MyKSC: Disaggregated Containerized Supercomputer Platform

Ju-Won Park(✉), Joon Woo, and Taeyoung Hong

Korea Institute of Science and Technology Information,
245 Daehak-ro, Daejeon 34141, Republic of Korea
{juwon.park,wjnadia,tyhong}@kisti.re.kr

Abstract. Recently, with the emergence of cloud-based machine learning application services such as ChatGPT and big data analytics, which require high-volume data learning, demand for large-scale high-performance computing (HPC) resources has significantly increased. However, there are many difficulties in providing cloud-based services owing to the differences in existing HPC resource operation and execution environments. To address these challenges, there are many ongoing research efforts to support cloud application services on HPC systems using container technology. However, there are fundamental differences between traditional HPC and cloud applications that make administrators hesitant to adopt them in actual operating environments. To solve these problems, this paper introduces MyKSC, which provides a unified user interface based on a loosely-coupled architecture. MyKSC is a system applied to `Nurion`, a supercomputer operated by the Korean National Supercomputing Center (KISTI). `Nurion` users can use new cloud application services such as Jupyter and RStudio along with existing HPC application services like MPI through MyKSC. To achieve this, MyKSC divides the entire system into Kubernetes and HPC clusters and selects suitable cluster resources to run the application based on user preferences. This paper proposes the loosely-coupled containerized supercomputer platform and introduces its current implementation.

Keywords: Loosely-coupled · containerization · unified interface

1 Introduction

Recently, with the emergence of cloud-based machine learning application services such as ChatGPT and big data analytics, which require high-volume data learning, the demand for large-scale high-performance computing (HPC) resources has significantly increased. However, supporting cloud application services through existing HPC resources is challenging owing to the rigid software stack [8,13]. Container technology can be a very good alternative to solving

This work was supported by the Korea Institute of Science and Technology Information (Grant No. K-23-L02-C01-S01).

Y. Zhang and L.-J. Zhang (Eds.): ICWS 2023, LNCS 14209, pp. 83–91, 2023.
https://doi.org/10.1007/978-3-031-44836-2_6

these problems. Because it shows very good performance through a lightweight virtualization layer, many studies are being conducted to introduce container technology in the HPC field [3,4,9]. However, despite these efforts, there are restrictions due to fundamental differences between typical HPC and cloud applications, including resource utilization, the scale of tasks and service execution times, and the way services are executed.

To solve these problems, this paper introduces MyKSC, which provides a unified user interface based on a loosely-coupled architecture. MyKSC is a system applied to `Nurion`, a supercomputer operated by KISTI. `Nurion` users can use new cloud application services such as Jupyter and RStudio along with existing HPC application services like MPI through MyKSC. MyKSC divides the entire system into a Kubernetes cluster for providing cloud application services and a traditional scheduler-based HPC cluster. Based on these divided resources, MyKSC selects suitable cluster resources to run the application depending on the type of service chosen by the user. That is, new cloud application services such as Jupyter and RStudio are executed on the Kubernetes cluster, while for traditional HPC applications, it generates a job script file based on user interaction (e.g., file selection, parameter input) and submits it to the existing HPC cluster's batch job scheduler to perform the actual task on the HPC cluster.

The remainder of this paper is organized as follows. Section 2 introduces `Nurion` and related studies. In Sect. 3, we present the architecture of the proposed platform. We next present the implementation of MyKSC in Sect. 4. Finally, we present our conclusions in Sect. 5.

