributions. By clustering nodes into resource groups with similar distributions, we can extract a representative resource capacity distribution for each group. This approach maximizes the retention of original data characteristics while minimizing information loss. Given that resource capacity distributions are multi-dimensional and may follow arbitrary patterns, we employ a model-based clustering algorithm to ensure accurate aggregation of resource capacity distributions.

Figure 2 illustrates the proposed resource aggregation method. Initially, the resource capacity distribution of each computing node is extracted and reported to the gateway nodes (GNs). Next, the GNs group computing nodes (CNs) with similar distributions into the same resource group and calculate a representative resource capacity distribution for each group. These aggregated representative resource capacity distributions are subsequently reported to the scheduling nodes (SNs), which share and exchange this information among themselves. Finally, the SNs match resource requirements with the representative resource capacity distributions to achieve efficient resource allocation.



Scheduling Node (SN)
Gateway Node (GN)
Computing Node (CN)
Client
User
Network Connection
Mapping of User Requirements
User request
Resource Capacity Distribution
User request
User
Client
SN1
Resource information announcement
SN2
Resource information announcement
SN3
Resource Aggregation
CN1
CN2
CN3
Resource Group
Representative Resource Distribution of Resource Group
GN1
Resource information submission
Resource Capacity Distribution
Resource information submission
Resource Capacity Distribution
CN1
CN2
CN3

Figure 2. Computing resource aggregation diagram.

4. Proposed Method

4.1. Statistical Resource Usage Pattern

4.1.1. User Resource Requirement

In this paper, the resource requirement is defined as a triplet $\{R, C_d, P\}$, where R represents different types of resources (such as CPU, memory, storage, etc.), C_d denotes the minimum required resource capacity, and P indicates the application's tolerance to resource overload. Tolerance P is defined as the proportion of time within a specific period during which the application can accept resource capacity falling below C_d. It measures the extent to which an application can tolerate insufficient resource supply without impacting its normal operation or causing significant performance degradation.

The parameter C_d specifies the minimum resource capacity necessary to maintain acceptable performance, while the parameter P defines the application's tolerance for resource fluctuations. This definition ensures that resource allocation can meet the application's operational needs without incurring unnecessary overhead.

4.1.2. Resource Capacity Distribution

Based on the definitions above, an application can specify different resource requirement parameters R, C_d, and P. To match these diverse requirements, we introduce the concept of resource capacity distribution to describe the resource usage of computing nodes over a given period of time.

To quantify the resource usage of computing nodes, we divide the resource capacity into several continuous integer intervals, with each interval representing a capacity level. The capacity level vector is defined as

$$\boldsymbol{C_r} = \begin{pmatrix} c_{r1} \\ c_{r2} \\ \vdots \\ c_{rm} \end{pmatrix} \tag{1}$$

where $\mathbf{c_{ri}}$ represents the ith capacity level and m denotes the total number of capacity levels.

Assuming that at any given time t, the resource capacity of a computing node is $c_r(t)$, the sampled resource capacity over the time interval $[t_0, t_1]$ with a sampling interval of Δt can be expressed as

$$\boldsymbol{C_r}(t_0, t_1) = \begin{pmatrix} c_r(t_0) \\ c_r(t_0 + \Delta t) \\ \vdots \\ c_r(t_1) \end{pmatrix} \tag{2}$$

where $\boldsymbol{C_r}(t_0, t_1)$ represents the sequence of resource capacity samples of a computing node over the time interval $[t_0, t_1]$. Based on the above definitions, we define the resource capacity distribution as $\boldsymbol{X}(\boldsymbol{C_r})$. $\boldsymbol{X}(\boldsymbol{C_r})$ represents the probability vector of a computing node's resource capacity across different capacity levels within a specific time interval. Specifically, over the time interval $[t_0, t_1]$, the probability distribution of the resource capacity samples $\boldsymbol{C_r}(t_0, t_1)$ across various capacity levels can be expressed as $\boldsymbol{X}(\boldsymbol{C_r}(t_0, t_1))$. Let $\boldsymbol{n}(\boldsymbol{C_r})$ represent the sequence of the number of sampling points at each capacity level $\boldsymbol{C_r}$. Then, the resource capacity distribution $\boldsymbol{X}(\boldsymbol{C_r})$ is defined as

$$\boldsymbol{X}(\boldsymbol{C_r}) = \frac{\boldsymbol{n}(\boldsymbol{C_r})}{J} = \begin{pmatrix} \frac{n(c_{r1})}{J} \\ \frac{n(c_{r2})}{J} \\ \vdots \\ \frac{n(c_{rm})}{J} \end{pmatrix} = \begin{pmatrix} x_{c_{r1}} \\ x_{c_{r2}} \\ \vdots \\ x_{c_{rm}} \end{pmatrix} \tag{3}$$

where $n(c_{ri})$ represents the number of sampling points at which the resource capacity of the computing node falls within the capacity level c_{ri}, and $x_{c_{ri}}$ denotes the probability that the resource capacity falls within the capacity level c_{ri} during the time interval. J represents the total number of sampling points within the time interval, defined as $J = \sum_{i=1}^{m} n(c_{ri})$.

The resource capacity distribution $\boldsymbol{X}(\boldsymbol{C_r})$ is the probability distribution of the node's capacity across various levels, and the sum of the components $x_{c_{ri}}$ satisfies the following equation:

$$\sum_{i=1}^{m} x_{c_{ri}} = 1 \tag{4}$$

This equation indicates that the sum of the probabilities of the resource capacity across all possible capacity levels for a computing node equals 1.

4.1.3. Mapping of User Resource Requirement

The resource requirements can be directly mapped into the resource capacity distribution to determine whether a computing node can meet the application's needs. Algorithm 1 demonstrates the specific mapping method. We introduce the indicator function $\delta(C_r \geq C_d)$ to evaluate whether each capacity level c_{ri} meets the minimum capacity requirement C_d. The indicator function is defined as follows:

$$\delta(C_r \geq C_d) = \begin{cases} 1, if\ C_r \geq C_d \\ 0, if\ C_r < C_d \end{cases} \tag{5}$$

Building on this, we define the capacity threshold vector $\boldsymbol{I}$, where each element indicates whether the corresponding capacity level of the computing node meets the minimum capacity requirement C_d:

$$\boldsymbol{I} = \begin{pmatrix} \delta(c_{r1} \geq C_d) \\ \delta(c_{r2} \geq C_d) \\ \vdots \\ \delta(c_{rm} \geq C_d) \end{pmatrix} \tag{6}$$

We use a vector representation to express the matching relationship between the resource requirements and the resource capacity distribution. The matching condition can be represented by the dot product of the resource capacity distribution $\boldsymbol{X}(\boldsymbol{C_r})$ and the capacity threshold vector $\boldsymbol{I}$:

$$\boldsymbol{I} \cdot \boldsymbol{X}(\boldsymbol{C_r}) = \sum_{\mathbf{i}=1}^{\mathbf{m}} \delta(c_{ri} \geq C_d) \cdot \mathbf{x}_{\mathbf{c_{ri}}} \tag{7}$$

$\boldsymbol{I} \cdot \boldsymbol{X}(\boldsymbol{C_r})$ represents the probability that the resource capacity distribution of the computing node meets the capacity requirement C_d within the given time interval. If this probability is greater than or equal to the tolerance level P specified by the user, the computing node satisfies the user's resource requirements:

$$\sum_{i=1}^{m} \delta(c_{ri} \geq C_d) \cdot x_{c_{ri}} \geq P \tag{8}$$

Algorithm 1 Mapping of User Resource Requirements

```
Input: $C_d$; $P$; $\boldsymbol{X}(\boldsymbol{C_r})$;
Output: TRUE/FALSE;
 1: $total\_percentage = 0$
 2: for $c_{ri}$ in $\boldsymbol{X}(\boldsymbol{C_r})$ do
 3:     if $c_{ri} \geq C_d$ then
 4:         $total\_percentage += x_{c_{ri}}$
 5:     end if
 6: end for
 7: if $total\_percentage \geq P$ then
 8:     //The node can satisfy the demand
 9:     return TRUE
10: else
11:     //The node cannot satisfy the demand
12:     return FALSE
13: end if
```

4.2. Resource Group Construction

4.2.1. Analysis of Resource Capacity Distribution Similarity

The resource capacity distributions of computing nodes often exhibit similarities, primarily due to the similar operating environments and workload tasks they handle. In the same data center, multiple nodes may host the same or similar types of applications, leading to consistent resource usage patterns over time. For example, nodes executing similar computational tasks may show similar probability distributions of CPU and memory utilization across specific capacity ranges. Additionally, nodes within the same data center typically share uniform hardware configurations and network environments, which further contribute to the consistency of their resource capacity distributions. Based on these similarities, aggregating computing nodes with similar resource capacity distributions can effectively optimize resource management and scheduling efficiency.

