*Journal of Network Intelligence*
**Taiwan Ubiquitous Information**
 ISSN 2414-8105 (Online)
Volume **9**, Number **2**, May **2024**

# Optimization Algorithm for Computing Power Resource Scheduling Based on Container Cluster Deployment

Wei Fang*

China Mobile Communications Group Zhejiang Co., Ltd, network management center
Hangzhou, Zhejiang, China, 310000
fangwei@zj.chinamobile.com

Jie Wu

China Mobile Communications Group Zhejiang Co., Ltd, network management center
Hangzhou, Zhejiang, China, 310000
wujie21@zj.chinamobile.com

Xiaoguang Luo

China Mobile Communications Group Zhejiang Co., Ltd, network management center
Hangzhou, Zhejiang, China, 310000
luoxiaog@zj.chinamobile.com

*Corresponding author: Wei Fang

Received July 17, 2023, revised October 11, 2023, accepted January 2, 2024.

Abstract. *Currently, the demand for computing power in data processing is increasing rapidly, and traditional data centers or terminal devices are no longer able to meet the computing power needs of businesses. This study proposes an Optimization Algorithm of Computing Resource Scheduling based on Container Cluster Deployment (CRS-CCD). Firstly, optimize the deployment of container clusters based on cloud data centers, and implement cost control for the construction of cloud data centers and communication networks. In addition, the parallel computing power optimization framework based on improved kernel functions optimizes the steady-state safe operation of classical computing power systems. CUDA function and GPU framework are used to improve the Algorithmic efficiency, and container resource cluster is used to optimize the deployment of computing resources. To validate the proposed algorithm, in a real computing resource network environment, compare the proposed computing resource scheduling optimization algorithm based on container cluster deployment with other benchmark algorithms. By comparing the area value under the curve and average accuracy results on the computing power network data set, it is not difficult to see that the computational resource scheduling optimization algorithm based on container cluster deployment proposed in this paper has a high average accuracy value, which is more efficient and accurate than other benchmark algorithms for optimizing computing resources, and also reduces the overall loss of the computing resource network.*
**Keywords:** container cluster deployment; Computational resources; Distributed scheduling; optimization algorithm

1. **Introduction.** As a standard software unit, containers package code and corresponding dependency files, enabling applications to quickly, reliably, and conveniently transfer from one computing environment to another, and run directly in the new computing environment [1, 2]. Early container instances utilized the Chroot tool in the Unix operating system to achieve resource isolation and restriction, relying on

Chroot to switch the root directory used by the process to a custom file directory, set access permissions for this file directory, and complete the isolation of file resources [3]. Containers, like other virtualization technologies, are also a collective term. Currently, container technologies include Docker, Rocket, LXC, and Warden [4, 5, 6]. Scholars have relied on K8S for the entire hybrid system, by containerizing both online and offline applications and hosting them in K8S, and utilizing methods such as application classification, scheduling, and isolation to ensure that online and offline applications share the same resource pool without affecting the quality of online service [7, 8], thereby improving resource utilization [9]. The development of computing power resources requires not only large-scale data centers to ensure the supply of computing power, but also efficient calculation of priority computing power resources for computing power management, achieving unified management, unified arrangement, intelligent scheduling, and improving computing power efficiency. Therefore, a unified computing power base is indispensable [10, 11].

Initially, Docker, a container technology under the open-source cloud service provider Dot Cloud, was developed using Go language and relied on host kernel namespace and cgroups technology to isolate file resources and limit resource usage. Docker container technology unified container packaging standards and became a widely used and mature container technology [12]. The container instances generated by Docker container technology are called Docker containers [13]. Docker containers are widely used to host virtual network elements in NFV architectures. In order to improve the security of containerized virtual network elements, a 5G mobile communication network with high security level is implemented [14, 15, 16]. Compared to virtual machines, Docker containers have advantages such as lighter weight, smaller size, faster startup speed, higher resource utilization efficiency, and more flexible scheduling. And both the Docker image and container adopt a hierarchical storage structure. Without containing the complete Linux system kernel, the Docker image only contains the file system structure which is required by the Docker container [17]. The Docker image has the advantage of being composed of one or more mirror layers, and the Docker image is a read-only template; for Docker, an image layer is added on top of the original image to create or update old version images, including using dockfiles to build images and submitting new images through containers. The Docker container utilizes containers to run applications, which have stronger readability and can be used as a simple Linux operating system and run application programs [18].

In the optimization model of computing resources, the cluster management module of multiple clusters will separate the management cluster and business cluster in the multi cluster deployment architecture. For business clusters, multiple business clusters can also be labeled as a cluster group [19]. Through the resource distribution module of multiple clusters, build multiple cluster container resources in the management cluster and distribute them to various business clusters, and run the container resources in the business cluster. When the container starts, Docker assigns independent Name spaces to each container based on its configuration parameters [20]. By using PIDName space, the pid of the processes in the container can start at 1, making the container think it is running on a new independent server [21]. The processes in different containers are not visible to each other and cannot communicate with each other, resulting in excellent isolation. From this perspective, it can be considered that each container is an isolated process group running on the host. Provide independent network IPs for containers through Network Namespaces, and achieve communication between containers, hosts, and external networks through technologies such as iptables [22]. By isolating the domain name and host name provided by UTSName space, each container has a separate domain name. The combination of MountName space technology and rootfs mounts specific directories on the host as the root directory of each container. Placing container image files in these host specific directories in advance can provide a file system for the isolated execution environment of container processes [23, 24].

This study is based on a programmable network computing power scheduling architecture and adopt virtualized container technology, targeting distributed clusters to control the cluster through containers and issue network control instructions to the data control surface; Send data forwarding strategy, data Routing table and other network forwarding protocols to the forwarding equipment on the data forwarding side through the controller; By accessing the computing cluster through data forwarding, the scheduling of computing power nodes and container loading can be achieved. Data forwarding based on programmable network architecture can change the scheduling of existing container resources within the data center, achieving distributed scheduling and optimization of computing power resources across data centers.

The research idea of this paper is to schedule computing power based on container cloud technology and optimize the container cluster deployed in the cloud data center to control the construction cost of cloud data center and communication network. This study optimizes the steady-state safe operation of the classical computing power system by improving the parallel computing power framework of the kernel function, proposes a computing resource scheduling optimization algorithm based on container cluster

deployment, and uses the CUDA function and GPU framework to improve the algorithm efficiency and realize the container cluster part. Efficient computing resource scheduling optimization. In order to test whether the CRS-CCD optimization algorithm proposed in this paper can be superior to other benchmark algorithms, this study uses the 2010, 2015, and 2020 computing power network datasets of Chinese listed companies, in order to verify the superior performance of the algorithm proposed in this study. The algorithm proposed in this study can provide an efficient example resource scheduling optimization solution for cloud container clusters, solving the problem of how to efficiently and reasonably allocate and deploy computing resources in cloud container clusters. This study has multiple innovations, proposing an innovative cloud data center deployment optimization model that not only uses improved kernel functions to optimize parallel computing frameworks, but also proposes a resource preemption scheduling method based on container resource cluster deployment to achieve reasonable and efficient optimization of computing resource allocation, and the algorithm performance is superior to many benchmark algorithms.