2 Background

2.1 Specifications and Structure of the 5th Supercomputer

`Nurion` is the 5th HPC system built by the Korean National Supercomputing Center (KISTI) in 2018. It is a cluster system consisting of 8,305 Intel Xeon Phi-based KNL CPU nodes and 132 Intel Xeon SKL CPU nodes, with a theoretical performance of 25.7 *PFlops*. The user data are stored in a 20 *PBytes* Lustre parallel file system, and all compute nodes and parallel file systems are connected with a 100 *Gbps* omni-path architecture (OPA) interconnector that provides ultra-high-speed/low-latency communication. `Nurion` provides users with two personal folders: (*/scratch*) and (*/home*). */home* provides 64 GB of storage space for user-specific data and has no file deletion policy. By contrast, */scratch* is a temporary storage space for all files needed for task execution: it provides up to 100 TB of space but automatically deletes unused files every 15 days. To prevent performance degradation in the parallel file system, a burst buffer was introduced. The burst buffer is a cache layer for I/O acceleration between compute nodes and storage, preventing performance degradation due to small I/O or random I/O in the parallel file system and maximizing parallel I/O performance. In addition, `Nurion` uses the portable batch system (PBS) as the batch job scheduler for workload management, and an exclusive node allocation policy is applied by default to ensure that only one user's task can be executed per node to guarantee the maximum application performance.

2.2 Related Work

Existing HPC cluster systems are built and operated in a form optimized for parallel programs, which imposes many restrictions on supporting new types of services. To overcome these rigid utilization problems and provide users with a more diverse service environment, researchers have conducted numerous studies to configure clusters using cloud virtualization technology. Virtualization technology can be divided into hypervisor-based virtualization technology and container-based virtualization technology. However, as many recent studies have shown, hypervisor-based virtualization technology inevitably results in performance degradation, whereas container-based virtualization technology is known to achieve performance close to native [5,6,11]. Consequently, researchers are conducting many studies to secure the flexibility of HPC resources by utilizing container technology in the HPC field. Notable container application technologies in the HPC field are Singularity, Shifter, and Charliecloud [3,4,9]. Singularity was specifically designed from the outset for HPC systems. Singularity [4] has a special file format (called the Singularity Image Format or SIF) to support novel features such as security, extreme portability, and guaranteed reproducibility. Shifter [3] is a prototypical implementation of container engine for HPC developed by NERSC. Shifter allow to deploy user-created images at large scale. It can support Docker images as well as several other standard image formats (vmware, ext4, squashfs, etc.) and is tied into the batch system at NERSC. CharlieCloud [9] is an open-source container technology developed by Los Alamos National Laboratory for supercomputing clusters. It uses the Linux user and mount namespaces to run industry-standard Docker containers with no privileged operations or daemons.

Recently, there has been an increase in cases of HPC operating centers running Kubernetes clusters alongside HPC clusters [2,10]. For example, the Ohio Supercomputer Center has deployed a Kubernetes cluster with tight integration to a high performance computing (HPC) environment [2,7]. Purdue University provide a composable infrastructure to launch container-based applications with Kubernetes [10]. Oak Ridge National Laboratory also operate the OpenShift environment with kubernetes [7].

3 Loosely-Coupled Integration of Kubernetes on Supercomputer

Many previous studies have attempted to integrate cloud applications with HPC systems in a tightly-coupled structure [3,4,9]. However, there are limitations due to fundamental differences between typical HPC and cloud applications. First, there are differences in how resources are utilized. Typical HPC applications require very large resources and high performance, mainly relying on exclusive resource allocation. By contrast, cloud application services mainly employ on-demand, shared resource allocation. Therefore, batch job schedulers such as PBS Pro and SLURM, widely used in HPC systems, are insufficient to meet these requirements.

Second, there are differences in the scale of tasks and service execution times. HPC applications usually execute very large-scale tasks, allocating large amounts of resources and running for very long periods. Therefore, most supercomputer operation centers set a wall-time limit to ensure fair resource usage and prevent tasks from running beyond the limit. Consequently, users must save checkpoints at regular intervals to resume their tasks from the saved point if the workload manager forcibly terminates them. By contrast, cloud applications usually involve multiple micro-services connected and executed through REST APIs. These micro-services flexibly start and stop as needed, and connections may frequently break due to various issues; hence, the service dynamically finds and connects to another micro-service. Therefore, rather than checkpoint techniques, cloud applications require methods that periodically monitor the execution status of services and dynamically create, connect, and terminate services.