In Section 4.1.2, we use resource capacity distribution to capture the dynamic capacity information of each node. Each computing node's resource capacity distribution, denoted as $\boldsymbol{X}(\boldsymbol{C_r})$, consists of multiple components $x_{c_{ri}}$. However, transmitting all of these resource capacity distribution data across the network would result in significant data announcement overhead, compromising the system's scalability. To address this issue, we propose a resource aggregation method that groups computing nodes into different resource groups and identifies a representative resource capacity distribution for each group. This approach significantly reduces the amount of data that need to be transmitted, thereby lowering network communication overhead.

We define a resource group as a collection of computing nodes with similar resource capacity distributions, where nodes within the same resource group share identical resource usage characteristics. We cluster the nodes based on the similarity of their resource capacity distributions, where this similarity refers to the degree of proximity in the probability distributions of resource usage across different capacity levels between two computing nodes. This clustering method effectively aggregates nodes with similar resource capacity distributions into the same resource group, enabling the representative resource capacity distribution to more accurately reflect the overall resource status of the nodes within the group, thereby providing more precise information for scheduling and allocation processes.

4.2.2. Resource Group Construction Based on Clustering

To accurately identify and aggregate computing nodes with similar resource capacity distributions, this paper employs a clustering algorithm to construct resource groups. The clustering algorithm must satisfy two key requirements: (1) the data to be clustered can be multi-element data; (2) the resource capacity distribution of computing nodes can follow any distribution, without the need to assume a standard distribution. The Expectation-Maximization (EM) clustering algorithm based on a multinomial mixture model meets these requirements. This algorithm excels in modeling mixed probability distributions, enabling it to accurately identify and group computing nodes with similar resource capacity distributions.

The EM algorithm, commonly used for text clustering, categorizes documents into different groups based on content similarity. Analogously, this algorithm assesses the similarity between computing nodes by comparing their resource capacity distributions, which are modeled using multinomial functions. Specifically, the algorithm calculates the probability that each node belongs to a particular cluster, thereby quantifying the degree of alignment between the node's resource capacity distribution and the multinomial distribution defined within the cluster model.

For each resource capacity distribution, the system treats it as multi-element data and models each node's resource usage using multinomial distributions. Based on this modeling, our method clusters nodes by comparing the similarity of their resource capacity distributions. The algorithm processes a series of multidimensional vectors as input and classifies them into groups based on the similarity between these vector elements. The system then computes the degree of match between each node and a cluster distribution, expressed as the posterior probability that the node belongs to that cluster. These posterior probabilities measure the similarity between a node's resource capacity distribution and the distributions of other nodes within the cluster. The closer a node's resource distribution is to the central distribution of the cluster, the higher its probability of belonging to that cluster.

The EM algorithm based on a multinomial mixture model is an iterative optimization process. It gradually optimizes the clustering results by continuously adjusting the association between vectors and their respective clusters, ultimately maximizing the expected clustering outcome. This algorithm is capable of handling resource capacity distributions composed of multiple elements and can process vectors of any type without assuming a specific distribution, making it highly effective for clustering various resource capacity distributions of different computing nodes.

Building on this foundation, we constructed a multinomial mixture model for the resource capacity distribution. This process involves probabilistic modeling of the resource capacity distributions of computing nodes and solving for the model parameters using the Expectation-Maximization (EM) algorithm. The following outlines the construction process of the multinomial mixture model for resource capacity distribution:

The set of model parameters for the multinomial distribution is denoted as $\Theta = \{\pi_k, \theta_k\}_{k=1}^{K}$. The resource capacity distribution $\boldsymbol{X}(\boldsymbol{C_r})$ can be represented as a mixture of K multinomial distributions, as follows:

$$P(X(C_r) \mid \Theta) = \sum_{k=1}^{K} \pi_k P(\boldsymbol{X}(\boldsymbol{C_r}) \mid \theta_k) \tag{9}$$

where K is the number of clusters, π_k is the mixing coefficient for the kth cluster, satisfying $\sum_{k=1}^{K} \pi_k = 1$, and $P(\boldsymbol{X}(\boldsymbol{C_r}) \mid \theta_k)$ is the multinomial probability density function for the kth cluster, with parameters θ_k. The multinomial probability density function for each cluster k is given by

$$P(\boldsymbol{X}(\boldsymbol{C_r}) \mid \theta_k) = \prod_{i=1}^{m} \left(\frac{x_{c_{ri}}!}{\prod_{j=1}^{J} n_j!} \right) \prod_{j=1}^{J} \left(\theta_{k_j}^{n_j} \right) \tag{10}$$

where m is the total number of capacity levels, $x_{c_{ri}}$ is the probability component of the resource capacity distribution $\boldsymbol{X}(\boldsymbol{C_r})$, and n_j is the number of the jth sampling points at the ith capacity level. To maximize the parameter estimation of the mixture model, the log-likelihood function is defined as

$$L(\Theta) = \sum_{n=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k P(\boldsymbol{X_n}(\boldsymbol{C_r}) \mid \theta_k) \right) \tag{11}$$

where $\boldsymbol{X_n}(\boldsymbol{C_r})$ represents the resource capacity distribution of the nth computing node.

The model parameters can be estimated using the Expectation-Maximization (EM) algorithm. The EM algorithm consists of the following two steps:

E-Step: Calculate the posterior probability function γ_{nk},

$$\gamma_{nk} = \frac{\pi_k P(\boldsymbol{X_n}(\boldsymbol{C_r}) \mid \theta_k)}{\sum_{s=1}^{K} \pi_s P(\boldsymbol{X_n}(\boldsymbol{C_r}) \mid \theta_s)} \tag{12}$$

M-Step: In each iteration of the EM algorithm, the estimated values of the model parameters are

$$\hat{\pi}_k = \frac{1}{N} \sum_{n=1}^{N} \gamma_{nk} \tag{13}$$

$$\hat{\theta}_k = \frac{\sum_{n=1}^{N} \gamma_{nk} \boldsymbol{X_n}(\boldsymbol{C_r})}{\sum_{n=1}^{N} \gamma_{nk}} \tag{14}$$

In the clustering process described in this paper, we employed the Bayesian Information Criterion (BIC) to automatically determine the optimal number of resource groups. BIC is a model selection criterion based on the likelihood function, designed to evaluate the model's fit by quantifying its effectiveness. During clustering, the model parameters are iteratively optimized using the EM algorithm. After each iteration, the corresponding BIC value is calculated to assess different cluster numbers. The BIC formula is as follows:

$$BIC = -2 \cdot \ln(\hat{L}) + m_p \cdot ln(N) \tag{15}$$

where $\hat{L}$ is the maximum likelihood estimate of the model, m_p is the number of model parameters, and N is the total number of computing nodes. The number of model parameters is directly related to the number of resource groups. A model with a lower BIC value is considered to have achieved the best balance between complexity and fit. The clustering number corresponding to the minimum BIC value is the optimal number of resource groups determined by this method. Algorithm 2 presents the specific algorithmic process for resource group construction.