2. **Overview of Literature and Related Technologies.** As the cloud computing develops rapidly, edge computing and intelligent devices, computing power resources are being deployed everywhere. Due to the islanding effect of computing power, traditional network architectures cannot effectively utilize these distributed computing power resources [25, 26, 27]. In order to overcome these problems and improve the efficiency of network work, computational power networks have emerged. With the rapid development of the new generation of information technology, the demand for data resource storage, computing, and application has significantly increased. It is necessary to accelerate the evolution towards new data centers, build an intelligent computing ecosystem centered on new data centers, and play a role in empowering and driving the digital economy [28, 29]. The scale of intelligent computing power in China is growing rapidly, and the demand for artificial intelligence computing power is steadily increasing driven by technological policies. Traditional benchmarking methods such as degree center (DC), intermediate center (IC) and proximity center (CNC) are less efficient, unable to efficiently complete computing requirements, and have problems such as high network construction costs, resulting in resource network losses. Therefore, this paper proposes a computational resource scheduling optimization algorithm based on container cluster deployment (CRS-CCD). After verification, the optimization algorithm proposed in this paper can effectively enhance computing power and reduce the cost of cloud data center and communication network construction. Although container technology was initially mainly implemented based on operating system virtualization technology, with the continuous attention to container security issues, virtualization container technology has emerged [30]. So container technology currently mainly refers to virtualization technology with high resource utilization, fast start stop, and the use of Docker layered image format [31]. So container technology can be mainly divided into operating system level virtualization based container technology and hypervisor virtualization based container technology according to the specific implementation technology [32, 33].

A container is a collection of processes with view isolation, resource restrictability, and independent file systems. Its implementation mainly relies on Linux's underlying Cgroups, rootfs, and namespace technologies. Moreover, the container is a "single process" model with packaging "mirroring" technology, making it lightweight, agile, and portable, ensuring high consistency between the "local" and "cloud" environments. The container lifecycle is equivalent to the application lifecycle, which is the essence of the container's "single process" model [34]. The "single process" model does not fully represent that a container has only one process, but rather, within the container's namespace, the process with PID=1 seen by other processes is the application itself represented by the container; However, in principle, the container will be designed as a single process. Only when performing exec and other operations on the container, it will become a Child process of the process whose PID=1 in the container, which is assigned with the process number in the container, and also has the "real" process number on the host, which is the role of invisible boundaries. The single process model characteristic of containers is an important reason for determining agility and lightness, and another important reason is the implementation principle of container image packaging [35, 36].

The essence of Docker container mirroring is to encapsulate the system files that the application and its running environment rely on, which mainly relies on the Mount Namespace system function to implement. It can change the "mount point" of the container file system and provide a file system for the isolated execution environment of the container process [37, 38]. In addition, it is also a static view that only encapsulates the files, configurations, and directories contained in the operating system, and does not include the Linux operating system kernel, but rather shares a common host kernel; This also endows it with lightweight features at another level, as well as replicable and portable features [39, 40]. At the same time, rootfs packages files, configurations, and directories related to the entire operating system

of an application, which encapsulates the dependencies required for application operation, ensuring high consistency between local development and cloud execution environments. In addition, Docker mirroring has an important feature, which is its image layering mechanism [41, 42].

Container technology is ultimately just a low-level virtualization technology. When large-scale implementation practices are carried out, container clusters will emerge and further rise to the cloud level [43]. At this time, an orchestration tool is needed to provide operation and maintenance capabilities such as resource orchestration, scheduling, distributed management and routing gateway, horizontal scaling, monitoring and backup. Therefore, this paper selects the Kubernetes project to support the development of the "Autoscaling" system [44]. Kubernetes is an open source system which is used to deploy, extend, and manage containerized applications for automatically. Originally developed by Google, it aims to provide a better solution for managing distributed components and services across different infrastructures. Due to the continuous development of container technology, container layout technology is also constantly following suit. It is during these years of development that Kubernetes has become the standard and specification for container layout [45]. Different interfaces are provided, which can be compatible with various container engines such as Docker and Containered.

Kubernetes clusters are generally divided into two types of nodes, namely management nodes and work nodes. The management node is responsible for managing the entire Kubernetes cluster, while the work node is mainly responsible for running the container. Generally, three management nodes can form a highly available architecture. Various resources in the cluster are abstracted [46]. OpenStack is an open source platform that provides powerful virtual servers and services required for cloud computing. It mainly deploys infrastructure services, providing hardware tools and components for the processing, storage, and network resources of the entire data center. Besides, OpenStack can be understood as a software platform that uses pooled virtual resources to build and manage clouds, including the public cloud and the private cloud. OpenStack mainly exists in the form of containers on Kubernetes, and container based OpenStack deployment can be presented through a tree structure, where nodes represent a set of containers, and each leaf node represents a container. The implementation of OpenStack containerization mainly involves composing each component of OpenStack into a container set, which is composed of one or more subsets of containers. The latter is composed of one or more independent containers, and each container set provides an independent logical service. For container subsets, taking Nova services as an example, they are composed of independent containers such as Nova API and Nova Compute. The container set is managed as a unified unit.

3. **Model construction.** On the basis of computing power scheduling based on container cloud technology, the supplement and modification, the unexplained model parameters are consistent with the above model. The model of the algorithm resource $a_i$ is $\{D_i^{sum}, fwc_i, K_i\}$. Among them, $D_i^{sum}$ is the total business data flow, which represents the amount of data that business $a_i$ needs to transmit per unit time. It can also represents the storage and communication load of business $a_i$, which is as shown in Equation (1).

$$D_i^{sum} = \frac{\sum_{d_j \in D}(da_j * r_{i,j})}{\Delta t} \tag{1}$$

Among them, $r_{i,j}$ is the element in the correlation matrix and is a 0–1 variable. If $r_{i,j} = 1$, then the business $a_i$ is related to the data set $d_j$; if $r_{i,j} = 0$, the business $a_i$ is not related to the data set $d_j$. $da_j$ is the data volume of data sets, and $\Delta t$ is the time interval for service data transfer. $fwc_i$ represents the calculated load, whose specific form is the equipment computing resources occupied in the business processing process. While the matrix $K_i$ indicates that the association between the service and the site is $\{k_{i,1}, k_{i,2}, \ldots, k_{i,n}\}$. Among them, $k_{i,n}$ represents the relevance between the business $a_i$ and the site $n$, which can be shown as Equation (2).

$$k_{i,n} = \sum_{d_j \in D} (l_{j,n} * r_{i,j} * da_j) \tag{2}$$

Due to the fact that the optimization and deployment of cloud data centers consider businesses that are processed in the cloud data center, the security parameters of the business model are not considered. Due to the long-term scale of cloud data center optimization deployment, the model does not consider short-term business arrival characteristics and load fluctuations. The real-time nature of the business can be transformed into the requirements of the business load on the hardware resources of the cloud data center. The real-time nature of the business can be met by adjusting the capacity of the cloud data center in the optimization process. Therefore, the business model does not consider the delay threshold. The ordinary computing power node model represents the characteristics of the computing power Internet

of things service within the site [47], which is shown as $NM = \{N, K', \alpha^n, \beta^n\}$. When analyzing the optimization problem, you can replace the business model by the site model. $N = \{n_1, n_2, \ldots, n_m\}$ is a set of common sites, and all of the computing power sites are included. $K'$ represents a strong correlation between the business and the site. $k'_{i,j} \in 0, 1$ is the value of $k'_{i,j}$. For example, when $k'_{i,j} = 1$, the relevance between $a_i$ and site $n_j$ is maximum, so site $n_j$ can be regarded as the strong association site of business $a_i$. In order to transform the business model into a site model, it is stipulated that all the associated data sets of the business are stored in the strongly associated site of the business, and the business is processed in the cloud data center with a communication connection with the strongly associated site. $\alpha^n$ is the site calculation load, and $\beta^n$ is the total site data traffic, and the calculated load of the site $n_j$ can be calculated by Equation (3).

