Journal of Computers Vol. 34 No. 6, December 2023, pp. 133-143
doi: 10.53106/199115992023123406010

Mitigating Cloud Computing Virtualization Performance Problems with an Upgraded Logical Convergence Strategy

Ming Zhao*, Zhen Wang, Yalong Li, Xiumei Qin

School of Computer Science, Yangtze University, Jingzhou, China
hitmzhao@gmail.com, 1164050018wangzhen@gmail.com

Received 23 May 2023; Revised 23 September 2023; Accepted 23 October 2023

Abstract. In the domain of cloud computing and network resource virtualization, existing fusion techniques for containers and virtual machines suffer from high energy consumption, inflexible scheduling requirements, and suboptimal resource utilization. This study critically examined the current methods, accounted for the contemporary requirements, and developed a novel strategy aimed at maximizing resource utilization while minimizing energy consumption. Comprehensive experiments illustrate the superiority of our approach over state-of-the-art fusion strategies such as Kubernetes+Kubevirt and OpenStack+Kubernetes, demonstrating significant reductions in energy consumption, improved resource utilization, and enhanced system performance.

Keywords: cloud computing, container, logical convergence, deep learning, network resource virtualization, virtual machine

1 Introduction

Cloud computing is a computer model that is built on the Internet. By storing data and processing power in a data center (cloud), it is possible to provide customers with a wide range of computing resources and services, allowing them to use and share resources on demand [1]. Traditional computing approaches typically need users to have local particular computer equipment and software to deal with and store data. However, in the cloud computing mode, users can connect to the cloud service provider's data center via the Internet and use the computing resources provided by the cloud service provider, such as virtual machines, storage space, databases, applications, and so on to complete various computing tasks. With the continuous development of cloud computing systems, relevant core technologies such as virtualization technology, distributed technology, microservice technology, and container orchestration technology have steadily progressed. In recent years, the emergence of new technology, such as the Docker container system [2], distributed storage systems based on erasure code technology [3], the Spark system based on memory computing [4], a copy of the based on distributed data storage technology, and distributed concurrent programming models based on graphs, have subverted the original virtualization technology. These new technologies not only increase cloud platform resource utilization and computing speed, but also provide organizations with more big data application models such as batch processing, real-time data processing, stream data processing, random data inquiry, and data mining.

Virtualization technology is the most important and fundamental technology in cloud computing [5]. There are now two types of virtualization technologies: hardware-based virtualization technology and operating system-based virtualization technology. Multiple virtual machines can be installed and run on the same physical machine using hardware-based virtualization, each with its own operating system and separated from other virtual machines on the same physical machine. By providing a virtualization layer, the primary premise is to separate the hardware from the operating system. The virtualization layer's primary job is to run numerous operating system instances on a single physical server at the same time. The virtualization layer enables operating system instances to share physical server resources via dynamic partitioning, so that each virtual machine has a set of independent simulation hardware equipment, including CPU, memory, storage, motherboard, graphics card, network card, and other hardware resources. These virtual hardware resources are then used to install their own operating systems, known as Guest operating systems, on their own computers. The user's program is ultimately executed in the Guest operating system. Infrastructure as a service (IaaS) is a crucial component of cloud services, and users can create virtual machines as needed. Hosted architectures and "bare metal" architectures are two common

* Corresponding Author

server virtualization designs. The virtualization layer is run as an application on top of the operating system in the hosted architecture [6].

However, in bare-metal designs, the virtualization layer is run directly on x86 hardware platforms, followed by the installation of the operating system and applications. VMware Server is representative of residential architecture virtualization, whereas Xen [7] and KVM [8] incorporate bare metal architecture virtualization.

Containers in an operating system-based model share a host operating system and any required library, driver, or binary files. Services can run inside containers at a fraction of the overhead introduced by virtual machines because there is no hardware abstraction layer. Their central tenet is to establish Control Groups on the Linux kernel to isolate the service runtime environment; this isolated runtime environment is referred to as a container. Containers can be thought of as instruments for packaging, delivering, and orchestrating software services and applications. Docker is a container platform that simplifies and standardizes application deployment in a variety of contexts. There are numerous ecosystem software applications related to distributed container management.

In the field of cloud computing virtualization, virtual machine technology and containers complement one another. They apply virtualization to different levels. The fundamental advantage of containers is their lower performance overhead, whereas VMS provides better isolation.