Third, there are differences in the way services are executed. Performance is the most critical factor in HPC applications. To satisfy performance requirements, all compute nodes have the packages and libraries necessary for running HPC applications preinstalled, and even kernel parameters are customized. As a result, HPC applications execute jobs in batches within this optimized, closed environment. By contrast, cloud applications require a highly flexible execution approach, dynamically providing the execution environment to meet diverse user requirements. Specifically, in the cloud, rather than customizing all nodes for specific applications, flexibility is enhanced through a hypervisor layer to provide diverse application environments, even if it sacrifices performance.

Due to these fundamental dissimilarities between HPC and cloud applications, supercomputing centers operating actual systems are hesitant to adopt tightly-coupled container technologies. To address this issue, we introduce MyKSC, a loosely-coupled architecture. MyKSC is a platform that applies container technology to support the cloud-based AI and data analytics applications, which are increasing in popularity, on the supercomputer `Nurion`. It provides both traditional HPC applications and cloud-based applications through a unified interface. The architecture of MyKSC has three primary objectives: first, to minimize changes to existing operational HPC systems and software structures; second, to provide both HPC and cloud applications through a unified interface; and third, to enable access to the same data across all services.

Figure 1 shows the architecture of MyKSC. To provide cloud services, the entire system's resources are divided and operated between Kubernetes and traditional HPC clusters, isolating them from each other. Thus, the existing HPC cluster software remains unchanged and coexists with the Kubernetes cluster. MyKSC also controls and manages the resources of the two clusters through a web-based unified interface. It selects the appropriate cluster resources to execute applications based on the type of service that users require and delivers the results to users through a web browser. Traditional HPC applications create a job script file using the web-based interface provided by MyKSC and submit it to the batch job scheduler. The submitted job is executed through the HPC clusters, then the executed results are saved in the parallel file system. The saved results can be accessed directly allowing users to add, modify, or delete their data in MyKSC.

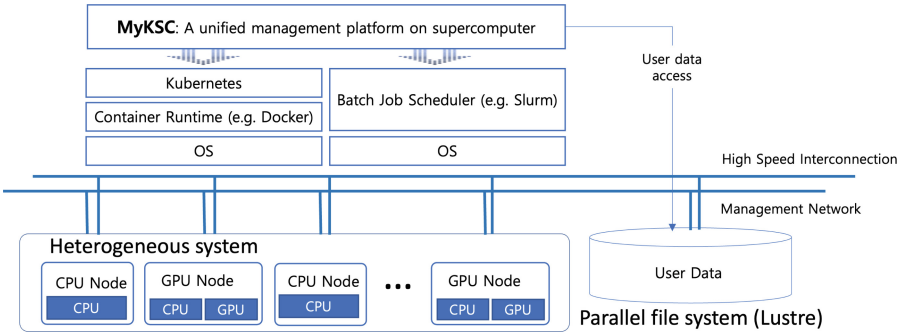

Fig. 1. Architecture of MyKSC.

On the other hand, cloud-based applications are provided in a containerized manner through a Kubernetes cluster. In MyKSC, the services provided as cloud services are jupyter, RStudio, and VNC. These applications are dynamically created through kubernetes when a user requests a service and deleted when the service is terminated.

4 Implementation

In MyKSC, the Kubernetes infrastructure is built using the software stack for cloud application services as shown in Table 1.

Table 1. MyKSC software stack

Software	Application Name	Version
OS	CentOS	7.9
Container Runtime Interface (CRI)	docker	20.10.17
CRI for GPU	nvidia-docker	2.11.0
container orchestrator	Kubernetes	1.23.9
Container Network Interface plugin	Calico	3.24.5
Service Mesh	Istio	1.15.1
Load Balancer	MetalLB	0.13.9

Kubernetes is the de facto standard framework for container orchestrators, with a rapidly growing community and ecosystem. Recently, it is also widely used in HPC systems [1,2,12]. Kubernetes consists of masters and workers, and services are executed as pods comprising one or multiple containers. In MyKSC, we chose Docker and Nvidia-docker as the container runtime interfaces. Docker is the most widely used container runtime engine, and Nvidia-docker is a container

runtime engine developed by Nvidia for managing NVIDIA GPU-based containers (Docker will no longer be supported as a CRI from Kubernetes Version 1.24, and it will be replaced with cri-o or containerd in future upgrades). Calico was used for pod networking and policy management. Calico supports both L2 connections based on XVLAN and L3 connections through BGP routing, making it a suitable choice for KISTI's current network environment. MetalLB was installed and configured to provide load balancing and high availability of services, while Istio was chosen for the ingress controller.