Algorithm 2 The Cluster-Based Resource Group Construction Algorithm

```
Input: $\boldsymbol{X}(\boldsymbol{C_r}) = \{\boldsymbol{X}_1(\boldsymbol{C_r}), \boldsymbol{X}_2(\boldsymbol{C_r}), \ldots, \boldsymbol{X_n}(\boldsymbol{C_r})\}$; $K$(initial number of clusters);
Output: $\Theta = \{\pi_k, \theta_k\}_{k=1}^{K}$ for each cluster $k$; $Z = \{z_1, z_2, \ldots, z_n\}$(cluster allocation for each point);
    $K^*$(the optimal number of clusters);
 1: Initialize $\Theta = \{\pi_k, \theta_k\}_{k=1}^{K}$ for $k = 1$ to $K$, $L_{new} = -\infty, L_{old} = 0, BIC_{best} = \infty$
 2: for $k$ in $K$ do
 3:     repeat
 4:         for $n$ in $N$ do
 5:             for $k$ in $K$ do
 6:                 //E-step
 7:                 calculation $\gamma_{nk}$
 8:             end for
 9:         end for
10:         for $n$ in $N$ do
11:             //M-step
12:             calculation $\hat{\pi}_k$
13:             calculation $\hat{\theta}_k$
14:         end for
15:         $L_{old} = L_{new}$
16:         $L_{new} = \sum_{n=1}^{N} \log\left(\sum_{k=1}^{K} \pi_k P(\boldsymbol{X_n}(\boldsymbol{C_r}) \mid \theta_k)\right)$
17:     until $|L_{new} - L_{old}| < \varepsilon$
18:     calculation BIC
19:     if BIC < $BIC_{best}$ then
20:         $BIC_{best} = BIC$
21:         $K^* = K$
22:     end if
23: end for
24: for $n$ in $N$ do
25:     $Z_n = \underset{k}{arg\ max}\, \gamma_{nk}$
26: end for
27: return $\Theta, Z, K^*$
```

4.3. Calculation of Representative Resource Distribution

When aggregating the resource capacity distributions of computing nodes using the Expectation-Maximization (EM) clustering algorithm based on a multinomial model, each resource group generates a multinomial distribution model that accurately captures the resource usage characteristics of the nodes. This model serves as the representative resource capacity distribution for the resource group, preserving the statistical properties of all nodes within the group. To achieve this, the representative distribution is derived by calculating an average of the resource capacity distributions of the nodes within the group. The calculation formula is as follows:

Assume that N is the number of computing nodes in a certain resource group, and each resource capacity distribution is represented as $\boldsymbol{X_j}(\boldsymbol{C_r}) = \left(x_{jc_{r1}}, x_{jc_{r2}}, \ldots, x_{jc_{rm}}\right)$, where j denotes the jth computing node in the resource group and each node's distribution has m data points. The formula for the representative resource capacity distribution $\boldsymbol{X_j}(\boldsymbol{C_r})$ is as follows:

$$\bar{\boldsymbol{X}}(\boldsymbol{C_r}) = \left(\frac{1}{N} \sum_{j=1}^{N} x_{jc_{r1}},\ \frac{1}{N} \sum_{j=1}^{N} x_{jc_{r2}},\ \ldots,\ \frac{1}{N} \sum_{j=1}^{N} x_{jc_{rm}} \right) \tag{16}$$

This approach ensures that the representative distribution reflects both the individual resource usage characteristics of the nodes and provides a concise summary of the entire resource group, thereby facilitating more efficient decision-making in resource scheduling and management.

Once the representative resource distribution is determined, each resource group needs to only transmit these simplified data to the network, significantly reducing the number of nodes required for resource announcements. This reduction in data transmission not only lowers network communication overhead but also enhances the efficiency of resource management in the distributed computing network. The representative resource distribution provides a concise and effective summary of the overall performance of the resource group, thereby improving the accuracy of resource matching.

5. Simulation Experiment

5.1. Simulation Setup

This paper conducts simulation experiments based on the network topology created using NetworkX, a Python library for network analysis that is easy to learn and use. In this experiment, we used NetworkX to create the network topology shown in Figure 3 to simulate a distributed computing network. The topology includes 10 scheduling nodes, 10 gateway nodes, and 4000 computing nodes. Each computing node is connected to a gateway node, each gateway node is connected to a scheduling node, and the scheduling nodes are interconnected. In Figure 3, the black nodes represent scheduling nodes, the green nodes represent gateway nodes, and the blue nodes represent computing nodes. Due to the large number of computing nodes, only 40 are shown in the figure.

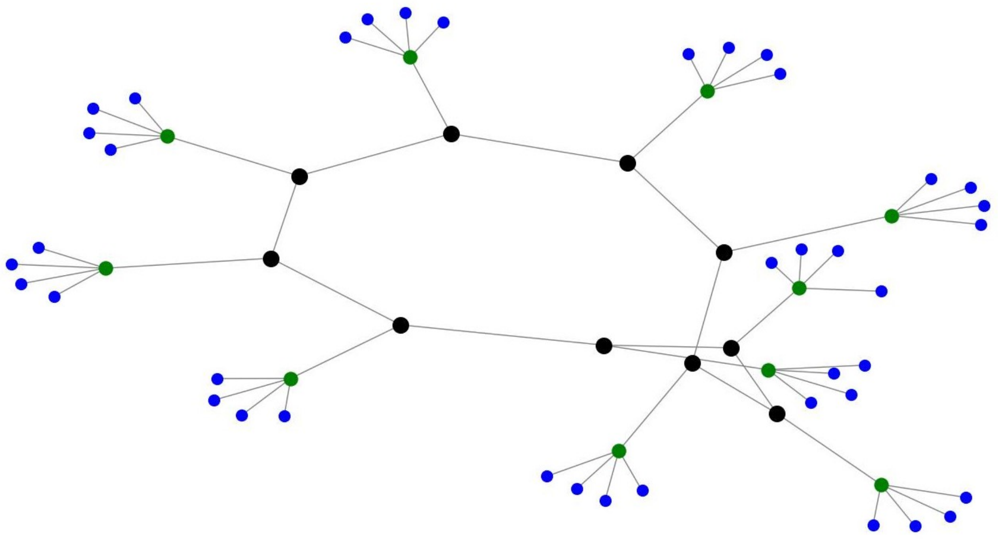


Figure 3. Topology in simulation experiment.

This study conducted simulation experiments based on the resource requirement and computing node resource information provided by the Alibaba Open Cluster Trace Program [57], an open-source dataset from the Alibaba Cloud production environment. The project collects resource data from each computing node every 15 min and records the time-shared resource demands of applications. This data format aligns with the experimental demands for resource requirements and computing node resource information, making this dataset suitable for evaluation in this study. The dataset includes information on CPU, memory, and disk resources. However, as historical data for disk usage are not included, this experiment focuses solely on CPU and memory resource information for performance analysis. Additionally, the experiment utilized data spanning ten consecutive calendar days and calculated the average resource capacity distribution for each computing node over a 24-h period. To facilitate more effective resource clustering, our experiment set intervals of 10 cores and 20 GB as the capacity range divisions for CPU and memory, respectively. It is assumed that the clustering model is updated at 00:00 each day.

Under these assumptions, resource requirements of 50, 100, 150, 200, 250, 300, 350, 400, 450, and 500 were generated based on the request information provided by the dataset. These demands were randomly distributed to the scheduling nodes in the topology. Computing nodes from the dataset were randomly assigned to the gateway nodes, which reported resource information to the gateway nodes. The gateway nodes then calculated the representative resource capacity distributions and reported them to the scheduling nodes. Resource status information was propagated among scheduling nodes to form a resource status information table. Each scheduling node used this table to match computing nodes to computing demands, completing the resource scheduling process. The specific parameters involved in the experiment are detailed in Table 1.

Table 1. Simulation configuration.

Parameter	Description
Dataset	https://github.com/alibaba/clusterdata (accessed on 1 January 2018)
Topology	NetworkX
Number of SNs	10
Number of GNs	10
Number of CNs	4000
Number of Resource Requirements	50, 100, 150, 200, 250, 300, 350, 400, 450, 500
Resource	CPU & Memory
Cycle Time	24 h
CPU Partitioning Interval	10 c
Memory Partitioning Interval	20 G

Using the experimental parameters outlined above, we employed Python 3.7.4 as the primary simulation tool, along with relevant Python libraries, to implement the resource aggregation algorithm proposed in this paper. The experimental results show that the average execution time of the proposed method is 0.58 s, demonstrating the algorithm's very low computational overhead.