KVM hardware virtualization technology and Docker container technology have emerged as the current major virtualization technologies after years of development and competition. These two technologies' popular application frameworks are relatively mature, stable, and open source. OpenStack and Kubernetes are both open-source projects. OpenStack is a project that aims to create an open-source cloud management platform that is both stable and efficient. Computational, storage, network, and other resources are abstracted into computational resource pools, network resource pools, and storage resource pools using virtualization technology. Then, keystone, nova, neutron, glance, cinder, and other components are used to achieve unified scheduling and management of fundamental virtual resources such as virtual machines, bare metal, block storage, file storage, object storage, network, load balancing, security groups, and firewalls [9]. Keystone is used for authentication service in an OpenStack cluster, Nova for virtual machine deployment and calculation, Neutron for network service, Cinder for cloud hard disk storage, Glance for mirror service, Swift for object storage, Ceilometer for monitoring, Horizon for the visual interface, and Heat for application orchestration. The Google company's open-source Kubernetes is used to provide automatic deployment across the host cluster, extension, and application of the system management container. Kubernetes clusters can be easily run across a variety of container management applications, providing load balancing, resource monitoring, logging access and acquisition, authentication and authorization, health examination, horizontal extension and automatic discovery, and other functions [10]. Kubernetes clusters are divided into two types of nodes: Master and Minion. The Master serves as a control node, while the Minion serves as a compute node. The Master node carries the weight of Kubernetes' essential control components, such as kube-apiserver, kube-controller-manager, and kube-scheduler. These components can handle Kubernetes API queries as well as integrate controllers to handle routine cluster operations and background process management. Furthermore, the Master node is in charge of scheduling the pod, the most basic deployment unit in the Kubernetes cluster, as well as providing key-value storage services for storing Kubernetes cluster data information. Minion nodes are installed with kubelet and kube-proxy, which are important components required to maintain the pod runtime environment. kubelet serves as a proxy service for pods, monitoring them via apiserver or local configuration and performing tasks such as loading volumes, downloading keys, starting containers, periodic health checks, reporting pod health status, and making image backups. kube-proxy, on the other hand, implements the Kubernetes-abstracted service notion by keeping network rules and a connection forwarding mechanism on the host node. The underlying method relies on iptables to forward traffic. In general, in the Kubernetes cluster design, the Master and Minion nodes each conduct their own tasks while working together to produce an efficient containerized cluster management system.

In conclusion, container technology and virtual machine technology are critical in the field of cloud computing virtualization. However, their traditional implementations have some restrictions. Although virtual machine technology can provide isolation and diverse operating system support, it is slow to boot and requires a lot of resources. While container technology has the advantages of being lightweight and quick to start, there are issues in terms of resource separation and security. As a result, techniques to fully use the benefits of containers and virtual machines while overcoming their constraints have emerged as the primary focus of current research [11].

2 Related Work

Previous research has suggested two key VM and container convergence approaches [12]: physical and logical. Both approaches aim to increase resource use and overall performance. In the case of physical convergence, running the container as a process of the VMs, or the VMs in the container [13] can take full advantage of the container's lightweight characteristics, reduce resource consumption and start-up time, and take into account its isolation [14]. However, this shared environment increases system complexity and virtualization overhead because they run on the same physical computer. As a result, performance may be compromised.

Conversely, logical convergence can improve system availability while also reducing system complexity. Logical convergence optimizes resource utilization, accelerates start-up times, and improves overall stability and security by operating containers and VMs on different physical servers and exploiting the lightweight nature of containers. Additionally, running VMs and containers on different physical machines enhances accessibility and prevents the destruction of all virtualized processes in the event of a server failure. However, although logical convergence simplifies resource management, it may lead to lower resource utilization, increased energy consumption [15], and overly complex deployment and management processes.

3 Background

RedHat's Kubevirt technology utilizes container group pods to launch VMs, as depicted in Fig. 1. This approach is well-suited for physical convergence solutions [16] and is implemented through the use of the OpenStack Foundation's Kata Container technology [17]. Fig. 2 illustrates the presence of container group pods within VMs, which enables a careful balance between isolation and portability. This balance is particularly important in multi-tenant environments that require rapid deployments as well as resource sharing for development and testing purposes [18].

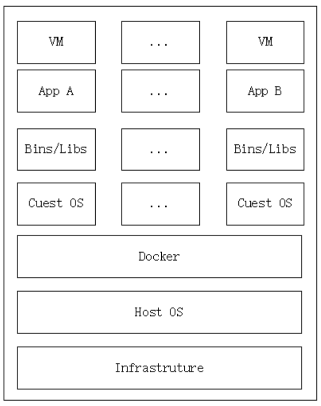

Fig. 1. Running virtual machines in a container

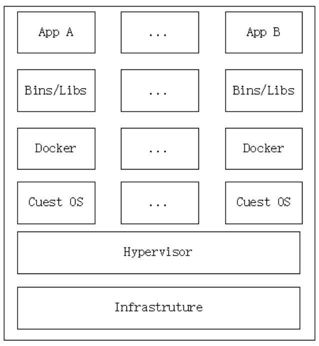

Fig. 2. Running a container in a virtual machine

In a recent study on logical convergence, it was found that China UnionPay implements an OpenStack cluster on a specific set of physical computers [19], whereas a separate Kubernetes cluster is deployed on another set, as illustrated in Fig. 3. This approach of using dedicated clusters for VMs and containers is particularly suitable for scenarios that demand high availability, such as large-scale application clusters and distributed systems. Moreover, adopting a logical convergence strategy can simplify system architectures and their management, resulting in improved operational efficiency and maintenance effectiveness.