$$\{\alpha_j^n, \beta_j^n\} = \sum_{a_i \in A} (k'_{i,j} * fwc_i, D_i^{sum}) \tag{3}$$

To facilitate the analysis of cloud data center deployment optimization issues, this regulation simplifies the storage and invocation of data sets, transforms business characteristics into computing power node characteristics, and can replace business models with site models. In fact, for non local businesses, it is required that all associated data sets of the business are stored in the Strongly correlated material site of the business, which will lead to deviation from the actual situation. Based on the common site model, the computing force node model also includes features related to the deployment of cloud data centers, where $PM = \{P, K', \alpha^p, \beta^p, sp^p, L_C^p, L_S^p\}$, while $P$ represents the set of compute force nodes, including all the compute force nodes $P = \{p_1, p_2, \ldots, p_m\}$. $K'$ is the association matrix between services and sites, and the meaning and value method are the same as ordinary sites. $\alpha^p$ and $\beta^p$ represent the calculated load and total data flow of the computing node, respectively, in a similar way to the ordinary site. $sp^p$ is the price of the computing power node, and the economy of computing node deployment of cloud data center is reflected in the price of low computing power. $L_C^p$ and $L_S^p$ are the upper limits of the deployment of computing devices and storage devices for computing power nodes, reflecting the limitations of the site space, operation and maintenance management, supply and other conditions of computing power nodes on the deployment of equipment. Computing equipment mainly refers to the large processor of data center, and storage equipment mainly refers to the large memory of data center.

The optimal deployment of cloud computing data center is oriented to a long time scale, which belongs to the planning and construction of the power of computing Internet of Things. Therefore, the economy of cloud data center construction and long-term operation is taken as the optimization goal. The objective function is to minimize the total cost $C_o$, so the operating cost of the computing power node can be calculated as shown in Equation (4):

$$C_o = \sum_{p_i \in P, n_j \in N} sp^{p_i} \omega(\alpha^{n_j} \eta + \beta^{n_j} \rho \varepsilon) t * (\chi_{p_i}^{(} n_j) + \gamma_{p_i}^{p_j} + s^{p_i}) \tag{4}$$

The construction cost includes the construction cost of cloud data centers and communication networks. The construction cost of cloud data center is divided into the Fixed cost of building cloud data center and the capacity cost of deploying computing, storage and other equipment. The deployment optimization of the cloud data center does not change the business characteristics, the calculation and storage load of the business, the total equipment capacity of the cloud data center, and the total capacity cost. Therefore, only Fixed cost needs to be considered in the optimization goal. The total cost is shown in Equation (5), and Dis represents the length of built fiber lines between sites, and $\lambda$ represents the construction cost of fiber lines per unit length, both of which belong to the given condition.

$$C_c = \sum_{p_i \in P} s^{p_i} \varphi + Dis_{p_i}^{n_j} \lambda * (\chi_{p_i}^{n_j} + \gamma_{p_i}^{p_j} + \delta_{p_i}^{p_j}) \tag{5}$$

## 4. Power Scheduling Optimization Algorithm Based on Container Cloud Technology.

### 4.1. Parallel Computing Force Optimization Framework Based on the Improved Kernel Function.

Safety analysis plays an significant role in ensuring the stable and safe operation of computing power systems, usually meeting the principles of safety verification. As the scale of computing power networks continues to increase, the computational cost of considering various possible device disconnections is becoming increasingly high. Traditional security verification uses serial computing, which has low computational efficiency and cannot meet the needs of online applications. Therefore, there is an urgent need to study accelerated algorithms for computing power security analysis.

Let matrix $B_{dc}$ be a large sparse matrix. In order to save memory usage and speed up the calculation speed, adopt the sparse storage form, and $A_x$, $A_i$ and $A_p$ are used to express the value, column number

and line offset respectively. Since the disconnection affects only the four elements of $B_{dc}$ and does not affect the sparsity of the matrix, we only need to generate $A_x$ in batches. Use the function of CUDA to copy $A_x$ into $m+s$ copies in video memory, and create m threads to modify the first m $A_x$, and the latter s $A_x$ does not have to be modified. To avoid the delay caused by multiple access to the global memory, and also facilitate the data division when the fault set is too large, it is not necessary to divide the fine grain too fine, a thread can handle a break, that is, a thread is responsible for modifying the four elements of the conduction matrix.

CUDA function is used to call GPU's Compute kernel. Parameters represent grid dimension and thread block dimension, and represent the number of threads executed in each interrupt process. At this point, it is shown by Equation (6). According to the principle of rounding up, the number of startup blocks is determined. The thread configuration of all kernel functions in the article is as described above.

$$\text{“}num\_blocks\text{”} = \frac{m + \text{“}num\_threads\text{”} - 1}{\text{“}num\_threads\text{”}} \tag{6}$$

Take the modification of main diagonal element of $B_{dc}$ corresponding to node $i$ as an example, and its improved kernel function is shown in Algorithm 1. Firstly, index each thread number to the branch number, and then determine the node numbers $i$ and $j$ at both ends of the branch. Due to the fact that each node has been rearranged in the CPU in the order of balancing nodes. Therefore, when the node number is smaller than the balance node number, it is necessary to index to the specific location of $A_x$ that needs to be modified based on node number, $A_i$, and $A_p$, and then make modifications to it.

**Algorithm 1** Improved kernel function used by the model

**Input:** parameter $A_x$, $A_i$, $A_p$
**Output:** $A_x^{index}$

```
define thread ID
define i, j ← index to break branch number
define i, j ← node number corresponding to the broken branch
if b < m, i < balance node ID then
    let index ← b/n, offset ← A_p, A_i
    M[i] ← 1.0, M[j] ← −1.0
    if b < (m + s) * n then
        (A_x^index)_b = (A_x^index) − 1/(x_b k_b)
        x_1[(n + 1) * index + offset] = x[b]
    end if
end if
```

Since the sparsity of the force matrix $B_{dc}$ will not change significantly in multiple scenarios, the batch LU decomposition technology can be used to solve the sparse System of linear equations. The matrix rearrangement can not only reduce the injection of non-zero elements, but also increase the numerical stability, and only needs to be carried out once. There are many High-performance computing libraries for solving sparse System of linear equations, such as the NICSLU and PARDISO computing libraries for CPU, which are solved by the Left looking algorithm and the super node algorithm respectively; There are also cuSolver and cuSparse function libraries for GPUs. The cuSolverRF function library in the cuSolver function library is used to accelerate the solution of System of linear equations by fast decomposition when new coefficients are given in the same sparse mode. In this research, the cuSolverSP function library is used to carry out a matrix rearrangement and complete LU decomposition on the CPU, and the results are transmitted to the GPU. The cuSolverRF function library can be used to carry out batch LU decomposition, and quickly solve System of linear equations in parallel.