5.2. Accuracy in Resource Matching

5.2.1. Experimental Comparison

In this study, to evaluate the effectiveness of the proposed method, we selected three resource matching methods for experimental comparison. These methods determine which computing nodes can meet application demands by comparing the application's resource requirements with either static or dynamic resource information:

- Static Information Resource Matching Method (SIRM): This method performs resource matching based solely on the resource capacity data at the current moment. Specifically, it determines whether a computing node meets the application's resource requirement based on its most recent resource capacity information;
- Resource Capacity Distribution Matching Method (RCDM): This method maintains a complete resource capacity distribution for each computing node and uses these data for resource matching. The specific matching method is detailed in Section 4.1.3. Additionally, this method serves as a benchmark to assess the impact of information loss due to resource aggregation on matching accuracy;
- Representative Resource Capacity Distribution Matching Method (RRCDM): Based on the resource aggregation method proposed in this paper, this method clusters computing nodes into resource groups according to the similarity of their resource

capacity distributions. It then calculates a representative resource capacity distribution for each group and determines whether the computing nodes within the group meet the application's resource requirements based on the representative distribution.

5.2.2. Evaluation Metrics

To evaluate the effectiveness of the resource matching methods, we used the following two evaluation metrics:

- Accuracy: In this paper, matching accuracy is defined as the ratio of the number of computing nodes that meet the actual resource requirements of an application to the total number of matched nodes [49]. Specifically, the "actual resource requirements" refer to the time-varying resource demands of an application over a 24-h period. This metric is used to evaluate the accuracy of the resource matching achieved by the proposed method, and it is calculated using the following formula:

$$Accuracy = \frac{n_{acc}}{n_{tot}} \tag{17}$$

  where n_{acc} represents the number of nodes that satisfy the actual resource requirements of the application and n_{tot} represents the total number of nodes matched. To determine whether a computing node meets the actual resource requirements of an application, the time-varying resource demand $D(t)$ over 24 h is compared with the time-varying resource status $C(t)$ of the computing node at each time point. If $C(t) \geq D(t)$, $\forall t \in [0, 24\text{ h}]$, the computing node is counted in n_{acc}. This means that a computing node is only included in n_{acc} if it can meet the application's time-varying resource demands at every sampling time t throughout the entire 24-h period.
  Accuracy reflects how many of the matched nodes actually meet the application's time-varying resource requirements, providing an effective measure of the performance of the matching method.
- Coverage Rate: This metric evaluates the ability of the resource matching method to identify eligible nodes and is defined as

$$Coverage\ Rate = \frac{n_{acc}}{N_{acc}} \tag{18}$$

  where N_{acc} is the number of nodes among all those available that can meet the actual application requirements throughout the 24-h period. Coverage rate indicates the proportion of all eligible nodes that the resource matching method successfully identifies, thus assessing the method's ability to recognize nodes that meet the criteria.

5.2.3. Matching Accuracy

Figure 4 presents a comparison of the *Accuracy* among the SIRM, the RCDM, and the RRCDM under different levels of resource requests. This comparison covers the matching accuracy of the three methods when matching CPU resources alone, memory resources alone, and both CPU and memory resources simultaneously.

As shown in Figure 4, the matching accuracy of RRCDM and RCDM ranges between 81% and 98%, while the accuracy of SIRM is only between 33% and 78%. RRCDM and RCDM show a clear advantage, indicating that considering the dynamic nature of computing resources can significantly improve matching accuracy. Additionally, Figure 4 shows a slight decrease in the matching accuracy of the proposed RRCDM method when both CPU and memory resources are matched simultaneously. This slight reduction is attributed to the influence of multi-dimensional information on the clustering process. Furthermore, the accuracy of the RRCDM method ranges from 81% to 95%, which is close to the 83% to 98% accuracy range of RCDM, suggesting that resource matching using the aggregated information obtained by the proposed RRCDM method has a minimal impact on matching accuracy.

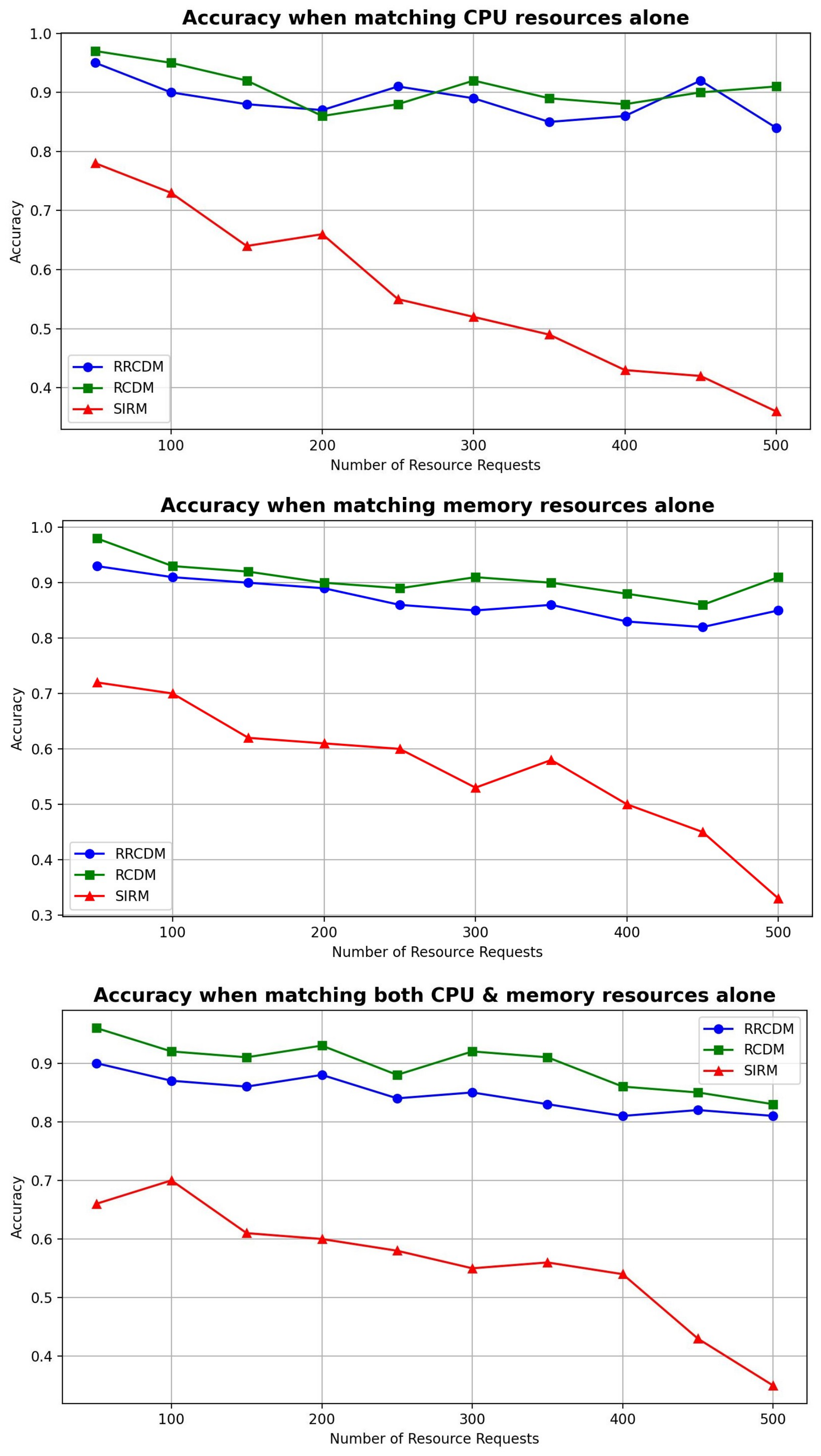


Figure 4. Matching accuracy.

5.2.4. Coverage Rate

Figure 5 presents a comparison of the *CoverageRate* among the SIRM, the RCDM, and the RRCDM under different levels of resource requests. This comparison covers the coverage rate of the three methods when matching CPU resources alone.

In addition, this experiment selects three applications with CPU requirements of 5 cores, 10 cores, and 15 cores, respectively, to evaluate the three resource matching methods. For each method, we statistically analyze the number of nodes that can meet the application requirements for 5 cores, 10 cores, and 15 cores (n_{acc}); the number of nodes that fail to meet these requirements (n_{unacc}); and the total number of nodes that can meet these requirements across all nodes (N_{acc}). As shown in Figure 5, a further comparative analysis is conducted.