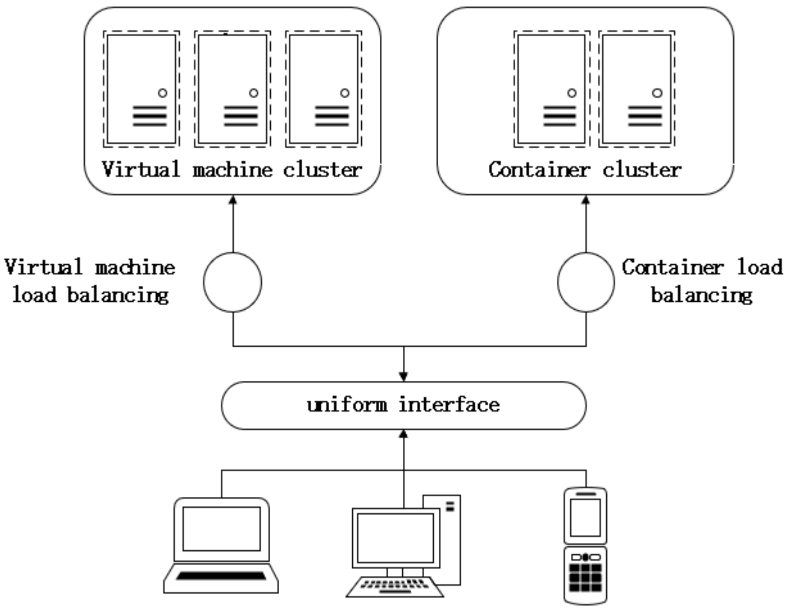

Fig. 3. Logical convergence of virtual machines and containers

This study introduces a resource-scheduling strategy aimed at addressing the limitations of the logical convergence approach [20]. OpenStack [21] and Kubernetes [22] frameworks are employed to manage containers and VMs, respectively. Additionally, the Prometheus [23] framework is utilized to manage physical resources and intelligently schedule clusters based on resource utilization. By implementing this scheduling policy, the results demonstrate reduced energy consumption and enhanced resource utilization, thereby improving overall efficiency.

4 Logical VM and Container Resource Scheduling Optimization

In the previous section, we introduced the background of the logical fusion strategy of containers and virtual machines, as well as their partial shortcomings. In this chapter, a detailed introduction is provided to the newly proposed resource scheduling optimization strategy based on the fusion of virtual machines and containers.

4.1 Strategy Overview

To address the issues of low resource utilization, complex deployment and management, and high energy consumption observed in logical convergence, a resource scheduling optimization strategy was proposed. Containers and VM clusters were deployed separately, and the compute-node servers in each cluster were monitored. If an underloaded cluster was identified, the VM and/or container instances were relocated, and the vacated hardware was placed into hibernation. Subsequently, overburdened clusters can utilize the hardware resources by bringing them out of hibernation. The load-balancing result leads to reduced energy consumption and improved resource optimization.

4.2 Agent Architecture

The initial step in the deployment process involved setting up OpenStack and Kubernetes as the management frameworks. Subsequently, the Prometheus framework was deployed to monitor resource utilization. Our novel converged scheduling (CS) middleware was then introduced, which interfaced with the OpenStack and Kubernetes application program interfaces (APIs). It retrieved cluster monitoring data collected by Prometheus and performed resource-based scheduling and management of containers and VMs. The architectural layout of this system is illustrated in Fig. 4.