This study uses a GPU accelerated batch generation distribution factor calculation method to create large-scale concurrent threads and batch generate distribution factors for each branch under various operating conditions, achieving higher parallelism. The previous text has derived formulas for the impact of power line disconnection on active power, and the impact of power line disconnection on active power. When reading the ground state data in the CPU, the transmission line transformation ratio of the original parameter's power results is changed from 0 to 1, avoiding more judgments. Each m threads process a scene, and the molecules need to be indexed to the corresponding circuit number to obtain the computing power circuit for corresponding calculations. When designing corresponding algorithms, memory merging is one of the most critical factors. Due to the use of corresponding generated $X_{l-l}$, $X_{k-l}$, and $X_{k-i}$ during

the solving process, continuous reading of memory ensures the effectiveness of merging access, making the algorithm efficient.

4.2. **Resource Preemption Scheduling Method Based on Container Resource Cluster Deployment.** To improve the overall service quality of all work in the cluster, there is a trade-off between ensuring the real-time timeout rate and reducing the completion time of batch work. If the batch container is preempted too many times, it is likely to be eliminated by the operating system because of too few resources, which will lead to the batch work must be re-executed again, extending the execution time of the batch work, and also equivalent to wasting the resources allocated by the previous batch work. On the other hand, if a real-time job waits for a long time due to insufficient resources, it increases the scheduling delay. Since the completion time of batch work occupying more resources is more relaxed, as long as the container is not removed after preemption, the resource preemption of batch work containers in operation is safe and efficient. So this study defines the meaning of cost for the container a certain amount of resources, and the remaining resources are still enough to run the possibility of batch processing task measure, if a container is likely to be removed by the operating system after insufficient resources, the higher the subsequent cost, otherwise the cost of the lower.

From the perspective of resource scheduling, the container model can be regarded as a two-dimensional vector $container_j$ composed of CPU and memory. In order to quantify the container selection, parameters are introduced in the container model to quantitatively decide which containers to select for preemption, by extending the container model as shown in Equation (7). By introducing the PreemptionPriority parameter to the container, it can be quantified to select multiple batch working containers for preemption. Initially, the PreemptionPriority parameter can be represented as shown in Equation (8).

$$container_j = (R_{jcpu} R_{jmem} PreemptionPriority_j)^T \tag{7}$$

$$PreemptionPriority = \frac{\alpha * R_{total}}{R_{current}} \tag{8}$$

Among them, pp represents PreemptionPriority, and $R_{current}$ is the number of resources currently occupied by the container, and $\alpha$ is a weight hyperparameter for a positive value. When the cluster resources are insufficient and must be preemption, $n$ PreemptionPriority minimum containers will be selected into the set $C_p$ of the preempted candidate containers, and they are subsets of the container collection C for all batch work for running tasks, and $n$ is the number of elements of the set $C_p$, and $n = crad(C_p)$. The parameter PreemptionPriority should not be unchanged. If the container with the minimum PreemptionPriority is preempted by the execution, then its execution speed will be seriously affected, thus dragging the whole batch work down. In addition, the container is seized the more, the greater the possibility of being eliminated, the batch work will have to perform, and after the container PreemptionPriority parameter is still the smallest, this will lead to the batch work eventually eliminated again, so it is necessary to dynamically update the container PreemptionPriority parameters. When dynamically updating the PreemptionPriority parameter of a container, first consider the number of times it has been preempted. Therefore, if a container is preempted too often, it is likely to be eliminated by the operating system. In addition, when the batch work approaches or exceeds its deadline, it is best not to choose its task container for preemption to ensure the service quality of the batch work. When preemption is required, for the running batch work, the container is shown in Equation (9) for the PreemptionPriority parameter to decide whether to select it into the preempcontainer set.

$$pp = nbp + \frac{\alpha * R_{total}}{R_{current}} + F_{te} + \frac{\beta * ddl - t_{arrival}}{|ddl - t_{current}|} \tag{9}$$

Where $nbp$ is the quantity of times the container has been preempted, and $R_{total}$ is the total amount of resources in the cluster, and $F_{te}$ is a binary sign which is used to mark whether the batch processing work which belongs to containers has been beyond its expiration date, and 0 without timeout, and 1 if otherwise. $t_{current}$ and $t_{arrival}$ represent the soft deadline of the batch work, the current time, and the arrival time of the batch work respectively, and $\beta$ is the weight of the item which is considered by time factors, and it is a hyperparameter with positive values.

After the candidate containers perform resource recovery, it is necessary to determine how many resources should be seized from each container. When the real-time working resource resource requirement $R = \langle R_{mem} R_{cpu} \rangle$ is given, the seized container set is shown in Equation (10). Let the current resource of the jth container be $R_j = \langle R_{jmem} R_{jcpu} \rangle$, and this study has designed algorithms to determine the amount of resources recovered from $R_{pj} = \langle R_{pjmem} R_{pjcpu} \rangle$, including equivalence method, proportional method, and step by step method.

$$C_p = \{container_0 container_1 container_2 \dots container_{n-1}\} \tag{10}$$

In the equivalence mechanism, each container in the set of preempted containers will be preempted and the same amount of resources will be reclaimed from each container, as shown in Algorithm 2. Each container in the set will be preempted by $1/n$ of $R_{mem}$. Due to the fact that the number of virtual cores in a container cannot be a decimal, $R_{jcpu} \in N^+$ only applies this method to calculate the amount of memory reclaimed from each preempted container. As for the real-time working vCPU demand, the number of CPU cores reclaimed from each container is calculated by rounding up the average, i.e. $ceil\left(\frac{R_{cpu}}{n}\right)$.

**Algorithm 2** Optimization algorithm for computing power resource scheduling based on container cluster deployment

**Input:** parameter $\mathbb{C}$, $n$, $R_{total}$, $R_{current}$, $t_{arrival}$, $t_{current}$
**Output:** $R_j$
**for** $c_i \in C$ **do**
  calculate the parameters $pp_i$
  select $n$ containers with the smallest $pp$ from $C$ to form the preempted container set $C_p$
  **for** $c_j \in C_p$ **do**
    calculate the amount of recycled memory $R_{pjmem} = R_{mem}/n$
    calculate the number of recovered CPU $R_{pjcpu} = ceil * R_{jcpu}/n * pf$
    the amount of recovered resources is $R_{pj} = \langle R_{pjmem}, R_{pjcpu} \rangle$
    Perform pending based resource preemption on container $c_j$
    let $R_j \leftarrow R_j - R_{pj}$
  **end for**
**end for**
Return

After determining the number of recycled resources for each container, container pending preemption will be executed. As all processes during resource preemption are calculated through quantitative indicators, this preemption is more fine-grained and precise, which can meet real-time work resource requirements while reducing the possibility of preempted containers being eliminated, thereby reducing the impact on batch processing work and shortening the completion time of batch processing work. In addition, although it is not mandatory, the multiple containers selected for recycling may be distributed on different nodes. The recycling steps on each node can be performed in parallel, which can speed up context saving to a certain extent, reduce the scheduling delay of real-time work, and allow more sufficient execution time to return successfully before the deadline. At the same time, due to the greater possibility of batch processing work being preempted continuing to execute on the original schedule after the completion of preemption, the damage to cluster resource utilization is less, and the work throughput capacity of the cluster is also better improved. This study sets the proposed method as the Optimization Algorithm of Computing Resource Scheduling based on Container Cluster Deployment (CRS-CCD) [48].