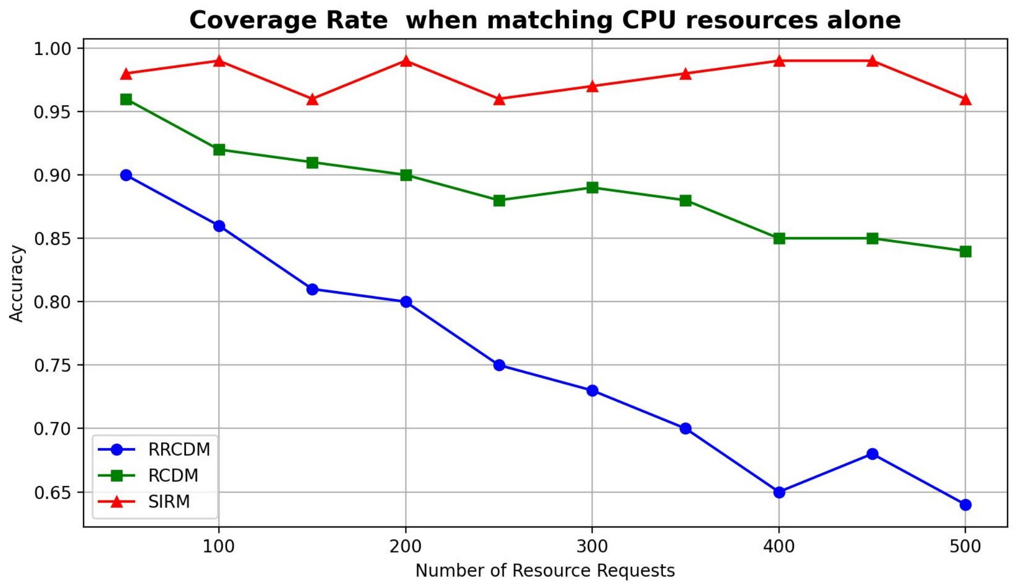


Figure 5. Comparison of matched computing node counts.

As shown in Figure 5, under different numbers of resource requests, SIRM has the highest coverage rate, followed by RCDM, with RRCDM having the lowest coverage rate. SIRM relies on static resource information for matching, which imposes fewer requirements on the nodes. As a result, it can match nearly all computing nodes that meet the requirements. However, it also matches many nodes that do not meet the requirements, leading to high values of both n_{acc} and n_{unacc}. This results in SIRM having low matching accuracy but a high coverage rate. RRCDM, on the other hand, uses dynamic resource information for matching, with stricter criteria for selecting computing nodes. Due to the information loss caused by aggregation, it misses some nodes that could actually meet the resource requirements. As shown in Figure 6, the number of nodes missed by RRCDM that meet the demand typically ranges between 5 and 15, while the number of unmatched nodes incorrectly identified by SIRM can reach up to 115. The impact of missing nodes in RRCDM is far less significant than the negative impact of SIRM matching nodes that do not meet the requirements, suggesting that the effect of RRCDM's omissions is smaller. From the application's perspective, the quality of node selection is primarily influenced by accuracy. Once an allocation decision is made, higher accuracy ensures the stability of application performance.

Based on the analysis of matching accuracy and coverage rate, it is clear that the RRCDM method proposed in this paper can effectively guide resource matching, leading to the selection of computing nodes that meet resource requirements with higher matching accuracy.

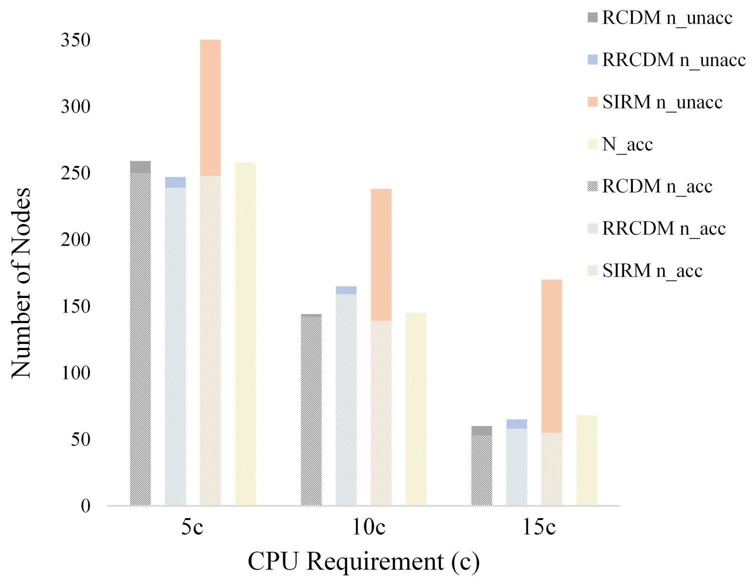


Figure 6. Comparison of matched computing node counts.

5.3. Analysis of Data Announcement Volume

In the proposed method, resource capacity distributions of computing nodes are used for resource matching. However, if the resource capacity distribution of every computing node is fully announced to the scheduling nodes in the network, it would result in a massive data announcement volume. To reduce this overhead, we propose a resource aggregation method, which compresses the amount of data transmitted by calculating the representative resource capacity distribution for each resource group.

5.3.1. Experimental Comparison

We conducted experimental comparisons using the SIRM-based information announcement method, the information announcement method based on full historical time-varying workload data, the RCDM-based information announcement method, and the RRCDM-based information announcement method.

- The SIRM-based information announcement method (SIRM): the announcement data only include the current CPU and memory capacity information;
- The information announcement method based on full historical time-varying workload data (AHDM): based on the dynamic resource allocation method in [17], the announcement data include full historical workload data announcements for each computing node;
- The RCDM-based information announcement method (RCDM): the announcement data include the resource capacity distribution information for each computing node;
- The RRCDM-based information announcement method (RRCDM): the announcement data include the representative resource capacity distribution information, aggregated using the method proposed in this paper.

5.3.2. Evaluation Metrics

- Announcement Volume: This metric compares the total data announcement volume (in bytes) under different numbers of computing nodes for the four methods;
- This metric calculates the saving rate of the SIRM, the RCDM, and the RRCDM compared to the AHDM. The formula is as follows:

$$Saving\ rate = \frac{V_{baseline} - V_{method}}{V_{baseline}} \times 100\% \tag{19}$$

where $V_{baseline}$ represents the announcement volume using the AHDM and V_{method} represents the announcement volume when using other announcement methods.

The saving rate reflects the percentage of announcement volume saved relative to the baseline volume.

5.3.3. Volume of Data Announcement

Figure 6 shows a comparison of the total data Announcement volume for the four methods under varying numbers of computing nodes.

As shown in Figure 7, the information announcement method based on full historical time-varying workload data transmits the complete resource usage information of nodes over the past 24 h at each sampling time, resulting in a significantly higher announcement volume compared to the other three methods. The advantage of this method lies in its provision of detailed historical data, which effectively supports dynamic resource scheduling. However, the trade-off is its high network communication overhead, making it unsuitable for large-scale node scenarios. The RCDM-based resource information announcement method, which models dynamic resource information using capacity distribution, has a slightly lower announcement volume than the full historical time-varying workload data method. Nevertheless, since it does not compress the number of computing nodes, the announcement volume remains relatively high. The RRCDM-based aggregated resource information announcement method proposed in this paper demonstrates effective data compression, with an announcement volume significantly lower than the full historical time-varying workload data method. Across different scales of computing nodes, this method reduces data announcement volume by an average of approximately 77.1% compared to the AHDM method. In this experiment, the proposed aggregation method exhibited the lowest total data announcement volume in all node number scenarios, even lower than the SIRM-based information announcement method, indicating its remarkable effectiveness in reducing communication overhead.