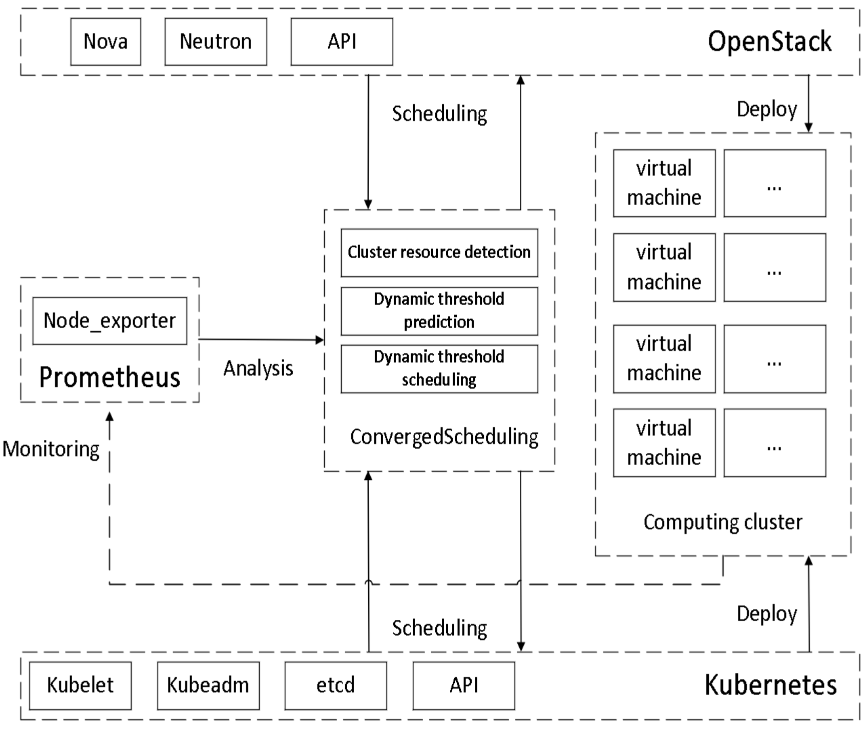

Fig. 4. Resource scheduling policy based on the logical convergence of virtual machines and containers

4.3 Core Algorithms

(1) Cluster Resource Detection

Resource monitoring and data collection in CS were performed through continuous polling of computational nodes in container and VM clusters [24]. A shell acquisition script was utilized to collect hardware resource data (e.g., central processing unit (CPU), memory, and power) [25]. These data were subsequently linked to the Prometheus time-series database, and real-time and historical resource data from each computing node of every cluster were analyzed using a PromQL query through the Prometheus API [26]. Fig. 5 illustrates our cluster resource detection algorithm. The memory consumption of compute node i is determined as follows:

$$U^{i}_{memory} = \frac{M^{i}_{total} - M^{i}_{\text{available}}}{M^{i}_{total}} \times 100\% , \tag{1}$$

where M^i_{total}, $M^i_{avaliable}$, and U^i_{memory} represent the total memory, usable memory, and memory consumption of node *i*, respectively.

The full CPU utilization of node i is determined as follows:

$$U^i_{cpu} = \frac{C^i_{total} - C^i_{\text{idle}}}{C^i_{total}} \times 100\% , \quad (2)$$

where C^i_{total}, C^i_{idel}, and U^i_{cpu}. represent the total, idle, and running CPU utilization of node i, respectively.

The energy use of node i is determined as follows:

$$U^i_{energy} = \sum\nolimits_{j=1}^{n} E^i_j , \quad (3)$$

where U^i_{energy} represents the energy consumption of node i, E^i_j isnergy expenditure of all processes j, and n is the total number of processes.

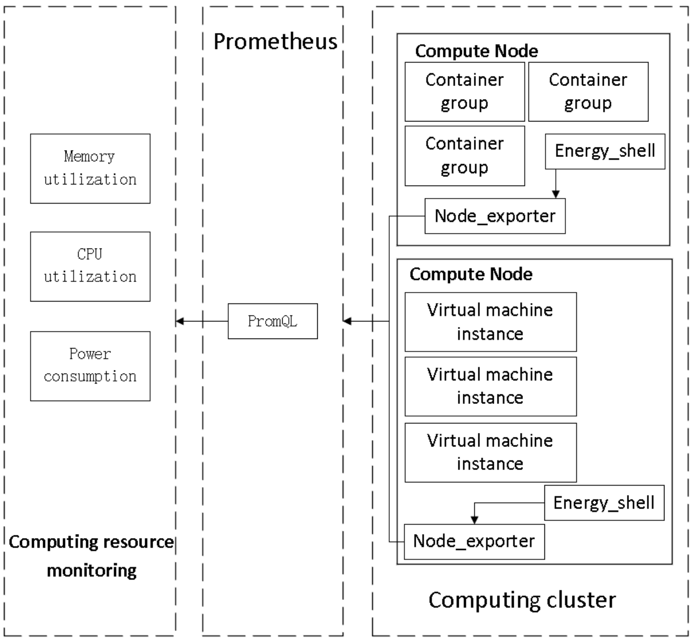

Fig. 5. Our cluster resource detection algorithm

(2) Dynamic Threshold Prediction

To detect instances of under- or over-utilization of resources in the compute node, a threshold must be established. Therefore, our algorithm utilizes the mean and standard deviation to predict a reasonably broad threshold range while accounting for the comprehensive historical data of resource utilization per cluster type. The threshold range can be further adjusted based on real-time performance data collected from all nodes. This approach is facilitated by the continuous polling of cluster resource data.