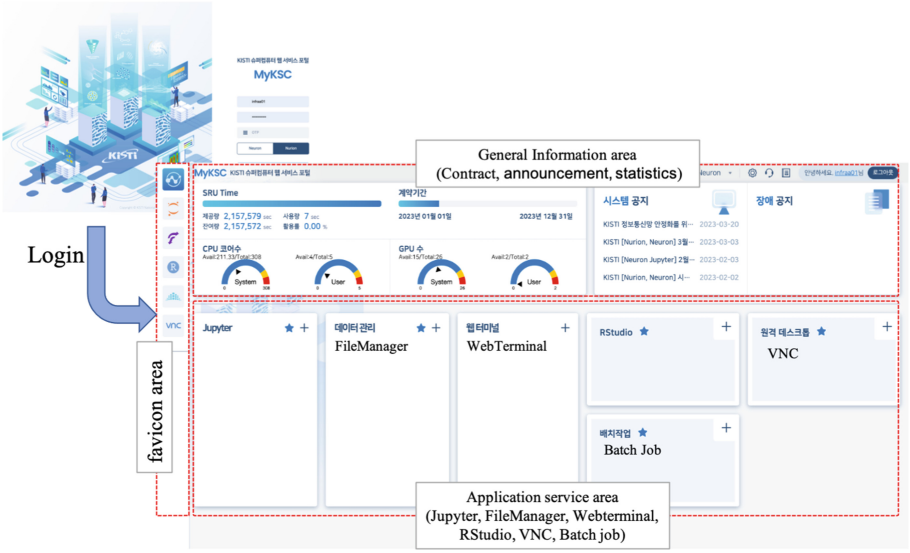

Fig. 2. MyKSC dashboard.

After logging in with a basic ID/password and two-factor authentication, users can view the dashboard shown in Fig. 2, which is the Graphical User Interface (GUI) of the implemented MyKSC. The MyKSC dashboard is divided into three sections. On the left, icons of favorites are placed for users to easily create and navigate their desired services. The top right section displays the user's contract period and remaining resources, along with statistics on Kubernetes cluster resources (CPU, Memory, GPU). The bottom right section shows the services currently provided through MyKSC, and users can create new services by clicking the "+" button. As shown in Fig. 3, created services are provided to users within the same browser frame in multiple windows.

Figure 4 shows the configuration and service creation method of MyKSC. Services are executed as pods in the Kubernetes infrastructure and created as a deployment type with three replicas for high availability and load balancing of network traffic. The created deployment connects to the outside through Istio's secure gateway, using the public IP assigned by MetalLB. SSL certificates