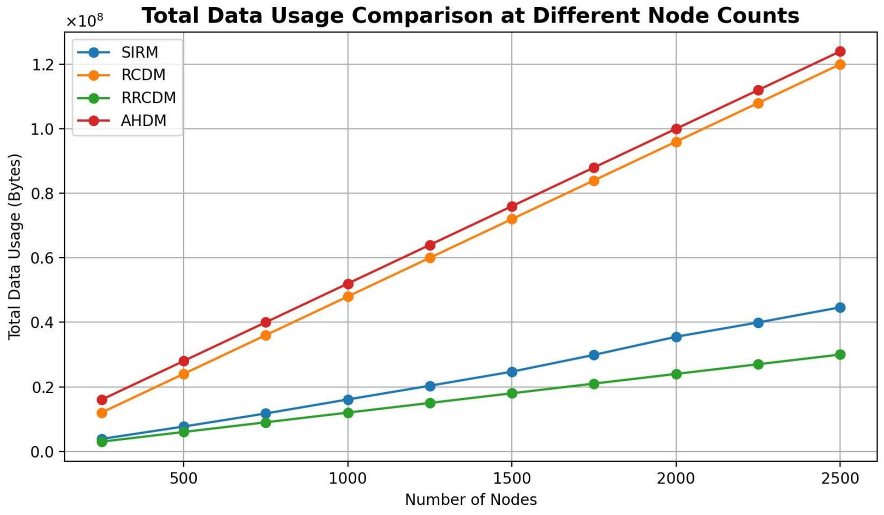


Figure 7. Total data usage comparison at different node counts.

5.3.4. Saving Rate

Figure 8 shows a comparison of the saving rates of the SIRM-based information announcement method, the RCDM-based resource information announcement method, and the RRCDM-based aggregated resource information announcement method under different numbers of computing nodes.

The results for the saving rates further demonstrate the advantages of the SIRM, RCDM, and RRCDM methods over the AHDM method in terms of announcement volume. Experimental results indicate that, compared to the data announcement volume of the AHDM method, the SIRM, RCDM, and RRCDM methods all save announcement data to varying degrees. Among them, the RRCDM-based information announcement method proposed in this paper achieves the highest saving rate, with an average saving of approximately 77.1%, and this rate remains stable as the number of nodes increases, suggesting that it can effectively compress information announcement volume even in large-scale networks. The experimental results show that the RRCDM method proposed in this paper significantly reduces network communication overhead by aggregating the resource capacity distributions of nodes while maintaining the accuracy of resource scheduling. In comparison, the RCDM method can also reduce the announcement volume to some extent, but its lack of compression of the number of computing nodes limits its applicability in large-scale networks; as the number of nodes increases, its saving rate gradually decreases. The SIRM method has certain advantages in small-scale node scenarios, but its scalability is affected as the number of nodes increases.

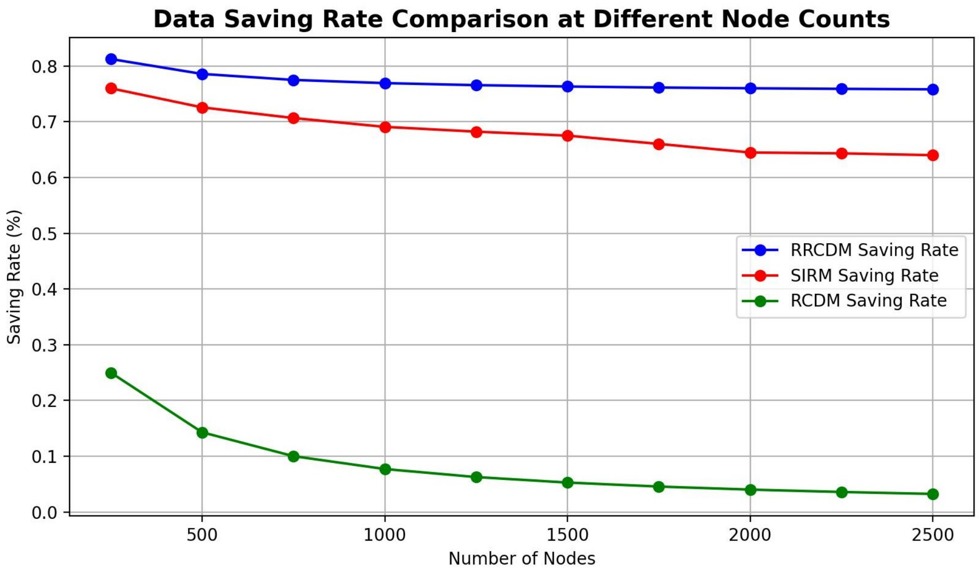


Figure 8. Data saving rate comparison at different node counts.

5.4. Robustness Analysis of the Model

We selected five calendar days to evaluate the stability and adaptability of the clustering model in response to variations in resource characteristics. We randomly sampled 1000 computing nodes for experimentation on each day. By analyzing the trends in the resource capacity distribution of the computing nodes, we determined the number of clusters that could be formed for each day. Subsequently, the clustering algorithm proposed in this paper was applied to cluster the computing nodes for each day, and the clustering results for different days were compared. To assess the clustering performance, we used Clustering Purity [58] as an evaluation metric to observe the model's performance under varying resource characteristics and load conditions. The experimental results are shown in Table 2.

Table 2. Clustering purity.

Day	The Number of Clusters	Purity
1	5	97.2%
2	5	97.3%
3	6	96.4%
4	8	95.8%
5	9	94.5%

The results show that the clustering model maintained a high level of clustering purity across the five calendar days. Although there were slight fluctuations in purity under different resource characteristics of computing nodes, it remained almost stable. This indicates that the clustering model proposed in this paper demonstrates good adaptability and stability under various resource characteristics and load conditions. This further validates the robustness of the model when dealing with diverse resource characteristics.

5.5. Information Loss Caused by Aggregation

In this paper, we aggregate computing nodes by constructing resource groups to reduce the amount of information that needs to be announced. However, this aggregation can introduce information loss, impacting the matching accuracy. In this section, we discuss the information loss resulting from the proposed aggregation method.

We use the Kolmogorov-Smirnov (K-S) method to measure the information loss. The Kolmogorov-Smirnov test is a useful nonparametric hypothesis test primarily used to compare whether two sample distributions are the same. We use the K-S statistic to measure the difference between the resource capacity distribution of computing nodes and the representative resource capacity distribution. The K-S statistic is calculated using the formula $D_n = \sup_{C_r} |X(C_r) - \bar{X}(C_r)|$, where $X(C_r)$ represents the resource capacity distribution of any computing node and $\bar{X}(C_r)$ represents the representative resource capacity distribution of the resource group. The value of the statistic D_n ranges between 0 and 1, where a smaller value indicates higher similarity between the distributions and less information loss.

To assess our method, we evaluate it under different numbers of clustering categories for the following schemes: the proposed method (resource group aggregation) and a random method (random). The random method refers to randomly grouping computing nodes and calculating the representative resource capacity distribution for each group in the same way.

Figure 9 illustrates the average K-S statistic values for the evaluated methods. For the proposed method in this paper, the highest K-S statistic value observed was 0.28, which gradually decreased as the number of clusters increased, reaching 0.13 at 19 clusters. In contrast, the K-S statistic value for the random method remained around 0.7. As analyzed in previous sections, the information loss introduced by the proposed method has been shown to have a minimal impact on resource accuracy. The experimental results in this section further confirm that the information loss caused by the proposed aggregation method is relatively minor.

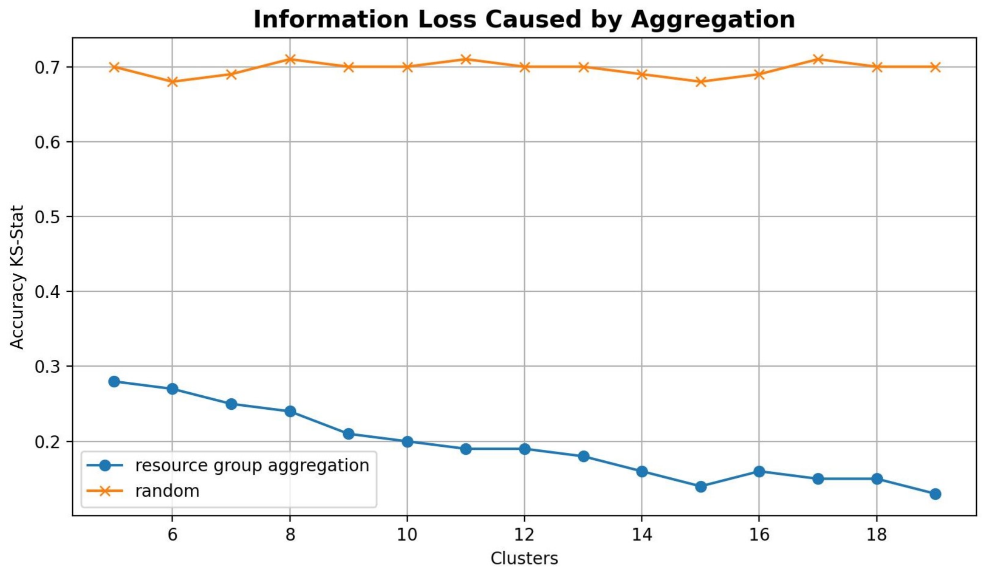


Figure 9. Information Loss Caused by Aggregation.