Compute node i is characterized by its CPU resource consumption U^i_{cpu} , memory resource utilization U^i_{memory}, and total resource utilization $U^i_{synthetical}$. The weight coefficient of the CPU is represented by C_{weight}, and the memory weight coefficient is denoted as M_{weight}. These weight coefficients are empirical parameters that are set based on the specific application scenario. In the current algorithm, the CPU weight is assigned a value of 0.3, whereas the memory weight is set to 0.7. Equation (4) is employed to calculate the total resource utilization while accounting for the weighted contributions of CPU and memory.

$$U^i_{\text{synthetical}} = U^i_{cpu} \times C_{\text{weight}} + U^i_{memory} \times M_{\text{weight}} \,. \tag{4}$$

The pseudocode of the dynamic threshold prediction algorithm is as follows:

```
for cloud in cluster do
   for i in cloud do
    get $U^i_{\text{synthetical}} = U^i_{cpu} \times C_{\text{weight}} + U^i_{memory} \times M_{\text{weight}}$
    get $U^i_{\text{hs}}$
    compute $U^i_{\text{avg}}$ from $U^i_{\text{hs}}$
    compute $U^i_{\text{std}}$ from $U^i_{\text{hs}}$
    $U^i_{\text{upt}} = U^i_{\text{avg}} + U^i_{\text{std}}$
    $U^i_{\text{lot}} = U^i_{\text{avg}} - U^i_{\text{std}}$
    get $U^i_{\text{osavg}}$
    compute $U^i_{\text{upt}}$ and $U^i_{\text{lot}}$ from $U^i_{\text{osavg}}$
endfor
endfor
```

In the first step, the VM and container clusters are traversed to collect the most recent comprehensive resource utilization metric $U^i_{\text{synthetical}}$ for each computing node. Subsequently, the historical comprehensive resource utilization U^i_{hs} is gathered for each computing node. The average resource utilization U^i_{avg} is then determined using the historical data U^i_{hs}. The standard deviation is also computed based on the historical integrated resource usage U^i_{hs}. These values are then used to determine the upper and lower thresholds, U^i_{upt} and U^i_{lot}, respectively. The threshold can be fine-tuned by adding or subtracting the mean and variance over time.

The threshold can be made more broadly representative, dynamically adaptable, and statistically stable by estimating the average resource usage of the other hosts, U^i_{osavg}. U^i_{upt} and U^i_{lot} are then modified based on the instance measures of U^i_{osavg}, where U^i_{upt} is increased by 0.1 and U^i_{lot} is decreased by 0.1, if $U^i_{\text{synthetical}}$ is greater than U^i_{osavg}. When $U^i_{\text{synthetical}}$ falls below U^i_{osavg}, U^i_{upt} decreases by 0.1 and U^i_{lot} increases by 0.1.

(3) Dynamic Threshold Scheduling

The pseudocode of the dynamic threshold scheduling algorithm is as follows:

```
for cloud in cluster do
   for i in cloud do
    get $U^i_{\text{synthetical}}$
    get $U^i_{\text{upt}}$ and $U^i_{\text{lot}}$
    if $U^i_{\text{synthetical}} > U^i_{\text{upt}}$
      add $U^n$
      migrate $U^i_{\text{cvm}}$ into $U^n_{\text{cvm}}$
    if $U^i_{\text{synthetical}} < U^i_{\text{lot}}$ .
      if $U^i_{\text{cvmc}} < U^o_{\text{cvmc}}$
        migrate $U^i_{\text{cvm}}$ into $U^o_{\text{cvm}}$
        delete $U^i$
      else:
        adjust $U^i_{\text{upt}}$
        end
      endfor
    endfor
```

Each compute node U^i in the cluster cloud is evaluated to determine its load status based on the threshold range generated by the dynamic threshold prediction algorithm. If a node is identified as being overloaded, a sleeping compute node U^i is awakened from hibernation to rejoin the cluster, and the VM instance or container U^i_{cvm} at the overloaded node is migrated to the newly added compute node U^n_{cvm} to alleviate the overload. U^i_{cvm} is then moved to another compute node U^o_{cvmc}, if it is overloaded. Prior to migration, the method determines whether the other nodes in the cluster have sufficient U^o_{cvmc} to accept U^i_{cvmc} . The host is then assessed using a dynamic threshold prediction technique, and its underload threshold U^i_{upt} is increased to ensure it is no longer overloaded. Upon successful migration, compute node U^i departs the cluster, clears all its cached data, and enters into hibernation.

5 Experiment

5.1 Experimental Environment

In this study, the effectiveness of our proposed resource scheduling model was evaluated by combining VMs and containers. To replicate a datacenter cluster environment, VMs were generated using VirtualBox. A total of 14 VMs were created in three different environments, and their precise configurations are provided in Table 1. The initial Centos7-equipped VM was configured, and the remaining 13 VMs were cloned to ensure consistent environmental conditions. The study compared the Kubernetes+Kubevirt and OpenStack+Kubernetes strategies, representing physical and logical convergence approaches, respectively. By conducting VM and container convergence studies, the performance and outcomes of these two strategies were assessed and compared.