## 5. Empirical Examples.

5.1. **Experimental Design.** In this study, the computing power data of Chinese listed companies in 2010, 2015 and 2020 were used to form the basic structure of the computing power resource network model for experiments (the specific network structure is shown in Table 1). In this study, the effectiveness of the proposed CRS-CCD algorithm was analyzed through experiments at different scales. Data ranges of determined parameters and fuzzy parameters for small and large instances reference [15]. This study is based on MATLAB software and Python framework on two Linux operating servers (Intel Xeon processor (34 GHZ) 64GB memory), each with 6 core CPU and two NVIDIA Titan X GPU and 100 GB RAM. Since the results of the public opinion control model in the experiment may be different in each run, the evaluation results are set as the average value after 200 iterative runs. Our research team implemented the proposed algorithm and the comparison algorithm on Tensorflow1.5.1. The experimental process adopts a grouping approach, dividing the data into 10 equal groups and using cross validation, that is, dividing the dataset into 10 equal parts. Each time, one set of data is selected as the test set, and the other groups are used as the training set. Finally, the average value is taken, and three different scenarios are formed: 2-fold, 4-fold, and 10-fold. And save all data in CSV format in the MySQL database for data processing.

This study proposes a computational resource scheduling optimization algorithm (CRS-CCD) based on container cluster deployment. Degree Centrality (DC), Intermediate Centrality (IC), Proximity Centrality

TABLE 1. Characteristics of the Computing Power Resource Network Data Set

| Network Serial Number | Data Set | Type | Number of Nodes | Number of Node Boundaries | Average Degree | Node Average Path | Cluster Coefficient |
|---|---|---|---|---|---|---|---|
| 1 | 2010 Computational Power Resource Network Data Set | Directed | 2245 | 8290 | 3.24 | 2.49 | 0.138 |
| 2 | 2015 Computational Power Resource Network Data Set | Directed | 2415 | 10393 | 3.59 | 2.70 | 0.173 |
| 3 | 2020 Computational Power Resource Network Data Set | Directed | 2849 | 12849 | 3.63 | 2.91 | 0.190 |

(CNC), Ant Colony Optimization (ACO), Swarm Optimization (SWO) Compare benchmark algorithms such as K-Shell Centrality (KSC) and Weighted K-Shell Degree Neighborhood (WKS-DN) [49]. In combination with reference [36, 50], two precision functions are used: Mean absolute error (MAE) and Root-mean-square deviation (RMSE). The specific calculation methods are shown in Equation (11) and Equation (12) respectively.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|f_i - y_i| \tag{11}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(observed_t - predicted_i)^2} \tag{12}$$

However, the optimization of computing resources is usually regarded as a binary classification task, and in the evaluation Confusion matrix of binary classification tasks with two categories [11]. True Positive (TP), indicating the correct number of predicted links; True Negative (TN), representing the number of correct unforecasted links; False Positive (FP), indicating the number of incorrectly predicted links; False Negative (FN), indicating the number of unpredictable links that are incorrect. Based on this, the following indicators can be obtained: true case rate/recall rate/sensitivity, etc. The calculation of True Positive Rate (TPR), False Positive Rate (FPR), True Negative Rate (TNR), and Precision can be referred to in reference [32]. The evaluation is based on the following two indicators, namely, Area Under the Receiver Operating Characteristics curve (AUROC) [22] and Average Precision (AP) [23]. The ROC curve is the curve between the true case rate (sensitivity) on the Y axis and the false positive case rate (1-specificity) on the $X$ axis, and the ROC curve is the area under the area $[0, 1]$, which can be calculated using the trapezoidal rule, which adds all trapezoids under the curve. The AUROC value of the link prediction method should be greater than 0.5, and the higher the AUROC value, the better the performance of the link prediction method. The mean accuracy is a single-point summary value calculated based on different recall thresholds, which is the average accuracy value of the interval $[0, 1]$ recall value. This is as shown in Equation (13). Where p represents the exact rate at different recall r thresholds, and R is the set of different thresholds.

$$AP = \sum_{k=1}^{R} p(k) * \Delta r(k) \tag{13}$$

5.2. **Experimental Results.** Table 2 reports the area values under the curve of the CRS-CCD container cluster deployment based computing power resource scheduling optimization algorithm and other benchmark methods proposed in this article in the computing power network dataset. This study found that the proposed CRS-CCD algorithm for computing resource scheduling optimization based on container cluster deployment has better experimental results in real computing network datasets. Table 3 reports the average accuracy results of the CRS-CCD container cluster deployment based computing power resource scheduling optimization algorithm and other benchmark algorithms proposed in this article in the computing power network dataset. The results show that the proposed CRS-CCD algorithm

for computing resource scheduling optimization based on container cluster deployment has a high average accuracy value in all computing network datasets.

Table 2. Area Values under Curves of Various Computational Resource Network Datasets in Different Methods

| Cross Validation Level | Data Set Name | Supply Chain Network Optimization Framework | | | |
|---|---|---|---|---|---|
| | | DC | IC | CNC | ACO |
| 2-fold | 2010 Computational Power Resource Network Dataset | 0.4282 | 0.3532 | 0.3893 | 0.5355 |
| | 2015 Computational Power Resource Network Dataset | 0.4461 | 0.2411 | 0.4048 | 0.4985 |
| | 2020 Computational Power Resource Network Dataset | 0.3019 | 0.3199 | 0.3002 | 0.4810 |
| 4-fold | 2010 Computational Power Resource Network Dataset | 0.4318 | 0.3849 | 0.4012 | 0.5838 |
| | 2015 Computational Power Resource Network Dataset | 0.4579 | 0.3191 | 0.4127 | 0.5492 |
| | 2020 Computational Power Resource Network Dataset | 0.3628 | 0.4292 | 0.5228 | 0.5051 |
| 10-fold | 2010 Computational Power Resource Network Dataset | 0.4583 | 0.4055 | 0.4828 | 0.6638 |
| | 2015 Computational Power Resource Network Dataset | 0.4910 | 0.3525 | 0.5485 | 0.5729 |
| | 2020 Computational Power Resource Network Dataset | 0.4739 | 0.4660 | 0.5829 | 0.5391 |
| Cross Validation Level | Data Set Name | Supply Chain Network Optimization Framework | | | |
| | | SWO | KSC | WKS-DN | CRS-CCD |
| 2-fold | 2010 Computational Power Resource Network Dataset | 0.6010 | 0.5939 | 0.6584 | 0.8472 |
| | 2015 Computational Power Resource Network Dataset | 0.6239 | 0.5328 | 0.6749 | 0.8329 |
| | 2020 Computational Power Resource Network Dataset | 0.5029 | 0.5930 | 0.7381 | 0.8493 |
| 4-fold | 2010 Computational Power Resource Network Dataset | 0.6122 | 0.6638 | 0.6692 | **0.8938** |
| | 2015 Computational Power Resource Network Dataset | 0.6283 | 0.6573 | 0.6839 | **0.8492** |
| | 2020 Computational Power Resource Network Dataset | 0.6650 | 0.6019 | 0.7680 | **0.8204** |
| 10-fold | 2010 Computational Power Resource Network Dataset | 0.5383 | 0.6782 | 0.6949 | **0.8930** |
| | 2015 Computational Power Resource Network Dataset | 0.5839 | 0.6839 | 0.7382 | **0.8575** |
| | 2020 Computational Power Resource Network Dataset | 0.5291 | 0.6371 | 0.7029 | **0.8957** |

Note: The values displayed in bold indicate that the corresponding algorithm performs well.