6. Conclusions

This paper addresses the challenge of dynamic resource aggregation in distributed computing networks. Given the highly dynamic characteristic of computing resources, relying solely on static resource information for scheduling presents significant drawbacks. To overcome this, we propose a novel resource aggregation method based on statistical capacity distributions. By formalizing resource requirements and accurately capturing the dynamic statistical capacity distributions of nodes, this method enables more precise matching of diverse computational demands. Additionally, by aggregating nodes with similar capacity distributions and computing a representative capacity distribution for each group, the method significantly reduces the data volume required for resource announcements. Experimental results and data analysis show that, compared to traditional resource matching methods based on static information, our approach enhances matching accuracy by 48% while substantially reducing announcement overhead. This demonstrates the potential of the proposed method as an effective solution for resource aggregation in distributed computing networks.

There are still challenges to address in dynamic resource aggregation. For example, while this paper proposes a clustering-model-based method for constructing resource groups, the frequency and scale of model updates can also impact both matching accuracy and announcement overhead. Frequent updates may lead to a large volume of data announcements, increasing network load, while infrequent updates could result in suboptimal scheduling decisions and reduced matching accuracy. Therefore, optimizing the efficiency of resource information updates remains a key research direction. Additionally, exploring multidimensional resource aggregation is another area worth investigating. In this paper, each dimension of computing resources was aggregated separately. Future work could explore combining multidimensional resources into a single aggregation approach and testing with more diverse datasets for further validation.

Author Contributions: Conceptualization, Y.W., J.Y. and Y.L.; methodology, Y.W., J.Y. and Y.L.; software, Y.W.; validation, Y.W.; formal analysis, Y.W.; investigation, Y.W.; writing—original draft preparation, Y.W.; writing—review and editing, J.Y. and Y.L.; visualization, Y.W.; supervision, J.Y. and Y.L.; project administration, J.Y. and Y.L.; funding acquisition, J.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key R&D Program of China: Application Demonstration of Polymorphic Network Environment for computing from the eastern areas to the western (Project No. 2023YFB2906404).

Data Availability Statement: Publicly available datasets were analyzed in this study. These data can be found here: https://github.com/alibaba/clusterdata (accessed on 1 January 2018).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Li, N.; Sun, Q.; Li, X.; Guo, F.; Huang, Y.; Chen, Z.; Yan, Y.; Peng, M. Towards the Deep Convergence of Communication and Computing in RAN: Scenarios, Architecture, Key Technologies, Challenges and Future Trends. *China Commun.* **2023**, *20*, 218–235. [CrossRef]
2. Tang, X.; Cao, C.; Wang, Y.; Zhang, S.; Liu, Y.; Li, M.; He, T. Computing power network: The architecture of convergence of computing and networking towards 6G requirement. *China Commun.* **2021**, *18*, 175–185. [CrossRef]
3. Lei, B.; Zhou, G. Exploration and practice of Computing Power Network(CPN) to realize convergence of computing and network. In Proceedings of the 2022 Optical Fiber Communications Conference and Exhibition (OFC), San Diego, CA, USA, 6–10 March 2022; pp. 1–3.
4. Chen, X.; Guo, Y.; Tan, B.; Huang, G. Open Service Interconnection Architecture Based Upon Computing and Network Convergence. *Inf. Commun. Technol.* **2022**, *16*, 53–59.
5. Zhou, D.; Liu, C.; Wang, Z.; Ding, Y.; Fang, Y. Research on demand side response computing power network scheduling strategy. *Telecommun. Eng. Technol. Stand.* **2024**, *37*, 36–41. [CrossRef]
6. Guo, L.; Guo, F.; Peng, M. A Novel Routing Method for Dynamic Control in Distributed Computing Power Networks. *Digit. Commun. Netw.* **2024**, *in press*.
7. Du, Z.; Li, Z.; Duan, X.; Wang, J. Service Information Informing in Computing Aware Networking. In Proceedings of the 2022 International Conference on Service Science (ICSS), Zhuhai, China, 13–15 May 2022; pp. 125–130.
8. Yao, H.; Lu, L.; Duan, X. Architecture and key technologies for computing-aware networking. *ZTE Commun. J.* **2021**, *27*, 7–11.
9. Huang, G.; Shi, W.; Tan, B. Computing power network resources based on SRv6 and its service arrangement and scheduling. *ZTE Commun. J.* **2021**, *27*, 23–28.
10. Li, K.; Zhang, X.; Wang, W. Multipath Information Announcement Algorithm for Computing Power Network based on Self-Organizing Map Network. In Proceedings of the 2023 IEEE/CIC International Conference on Communications in China (ICCC), Dalian, China, 10–12 August 2023; pp. 1–5.