In the experiments presented in this study, the CPU, memory, and energy usage of all compute nodes in a clustered environment were compared for each convergence technique. To ensure fair comparison, the exact identical test data were executed on each cluster setting. The test data were generated using the stress tool, which creates four container sets with random calls to free the CPU and random-access memory, and two VMs instances running a private Alpine Linux system.

Throughout the study, the performance metrics in each set of environments were continuously tracked using the Prometheus tool to enable the monitoring and analysis of various performance indicators alongside a comprehensive evaluation of the resource utilization and system performance for each convergence technique.

Table 1. Experimental environment configuration

Policy	Host	Host IP	CPUs	Memory (MB)	Hard disk (GB)	Operating system	Node type
Kubernetes +Kubevirt	K8sControllerKubevirt	10.20.0.27	2	2048	80	Centos7	control
	K8sComputeNode1	10.20.0.10	2	4000	80	Centos7	calculate
	K8sComputeNode2	10.20.0.11	2	4000	80	Centos7	
	K8sComputeNode3	10.20.0.12	2	4000	80	Centos7	
OpenStack +Kubernetes	K8sController	10.20.0.28	2	2048	80	Centos7	control
	OpenStackController	10.20.0.09	2	5000	80	Centos7	
	K8sOsComputeNode1	10.20.0.13	2	4000	80	Centos7	calculate
	K8sOsComputeNode2	10.20.0.14	2	4000	80	Centos7	
	K8sOsComputeNode3	10.20.0.15	2	4000	80	Centos7	
OpenStack +Kubernetes +rsm	K8sController	10.20.0.29	2	2048	80	Centos7	control
	OpenStackController	10.20.0.08	2	5000	80	Centos7	
	K8sOsComputeNode1	10.20.0.16	2	4000	80	Centos7	calculate
	K8sOsComputeNode2	10.20.0.17	2	4000	80	Centos7	
	K8sOsComputeNode3	10.20.0.18	2	4000	80	Centos7	

5.2 Experimental Analysis

Each environment's test data and results were distinct, and three indicators were assessed at identical times in each trial. Each environment's processing node's median energy consumption, CPU, and memory usage totals were tracked for the period from the start of the test for 10 min. This collection technique contributed to the accuracy and stability of the results in comparison to other methods.

The findings in Fig. 6 indicate that the compute node using the RSC policy exhibited significantly lower energy consumption compared with the other policies. Similarly, in Fig. 8, the RSC policy at the K8sComputeNode1 and K8sComputeNode2 demonstrated more effective memory utilization than the other policies. Fig. 7 shows that the RSC policy achieved better CPU utilization at the K8sComputeNode1 compared with the other policies. Additionally, in terms of CPU consumption at the K8sComputeNode2, the RSC policy outperformed the OpenStack+Kubernetes approach.

Table 2 provides a comprehensive comparison, revealing that the RSC method achieved a 39.96% reduction in energy consumption for cluster compute nodes compared with Kubernetes+Kubevirt. Moreover, the RSC method was 33.3% more cost-effective than the OpenStack+Kubernetes approach. The OpenStack+Kubernetes strategy exhibited a 0.71% improvement in overall CPU utilization compared with the other approaches, whereas its memory utilization was 8.37% better. Compared with Kubernetes+Kubevirt, the improvement in memory utilization was 2.15%. In summary, the RSC technique demonstrates significant advantages in terms of lower energy consumption and improved resource utilization.