In combination with reference [20], two precision functions are used: Mean absolute error (MAE) and Root-mean-square deviation (RMSE). Table 4 reports the MAE and RMSE values of the proposed CRS-CCD container cluster deployment based computing resource scheduling optimization algorithm and other benchmark algorithms in different computing resource network topologies. The higher the MAE and

TABLE 3. Average Accuracy Values of Various Computing Power Resource Network Datasets in Different Methods

| Cross Validation Level | Data Set Name | Supply Chain Network Optimization Framework | | | |
|---|---|---|---|---|---|
| | | DC | IC | CNC | ACO |
| 2-fold | 2010 Computational Power Resource Network Dataset | 0.2859 | 0.2729 | 0.3382 | 0.3291 |
| | 2015 Computational Power Resource Network Dataset | 0.2039 | 0.2049 | 0.4818 | 0.4293 |
| | 2020 Computational Power Resource Network Dataset | 0.2371 | 0.2385 | 0.3919 | 0.3527 |
| 4-fold | 2010 Computational Power Resource Network Dataset | 0.3271 | 0.3291 | 0.3418 | 0.3472 |
| | 2015 Computational Power Resource Network Dataset | 0.3391 | 0.2410 | 0.3204 | 0.4839 |
| | 2020 Computational Power Resource Network Dataset | 0.3204 | 0.2394 | 0.4491 | 0.4311 |
| 10-fold | 2010 Computational Power Resource Network Dataset | 0.2395 | 0.3417 | 0.4372 | 0.4638 |
| | 2015 Computational Power Resource Network Dataset | 0.3581 | 0.3921 | 0.3340 | 0.4953 |
| | 2020 Computational Power Resource Network Dataset | 0.3849 | 0.3100 | 0.4839 | 0.4552 |
| Cross Validation Level | Data Set Name | Supply Chain Network Optimization Framework | | | |
| | | SWO | KSC | WKS-DN | CRS-CCD |
| 2-fold | 2010 Computational Power Resource Network Dataset | 0.5395 | 0.5873 | **0.6497** | 0.6240 |
| | 2015 Computational Power Resource Network Dataset | 0.5856 | 0.5443 | 0.7354 | **0.8063** |
| | 2020 Computational Power Resource Network Dataset | 0.5748 | 0.6054 | 0.7703 | **0.8352** |
| 4-fold | 2010 Computational Power Resource Network Dataset | 0.5663 | 0.6281 | **0.6617** | 0.6512 |
| | 2015 Computational Power Resource Network Dataset | 0.6438 | 0.5985 | 0.6794 | **0.7445** |
| | 2020 Computational Power Resource Network Dataset | 0.6192 | 0.5474 | 0.7592 | **0.8186** |
| 10-fold | 2010 Computational Power Resource Network Dataset | 0.6479 | 0.5847 | 0.6873 | **0.7299** |
| | 2015 Computational Power Resource Network Dataset | 0.6753 | 0.6012 | 0.6944 | **0.7742** |
| | 2020 Computational Power Resource Network Dataset | 0.6191 | 0.6126 | 0.8205 | **0.8742** |

Note: The values displayed in bold indicate that the corresponding algorithm performs well.

RMSE values, the lower the accuracy of the prediction optimization algorithm. According to Figure 1, it can be seen that the proposed CRS-CCD based computing resource scheduling optimization algorithm for container cluster deployment is generally superior to other methods. This is because the proposed S-CCD based computing resource scheduling optimization algorithm for container cluster deployment has the ability to quickly respond and adjust and optimize the computing resource network in real-time, while also minimizing the overall loss of the computing resource network.

TABLE 4. Comparison Results of Accuracy Functions for Optimization Algorithms of Computing Power Resource Networks

| | Index | DC | IC | CNC | ACO |
|---|---|---|---|---|---|
| Actual Value | MAE | 0.6383 | 0.6738 | 0.6239 | 0.5371 |
| | RMSE | 0.7362 | 0.6988 | 0.6582 | 0.5481 |
| Optimal Value | MAE | 0.5353 | 0.6074 | 0.6248 | 0.5743 |
| | RMSE | 0.6353 | 0.6428 | 0.6739 | 0.5938 |
| | Index | SWO | KSC | WKS-DN | CRS-CCD |
| Actual Value | MAE | 0.5382 | 0.5644 | 0.4272 | **0.2371** |
| | RMSE | 0.5463 | 0.5738 | 0.4371 | **0.2492** |
| Optimal Value | MAE | 0.4739 | 0.5192 | 0.4472 | **0.1281** |
| | RMSE | 0.5291 | 0.5332 | 0.4738 | **0.2455** |

Note: The bold part indicates that this algorithm method is relatively optimal under this parameter condition.

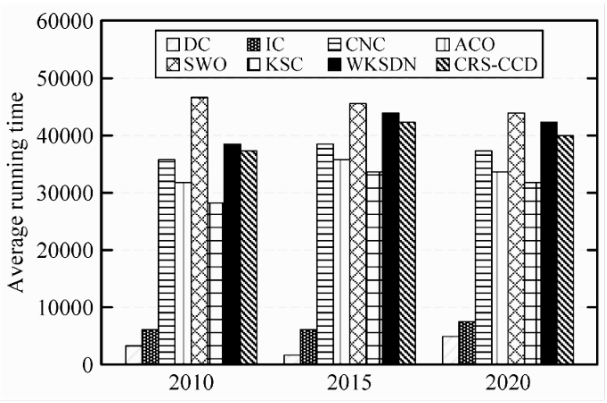


FIGURE 1. Comparison of Supply Chain Network Optimization Algorithms Based on Time Complexity

6. **Conclusion.** In order to adapt to the rapid development of big data and intelligent devices faster, it is the goal of current scholars to study an algorithm that can meet current needs and have efficient and accurate computing power. This study proposes an Optimization Algorithm of Computing Resource Scheduling based on Container Cluster Deployment (CRS-CCD). Firstly, optimize the deployment of container clusters based on cloud data centers, and implement cost control for the construction of cloud data centers and communication networks. In addition, the parallel computing power optimization framework based on improved kernel functions optimizes the steady-state safe operation of classical computing power systems. CUDA function and GPU framework are used to improve the Algorithmic efficiency, and container resource cluster is used to optimize the deployment of computing resources. To validate the proposed algorithm, in a real computing resource network environment, compare the proposed computing resource scheduling optimization algorithm based on container cluster deployment with other benchmark algorithms. It can be seen that the proposed algorithm for scheduling and optimizing computing resources based on container cluster deployment is more efficient and accurate than other benchmark algorithms for optimizing computing resources. The intelligent industry needs a large amount of computing resources when dealing with large-scale data and complex tasks. The optimization algorithm proposed in this paper can effectively improve computing efficiency, reduce costs and energy consumption, and also improve the user experience for the majority of users. It has played an active role in promoting intelligent technology innovation.

**Acknowledgment.** This study is sponsored by Hunan First Normal University.

## REFERENCES


[1] P. Lv, Q.-R. Liu, H.-C. Chen, T. Chen, "Domain-Oriented Software Defined Computing Architecture," *China Communications*, vol. 16, no. 6, pp. 162–167, 2019.

[2] S. Viktória, H. Lukáš, H. Jakub, "Nextflow in Bioinformatics: Executors Performance Comparison Using Genomics Data," *Future Generation Computer Systems*, vol. 142, pp. 328–339, 2023.