11. Iorio, M.; Risso, F.; Palesandro, A.; Camiciotti, L.; Manzalini, A. Computing Without Borders: The Way Towards Liquid Computing. *IEEE Trans. Cloud Comput.* **2023**, *11*, 2820–2838. [CrossRef]
12. Santos, J.; Zaccarini, M.; Poltronieri, F.; Tortonesi, M.; Sleianelli, C.; Di Cicco, N.; De Turck, F. Efficient Microservice Deployment in Kubernetes Multi-Clusters through Reinforcement Learning. In Proceedings of the NOMS 2024—2024 IEEE Network Operations and Management Symposium, Seoul, Republic of Korea, 6–10 May 2024; pp. 1–9.
13. Kee, Y.S.; Logothetis, D.; Huang, R.; Casanova, H.; Chien, A. Efficient resource description and high quality selection for virtual grids. In Proceedings of the CCGrid 2005, IEEE International Symposium on Cluster Computing and the Grid, Cardiff, UK, 9–12 May 2005; Volume 1, pp. 598–606.
14. Chen, Y.; Liu, Z.; Zhang, Y.; Wu, Y.; Chen, X.; Zhao, L. Deep Reinforcement Learning-Based Dynamic Resource Management for Mobile Edge Computing in Industrial Internet of Things. *IEEE Trans. Ind. Inform.* **2021**, *17*, 4925–4934. [CrossRef]
15. Etemadi, M.; Ghobaei-Arani, M.; Shahidinejad, A. Resource provisioning for IoT services in the fog computing environment: An autonomic approach. *Comput. Commun.* **2020**, *161*, 109–131. [CrossRef]
16. Yadav, R.; Zhang, W.; Kaiwartya, O.; Song, H.; Yu, S. Energy-Latency Tradeoff for Dynamic Computation Offloading in Vehicular Fog Computing. *IEEE Trans. Veh. Technol.* **2020**, *69*, 14198–14211. [CrossRef]
17. Hu, J.; Gu, J.; Sun, G.; Zhao, T. A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment. In Proceedings of the 2010 3rd International Symposium on Parallel Architectures, Algorithms and Programming, Liaoning, China, 18–20 December 2010; pp. 89–96. [CrossRef]
18. Guan, X.; Choi, B.Y.; Song, S. Energy efficient virtual network embedding for green data centers using data center topology and future migration. *Comput. Commun.* **2015**, *69*, 50–59. [CrossRef]
19. Guo, C.; Li, Y.; Yan, Y.; Chen, W.; Bose, S.K.; Shen, G. Efficiently Consolidating Virtual Data Centers for Time-Varying Resource Demands. *IEEE Trans. Cloud Comput.* **2022**, *10*, 1751–1764. [CrossRef]
20. Gong, X.; Bai, C.; Ren, S.; Wang, J.; Wang, C. A Survey of Compute First Networking. In Proceedings of the 2023 IEEE 23rd International Conference on Communication Technology (ICCT), Wuxi, China, 20–22 October 2023; pp. 688–695.
21. China Unicom Network Technology Research Institute. *White Paper of Computing Power Network Architecture and Technology System*; Technical Report; China Unicom Network Technology Research Institute: Beijing, China, 2019.
22. China Mobile. *Computing-Aware Networking White Paper*; Technical Report; China Mobile: Beijing, China, 2019.
23. Huawei Technologies Co., Ltd. *Computing Internet Technology White Paper*; Technical Report; Huawei: Shenzhen, China, 2021.
24. Wang, J.; Chen, G.; You, J.; Sun, P. Seanet: Architecture and technologies of an on-site, elastic, autonomous network. *J. Netw. New Media* **2020**, *6*, 1–8 .
25. Duan, X.; Yao, H.; Fu, Y.; Lu, L.; Sun, T. Computing force network technologies for computing and network integration evolution. *Telecommun. Sci.* **2021**, *37*, 76.
26. China Communications Standardization Association. *Research on Key Technologies of Computing Power Aware Network for the Whole Network*; Technical Report; China Communications Standardization Association: Beijing, China, 2020.
27. Yao, H.; Geng, L. Trend of next generation network architecture:computing and networking convergence evolution. *Telecommun. Sci.* **2019**, *35*, 38–43.
28. Ni, Z.; You, J.; Li, Y. An ICN-Based On-Path Computing Resource Scheduling Architecture with User Preference Awareness for Computing Network. *Electronics* **2024**, *13*, 933. [CrossRef]
29. China Institute of Communications. *Terminal Computing Aware Network White Paper*; Technical Report; China Institute of Communications: Beijing, China, 2022.
30. Edge Computing Consortium, Network 5.0 Industry and Technology Innovation Alliance. *Operator Edge Computing Network Technology White Paper*; Technical Report; Edge Computing Consortium: Beijing, China, 2019.
31. Yao, H.; Duan, X.; Fu, Y. A computing-aware routing protocol for Computing Force Network. In Proceedings of the 2022 International Conference on Service Science (ICSS), Zhuhai, China, 13–15 May 2022; pp. 137–141.
32. Zhang, X.; Li, K.; Lei, B.; Cui, F.; Ni, M.; Zhou, T. Computing Resource Information Sensing and Announcement System and Method in Computing Network. Patent CN115118647B, 20 May 2022.
33. Guan, C. Research on Service Routingmechanism in Computing Forcenetwork. Master's Thesis, Southeast University, Dhaka, Bangladesh, 2022.
34. Ungureanu, O.M.; Vlădeanu, C.; Kooij, R. Collaborative Cloud—Edge: A Declarative API orchestration model for the NextGen 5G Core. In Proceedings of the 2021 IEEE International Conference on Service-Oriented System Engineering (SOSE), Oxford, UK, 23–26 August 2021; pp. 124–133.
35. Theodoropoulos, T.; Rosa, L.; Boudi, A.; Benmerar, T.; Makris, A.; Taleb, T.; Cordeiro, L.; Tserpes, K.; Song, J. Cross-Cluster Networking to Support Extended Reality Services. *IEEE Netw.* **2024**, 1. [CrossRef]
36. Tamiru, M.A.; Pierre, G.; Tordsson, J.; Elmroth, E. mck8s: An orchestration platform for geo-distributed multi-cluster environments. In Proceedings of the 2021 International Conference on Computer Communications and Networks (ICCCN), Athens, Greece, 19–22 July 2021; pp. 1–10.
37. Nguyen, T.N.; Lee, J.; Vitumbiko, M.; Kim, Y. A Design and Development of Operator for Logical Kubernetes Cluster over Distributed Clouds. In Proceedings of the NOMS 2024—2024 IEEE Network Operations and Management Symposium, Seoul, Republic of Korea, 6–10 May 2024; pp. 1–6.

38. Faticanti, F.; Santoro, D.; Cretti, S.; Siracusa, D. An Application of Kubernetes Cluster Federation in Fog Computing. In Proceedings of the 2021 24th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), Paris, France, 1–4 March 2021; pp. 89–91.
39. Karmada: Open Multi-Cloud Multi-Cluster Kubernetes Orchestration. Available online: https://karmada.io/ (accessed on 5 September 2023).
40. Zeydan, E.; Mangues-Bafalluy, J.; Baranda, J.; Martínez, R.; Vettori, L. A Multi-criteria Decision Making Approach for Scaling and Placement of Virtual Network Functions. *J. Netw. Syst. Manage.* **2022**, *30*, 32. [CrossRef]
41. Chang, X.; Jiao, W. Improvement and Implementation of Kubernetes Resource Scheduling Algorithm. *Comput. Syst. Appl.* **2020**, *29*, 256.
42. Cheng, X.; Su, S.; Zhang, Z.; Wang, H.; Yang, F.; Luo, Y.; Wang, J. Virtual network embedding through topology-aware node ranking. *SIGCOMM Comput. Commun. Rev.* **2011**, *41*, 38–47. [CrossRef]
43. Fajjari, I.; Aitsaadi, N.; Pujolle, G.; Zimmermann, H. VNR Algorithm: A Greedy Approach for Virtual Networks Reconfigurations. In Proceedings of the 2011 IEEE Global Telecommunications Conference—GLOBECOM 2011, Houston, TX, USA, 5–9 December 2011; pp. 1–6. [CrossRef]
44. Li, D.; Hong, P.; Xue, K.; Pei, j. Virtual Network Function Placement Considering Resource Optimization and SFC Requests in Cloud Datacenter. *IEEE Trans. Parallel Distrib. Syst.* **2018**, *29*, 1664–1677. [CrossRef]
45. Zheng, K.; Wang, X.; Li, L.; Wang, X. Joint power optimization of data center network and servers with correlation analysis. In Proceedings of the IEEE INFOCOM 2014—IEEE Conference on Computer Communications, Toronto, ON, Canada, 27 April–2 May 2014; pp. 2598–2606. [CrossRef]
46. Ahmed, M.; Seraj, R.; Islam, S. The k-means Algorithm: A Comprehensive Survey and Performance Evaluation. *Electronics* **2020**, *9*, 1295. [CrossRef]
47. Murtagh, F.; Contreras, P. Algorithms for hierarchical clustering: An overview, II. *WIREs Data Min. Knowl. Discov.* **2017**, *7*, e1219. [CrossRef]
48. Chauhan, R.; Batra, P.; Chaudhary, S. A Survey of Density Based Clustering Algorithms. *Int. J. Comput. Sci. Technol.* **2014**, *5*, 169–171.
49. Chai, R.N.; Gao, S.; Lan, J.Y.; Liu, N.C. Efficient Computing Resource Metric Method in Computing-First Network. *J. Comput. Res. Dev.* **2023**, *60*, 763–771. [CrossRef]
50. Melnykov, V. Challenges in model-based clustering. *WIREs Comput. Stat.* **2013**, *5*, 135–148. [CrossRef]
51. Ståhl, D.; Sallis, H.M. Model-based cluster analysis. *WIREs Comput. Stat.* **2012**, *4*, 341–358. [CrossRef]
52. Xiao, Z.; Song, W.; Chen, Q. Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment. *IEEE Trans. Parallel Distrib. Syst.* **2013**, *24*, 1107–1117. [CrossRef]
53. Gulisano, V.; Jiménez-Peris, R.; Patiño-Martínez, M.; Soriente, C.; Valduriez, P. StreamCloud: An Elastic and Scalable Data Streaming System. *IEEE Trans. Parallel Distrib. Syst.* **2012**, *23*, 2351–2365. [CrossRef]
54. Chen, Y.; Alspaugh, S.; Katz, R. Interactive analytical processing in big data systems: A cross-industry study of MapReduce workloads. *Proc. VLDB Endow.* **2012**, *5*, 1802–1813. [CrossRef]
55. Mishra, A.K.; Hellerstein, J.L.; Cirne, W.; Das, C.R. Towards characterizing cloud backend workloads: Insights from Google compute clusters. *SIGMETRICS Perform. Eval. Rev.* **2010**, *37*, 34–41. [CrossRef]
56. Benson, T.; Anand, A.; Akella, A.; Zhang, M. Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.* **2010**, *40*, 92–99. [CrossRef]
57. Alibaba. Alibaba Open Trace. Available online: https://github.com/alibaba/clusterdata (accessed on 1 January 2018).
58. Zhu, M. Research on Intelligent Signal De-Interleaving and Behavior Recognition of Advanced Radar. Ph.D. Thesis, Beijing Institute of Technology , Beijing, China, 2022.