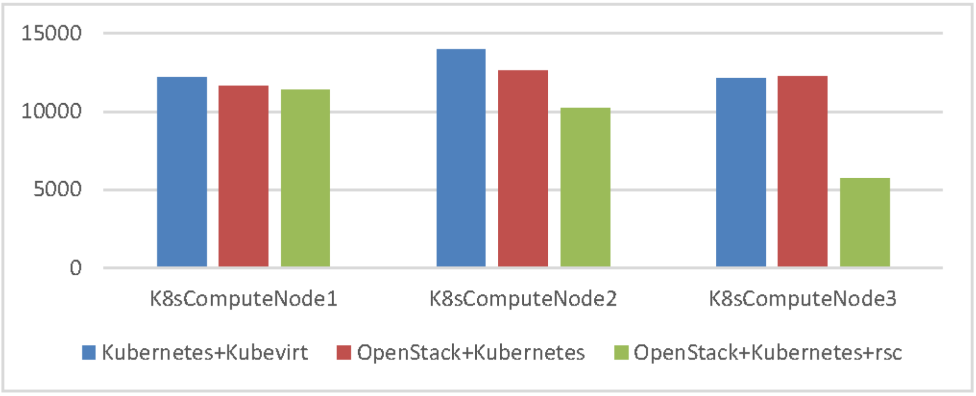

Fig. 6. Comparison of energy consumption of fusion strategy calculation nodes

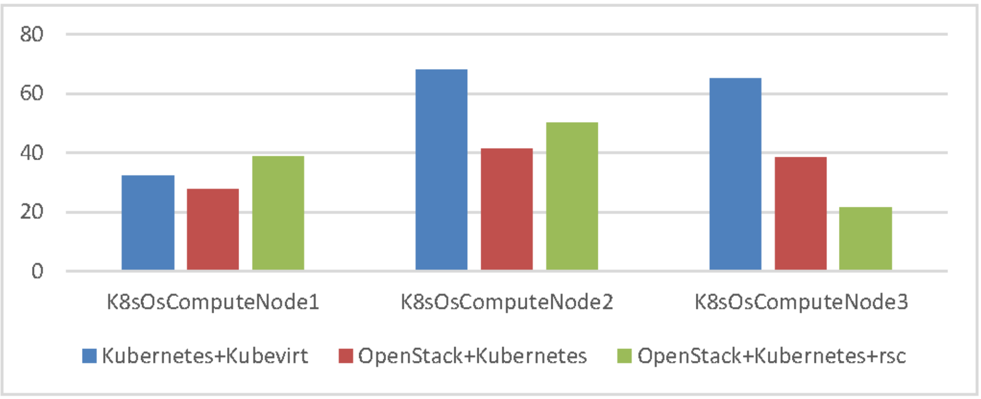

Fig. 7. Comparison of CPU utilization of fusion strategy computing nodes

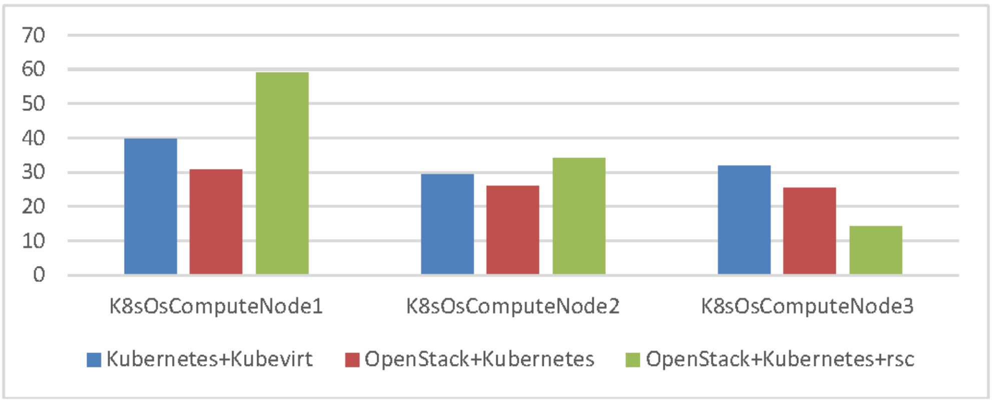

Fig. 8. Comparison of memory utilization of fusion strategy computing nodes

Table 2. Comparison table of overall performance of fusion strategy calculation nodes

Policy name	Total energy consumption (mw)	Total CPU utilizatio (%)	Total memory utilization (%)
Kubernetes+Kubevirt	38316.3	54.28	33.62
OpenStack+Kubernetes	36495.1	35.5	27.4
OpenStack+Kubernetes+rsc	27376.3	36.21	35.77

6 Conclusion

This study introduced a novel technique for fusing VMs and containers that effectively leveraged the advantages of virtualization while addressing its limitations through a logical conversion strategy. The experimental comparison between the conventional Kubernetes+Kubevirt technique and OpenStack+Kubernetes demonstrates significant benefits in terms of reduced energy consumption, improved resource utilization, and enhanced system performance.

As part of future work, the threshold scheduling method of our strategy will be enhanced by incorporating machine learning techniques to achieve more accurate threshold prediction. Additionally, the scheduling algorithm will be modified to improve migration efficiency, and the entire fusion architecture will be optimized for better utilization of idle resources. Finally, the integration of cluster compute nodes will be further enhanced to ensure seamless operation and improved overall performance.

References

[1] Z.-X. Wu, Advances on virtualization technology of cloud computing, Computer Applications 37(4)(2017) 915-923.

[2] B.-H. Yang, W.-J. Dai, Y.-L. Cao, Docker Technology Introduction And Actual Combat, Beijing: China Machine Press, 2015.

[3] J.S. Plank, Erasure codes for storage systems: A brief primer, login: the magazine of USENIX & SAGE 38(6)(2013) 44-50.