[3] D.-S. Dinamarca, J. Francisco, P. Hector, O.-J. Sabattin, R.-F. Rojas, D.-H. Reyes, Rogget Marcelo Reyes, Coronado-Hernández Jairo, Romero-Conrado Alfonso R, "Technical Evaluation of Thermal Containers for the Distribution of Thermolabile Medicines in the Chilean Pharmaceutical Supply Chain," *Procedia Computer Science*, vol. 220, pp. 922–927, 2023.

[4] G. Luciano, P. Fernando, G. Davide, N. Camilla, G. Flavia, D. Simona, B. Emilio, T. Giampaolo, C. Tonia, M. Maurizio, D.-P. Elisa, O.-M.Elisabetta, D.-B. Maria, N. Nicola, S. Giovanni, M. Angelo, "A Computational Framework for Comprehensive Genomic Profiling in Solid Cancers: The Analytical Performance of a High-Throughput Assay for Small and Copy Number Variants," *Cancers*, vol. 14, no. 24, 2022.

[5] M.-R. Li, S.-Q. Zhou, L.-K. Cheng, F.-N. Mo, L. Chen, S.-Z. Yu, J. Wei, "3D Printed Supercapacitor: Techniques, Materials, Designs, and Applications," *Advanced Functional Materials*, vol. 33, no. 1, 2022.

[6] T.- A. Khan, K. Karim, I. Muhammad, Y. Zhang, J.-Y. Long, M. Asif, M. Nasir, Z.-J. Xie, C. Li, H. Zhang, "Recent advance in two-dimensional MXenes: New horizons in flexible batteries and supercapacitors technologies," *Energy Storage Materials*, vol. 53, pp. 783–826, 2022.

[7] X. Hu, H.-X. Wang, W.-Z. Meng, K.-H. Yeh, "Attribute-based Data Sharing Scheme with Flexible Search Functionality for Cloud Assisted Autonomous Transportation System," *IEEE Transactions on Industrial Informatics*, 2023. Available: https://doi.org/10.1109/TII.2023.3242815.

[8] L.-L. Wang, Y. Lin, T. Yao, X. Hu, K.-T. Liang, "FABRIC: Fast and Secure Unbounded Cross-System Encrypted Data Sharing in Cloud Computing, " *IEEE Transactions on Dependable and Secure Computing*, 2023. Available: https://doi.org/10.1109/TDSC.2023.3240820.

[9] S. Kasun, G. Kosala, P. Nisitha, K. Nihal, M. Mehdi, "Modern Supercapacitors Technologies and Their Applicability in Mature Electrical Engineering Applications," *Energies*, vol. 15, no. 20, 2022.

[10] M. Przemyslaw, P. Michał, P. Damian, "Numerical and Experimental Investigations of the Influence of Operation on the Technical Condition of Pressure Vessels," *Materials*, vol. 15, no. 20, 2022.

[11] M. Alberto, P. Antonio, K. Amit, D.-L.-C. Luis, R. Diego, M.-J. Ignacio, "Integration of Machine Learning-Based Attack Detectors into Defensive Exercises of a 5G Cyber Range," *Applied Sciences*, vol. 12, no. 20, 2022.

[12] X.-C. Xu, Y.-X. Jiang, H. Wen, W.-J. Hou, S.-L. Chen, "A secure edge power system based on a Docker container," *Frontiers in Energy Research*, no. 1, 2022.

[13] M. Anand K, P. Emmanuel S, G. Mahesh C, "CONTAIN4n6: a systematic evaluation of container artifacts," *Journal of Cloud Computing*, vol. 11, no. 1, pp. 1–14, 2022.

[14] A. Tahir, A. Sikandar, K.-H. Ullah, S. Ali, A. Khalid, S.-M. Asif, "Container Performance and Vulnerability Management for Container Security Using Docker Engine," *Security and Communication Networks*, 2022, 2022.

[15] D. Fabio, S. Corrado, S.-F. Fausto,"Wale: A solution to share libraries in Docker containers," *FUTURE GENERATION COMPUTER SYSTEMS-THE INTERNATIONAL JOURNAL OF ESCIENCE*, vol. 100, pp. 513–522, 2019.

[16] T.-Y. Wu, X.-l. Guo, Y.-C. Chen, S. Kumari, C.-M. Chen, "Amassing the Security: An Enhanced Authentication Protocol for Drone Communications over 5G Networks, " *Drones*, vol. 6, no. 1, 10, 2022.

[17] D. Mo, H.-Y. Yu, Z.-P. Sun, M.-Z. Wu, L.Zhang, Y.-Y. Sui, G.-H. Yu, T. Han, R.-H. Zhao,"Design of IoT Gateway for Crop Growth Environmental Monitoring Based on Edge-Computing Technology, " *Computational Intelligence and Neuroscience*, 2022, 2022.

[18] A.-M. Daniils, I. Aleksandrs, G. Elans, R. Dmitrijs,"Evaluation of a Long-Distance," *IEEE 802.11ah Wireless Technology in Linux Using Docker Containers. ELEKTRONIKA IR ELEKTROTECHNIKA*, vol. 28, no. 3, pp. 71–77, 2022.

[19] K. Woojae, J. Inbum,"Simulator for Interactive and Effective Organization of Things in Edge Cluster Computing, " *SENSORS*, vol. 21, no. 8, 2021.

[20] J.-H. Gu, J.-H. Feng, H.-Y. Xu, T. Zhou,"Research on Terminal-Side Computing Force Network Based on Massive Terminals, " *Electronics*, vol. 11, no. 13, 2022.

[21] T. Ashleigh, G. Rupert,"A Comparative Review of Lead-Acid, Lithium-Ion and Ultra-Capacitor Technologies and Their Degradation Mechanisms, " *Energies*, vol. 15, no. 13, 2022.

[22] O.-V. Ladyzhenskaya, T.-S. Aniskina, V.-A. Kryuchkova,"Elements of container technology for growing blackberry varieties Ouachita," *IOP Conference Series: Earth and Environmental Science*, vol. 1045, no. 1, 2022.

[23] K. Danial, B. Hamidreza, V.-M. Joeri, B. Maitane,"A Comprehensive Review of Lithium-Ion Capacitor Technology: Theory, Development, Modeling, Thermal Management Systems, and Applications, "*Molecules*, vol. 27, no. 10, 2022.

[24] W. Richard, W. Stephen K., B. Chris J, "Containerless Techniques for in-situ X-Ray Measurements on Materials in Extreme Conditions, " *Journal of the Physical Society of Japan*, vol. 91, no. 9, 2022.

[25] D.-X. Hou, K.-L. Zhao, W.-F. Li, S.-D.Du, "A Realistic, Flexible and Extendible Network Emulation Platform for Space Networks, " *Electronics*, vol. 11, no. 8, 2022.

[26] G. Sourav, Y Sarita, D. Ambika, T. Tiju,"Techno-economic understanding of Indian energy-storage market: A perspective on green materials-based supercapacitor technologies, " *Renewable and Sustainable Energy Reviews*, vol. 161, 2022.

[27] T.-Y. Wu, F.-F. Kong, Q. Meng, S. Kumari, C.-M. Chen, "Rotating Behind Security: An enhanced authentication protocol for IoT-enabled devices in distributed cloud computing architecture," *EURASIP Journal on Wireless Communications and Networking*, vol. 2023, 36, 2023.

[28] M.-C.-J.-T.-W. Kithulwatta, P.-N.-K.Jayasena, T.-G.-S.-B. Kumara, M.-K.-T.-R. Rathnayaka,"Integration With Docker Container Technologies for Distributed and Microservices Applications: A State-of-the-Art Review, " *International Journal of Systems and Service-Oriented Engineering (IJSSOE)*, vol. 12, no. 1, 2022.

[29] Y.-S. Wang, L.-J. Feng, J.-F. Wang, H.-D. Zhao, P. Liu,"Technology Trend Forecasting and Technology Opportunity Discovery Based on Text Mining: The Case of Refrigerated Container Technology, " *Processes*, vol. 10, no. 3, 2022.

[30] M.-A. Bandurin, I.-P. Bandurina, A.-A. Mykhailin,"Application of geotextile containers for removal of silt layers of the Krasnodar reservoir, " *IOP Conference Series: Earth and Environmental Science*, vol. 996, no. 1, 2022.

[31] Q.-I. He, F. Zhang, G.-Q. Bian, W.-Q. Zhang, Z. Li, D.-L. Duan, "Real-time network virtualization based on SDN and Docker container, " *Cluster Computing*, vol. 26, no. 3, pp. 2069–2083, 2022.

[32] C.-R. Chiang, "Contention-aware container placement strategy for docker swarm with machine learning based clustering algorithms," *CLUSTER COMPUTING-THE JOURNAL OF NETWORKS SOFTWARE TOOLS AND APPLICATIONS*, vol. 26, no. 1, pp. 13–23, 2023.

[33] Y.-L. Wang, Q.-X. Wang, X.-S. Chen, D.-J. Chen, X.-J. Fang, M.-Y. Yin, N. Zhang, "ContainerGuard: A Real-Time Attack Detection System in Container-Based Big Data Platform," *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS*, vol. 18, no. 5, pp. 3327–3336, 2022.

[34] N. Penchalaiah, A.-S. Al-Humaimeedy, M. Maashi, J.-C. Babu, 4, O.-I. Khalaf, T.-H.-H. Aldhyani, "Clustered Single-Board Devices with Docker Container Big Stream Processing Architecture, " *Computers, Materials & Continua*, vol. 73, no. 3, 2022.

[35] M. Gabrielle, "Settler colonialism's container technologies: photographing crates in the Canadian Arctic (1926–1953)," *Settler Colonial Studies*, vol. 11, no. 4, 2021.

[36] T. Hao, X.-L. Xu, T.-Y. Lin, Y.-Q. Cheng, R. Cheng, R. Lei, B. Muhammad, "DIMA: Distributed cooperative microservice caching for internet of things in edge computing by deep reinforcement learning," *World Wide Web*, 2021 (prepublish).

[37] Z.-C. Hua, Y. Yu, J.-Y. Gu, Y.-B. Xia, H.-B. Chen, B.-Y. Zang, "TZ-Container: protecting container from untrusted OS with ARM TrustZone," *Science China Information Sciences*, vol. 64, no. 9, pp. 28–43, 2021.

[38] A. Davide, D. Claudio, E. Pasolli, F. Asnicar, S. Manara, D.-H. Parks, C. Rinke, M. Chuvochina, S. Nurk, D. Meleshko, A. Korobeynikov, D.-D. Kang, F. Li, E. Kirton, J.-N. Nissen, J. Johansen, R.-L. Allesøe, L.-X. Chen, K. Anantharaman, A. Shaiber, A. Shaiber, A.-M. Eren, R.-M. Bowers, N.-C. Kyrpides, R. Stepanauskas, E. Afgan, D. Baker, B. Batut, P.-D. Tommaso, M. Chatzou, E.-W.Floden, F. Strozzi, R. Janssen, R. Wurmus, P.-A. Ewels, A. Peltzer, S. Fillinger, D.-H. Parks, M. Imelfort, C.-T. Skennerton, Orakov A,A. Fullam, L.-P. Coelho, P.-P Chan, T.-M.Lowe, J. Goris, K.-T. Konstantinidis, J.-A. Klappenbach, J.-T. Evans, V.-J. Denef, M.-R. Olm, C.-T. Brown, B. Brooks, D. Li, C.-M. Liu, R. Luo, D. Albanese, C. Donati, B. Denis, S.-B. Schmidt Thomas, F. Anthony, O. Askarbek, "Large-scale quality assessment of prokaryotic genomes with metashot/prok-quality," *F1000Research*, vol. 10, 2021.

[39] S. Nawat, A. Chaodit, "Optimal Cloud Orchestration Model of Containerized Task Scheduling Strategy Using Integer Linear Programming: Case Studies of IoTcloudServe@TEIN Project," *Energies*, vol. 14, no. 15, 2021.

[40] W.-T. Cao, Y. Guo, H. Zhang, M.-H. Sun, M. Yuan, "Application of Container Technology in Numerical Ocean Model: a Kind of High-performance ROMS Containerized Architecture," *Journal of Physics: Conference Series*, vol. 1961, no. 1, 2021.

[41] K.-J. Christian, W. Lars, E. David, "BIGwas: Single-command quality control and association testing for multi-cohort and biobank-scale GWAS/PheWAS data," *GigaScience*, vol. 10, no. 6, 2021.

[42] H. Philipp, H. Tobias, M. Jürgen, "OPAL—The Toolbox for the Integration and Analysis of IoT in a Semantically Annotated Way," *Sensors*, vol. 21, no. 12, 2021.

[43] H.-F. Liu, S.-G. Chen, Y.-C Bao, et al."A High Performance, Scalable DNS Service for Very Large Scale Container Cloud Platforms," *19th ACM/IFIP/USENIX Middleware Conference (Industrial Track)*, 2018.

[44] G. Kamate, S.G.-R. Prasad,"Docker & Containers, the Future of Microservices," *Journal of Global Economy, Business and Finance*, vol. 3, no. 4, 2021.

[45] A. Bhardwaj, C.-R. Krishna,"Virtualization in Cloud Computing: Moving from Hypervisor to Containerization—A Survey," *Arabian Journal for Science and Engineering*, 2021 (prepublish).

[46] G. Tom, D.-T. Filip, V. Bruno, "Extending Kubernetes Clusters to Low-Resource Edge Devices Using Virtual Kubelets," *IEEE TRANSACTIONS ON CLOUD COMPUTING*, vol. 10, no. 4, pp. 2623–2636, 2022.

[47] T.-Y. Wu, L.-y. Wang, X.-l. Guo, Y.-C. Chen, S.-C. Chu, "SAKAP: SGX-Based Authentication Key Agreement Protocol in IoT-Enabled Cloud Computing," *Sustainability*, vol. 14, no. 17, 11054, 2022.

[48] C.-M. Chen, S. Lv, J. Ning, J.-M.-T. Wu, "A Genetic Algorithm for the "Waitable Time-Varying Multi-Depot Green Vehicle Routing Problem"," *Symmetry*, vol. 15, no. 1, 124, 2023.

[49] A.-L.-H.-P. Shaik, M.-K. Manoharan, A.-K. Pani, R.-R. Avala, C.-M. Chen, "Gaussian Mutation–Spider Monkey Optimization (GM-SMO) Model for Remote Sensing Scene Classification," *Remote Sensing*, vol. 14, no. 24, 6279, 2022.

[50] X. Hu, Q. Zheng, H. Xin, K.-H. Yeh, "Revocable and Unbounded Attribute-based Encryption Scheme with Adaptive Security for Integrating Digital Twins in Internet of Things, " *IEEE Journal on Selected Areas in Communications*, 2023. Availale: https://doi.org/10.1109/JSAC.2023.3310076.