[4] J. Bell, Apache Spark, in: J. Bell (Ed.), Machine Learning: Hands-On for Developers and Technical Professionals, John Wiley & Sons, Inc., 2020 (Chapter 13). https://doi.org/10.1002/9781119642183.ch13

[5] Y. Li, W. Li, C. Jiang, A survey of virtual machine system: Current technology and future trends, in: Proc. 2010 Third International Symposium on Electronic Commerce and Security, 2010.

[6] K. Gudeth, M. Pirretti, K. Hoeper, R. Buskey, Delivering secure applications on commercial mobile devices: the case for bare metal hypervisors, in: Proc. of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, 2011.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of

virtualization, ACM SIGOPS operating systems review 37(5)(2003) 164-177.

[8] A. Kivity, Y. Kamay, D. Laor, U. Lublin, A. Liguori, kvm: the Linux virtual machine monitor, in: Proc. of the Linux Symposium, 2007.

[9] T. Bell, B. Bompastor, S. Bukowiec, J.C. Leon, M.K. Denis, J. van. Eldik, M.F. Lobo, L.F. Alvarez, D.F. Rodriguez, A. Marino, B. Moreira, B. Noel, T. Oulevey, W. Takase, A. Wiebalck, S. Zilli, Scaling the CERN OpenStack cloud, Journal of Physics: Conference Series 664(2)(2015) 022003.

[10] B. Noel, D. Michelino, M. Velten, R. Rocha, S. Trigazis, Integrating containers in the CERN private cloud, Journal of Physics: Conference Series 898(9)(2017) 092045.

[11] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, W. Zhou, A comparative study of containers and virtual machines in big data environment, in: Proc. 2018 IEEE 11th International Conference on Cloud Computing, 2018.

[12] J.-T. Wu, L.-L. Chen, Z.-X. Li, Hyper-Integration Experimentation of Virtual Machine And Container, Computer Applications and Software 38(9)(2021) 10-15, 19.

[13] Z. Kozhirbayev, R.O. Sinnott, A performance comparison of container-based technologies for the cloud, Future Generation Computer Systems 68(2017) 175-182.

[14] G. Li, K. Takahashi, K. Ichikawa, H. Iida, C. Nakasan, P. Leelaprute, P. Thiengburanathum, P. Phannachitta, The Convergence of Container and Traditional Virtualization: Strengths and Limitations, SN Computer Science 4(4)(2023) 387.

[15] C. Zhao, L.-S. Yan, Y.-H. Cui, H.-L. Xing, B. Feng, Dynamic adjusting threshold algorithm for virtual machine migration, Journal of Computer Applications 37(9)(2017) 2547-2550.

[16] J. Zhang, M. Han, Cloud Native Virtual Computing Cluster, in: Proc. 2022 IEEE 8th International Conference on Cloud Computing and Intelligent Systems, 2022.

[17] A. Randazzo, I. Tinnirello, Kata containers: An emerging architecture for enabling MEC services in fast and secure way, in: Proc. 2019 Sixth International Conference on Internet of Things: Systems, Management and Security, 2019.

[18] I. Odun-Ayo, S. Misra, O. Abayomi-Alli, O. Ajayi, Cloud Multi-tenancy: Issues and Developments, in: Proc. 10th International Conference on Utility and Cloud Computing, 2017.

[19] J. Zhang, J. Wang, J. Wu, Z.-H. Lu, S.-Y. Zhang, Y.-P. Zhong, WARMOPS: A workload-aware resource management optimization strategy for IAAS private clouds, in: Proc. 2014 IEEE International Conference on Services Computing, 2014.

[20] I. Mavridis, H. Karatza, Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing, Future Generation Computer Systems 94(2019) 674-696.

[21] R. Kumar, N. Gupta, S. Charu, K. Jain, S.K. Jangir, Open source solution for cloud computing platform using OpenStack, International Journal of Computer Science and Mobile Computing 3(5)(2014) 89-98.

[22] M. Luksa, Kubernetes in Action, Simon and Schuster, 2017.

[23] N. Sukhija, E. Bautista, Towards a framework for monitoring and analyzing high performance computing environments using Kubernetes and Prometheus, in: Proc. 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation, 2019.

[24] A. Bhardwaj, C.R. Krishna, Virtualization in cloud computing: Moving from hypervisor to containerization-A survey, Arabian Journal for Science and Engineering 46(9)(2021) 8585-8601.

[25] X.-X. Wang, X.-F. Wang, Y. Liu, OpenStack-based Docker Scheduling Model with High Resource Utilization, Computer Engineering 48(9)(2022) 171-179.

[26] N. Sabharwal, P. Pandey, Working with Prometheus Query Language (PromQL), in: Monitoring Microservices and Containerized Applications, Apress, Berkeley, CA, 2020 (pp. 141-167). https://doi.org/10.1007/978-1-4842-6216-0_5