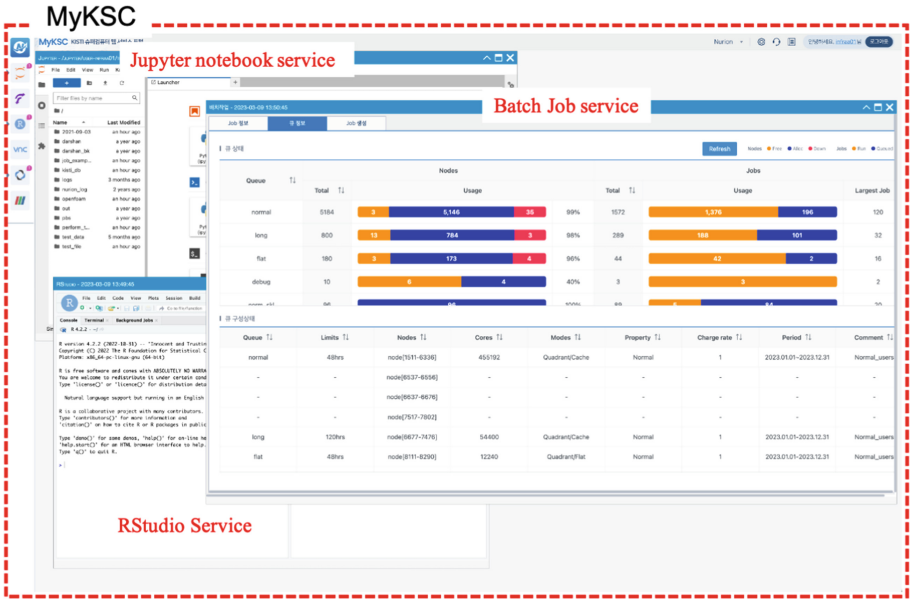

Fig. 3. The screenshot of MyKSC.

for secure communication are stored in Istio. By utilizing Istio's secure gateway, services created through MyKSC can run without an SSL certificate. In other words, users establish secure connection using the HTTPS secure protocol up to the secure gateway, and the connection between the secure gateway and the actual service uses the HTTP protocol. Also, since MyKSC does not use database, sessions cannot be shared between the created replicas. Therefore, Istio's DestinationRule fuction is used to maintain connections to the same pod for requests with a certain session.

The service creation method of MyKSC is as follows. First, when a user logs in, MyKSC retrieves the user's UID, GID, and username from LDAP and creates a namespace in the Kubernetes cluster for the logged-in user. After logging in, all Kubernetes resources created by the user are created and managed within the user's namespace. MyKSC provides a total of six services; five of them, excluding the webTerminal service that simply connects to the login node as a web terminal, are divided into the following three categories:

Jupyter, RStudio, and VNC: When a user creates Jupyter, RStudio, or VNC services, MyKSC creates pods and services through the kube-API server. When creating a pod, the "runAsUser" and "runAsGroup" values are set to the user's UID and GID values obtained from LDAP for security. These created pods are connected to clients in a sub-path format through the API Gateway implemented within MyKSC. Thus, a page requested with /jupyter connects to the Jupyter pod, and a page requested with /rstudio connects to the RStudio pod.

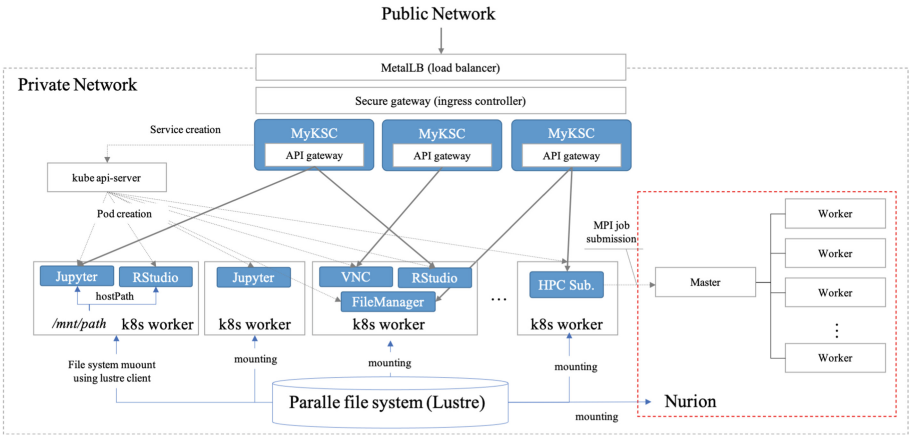

Fig. 4. The service creation process of MyKSC.

FileManager: FileManager provides a web-based file upload/download feature to manage files based on a GUI. MyKSC offers access control to two private directories provided by `Nurion` in the form of *hostPath* with the FileManager pod. As mentioned in Sect. 2, in `Nurion`, all compute nodes have the parallel file system, Lustre mounted as $/home/\{userName\}$ and $/scratch/\{userName\}$. Therefore, when creating a FileManager pod in MyKSC, it is possible to directly access the two private directories of the parallel file system by attaching $/home/\{userName\}$ and $/scratch/\{userName\}$ as *hostPath* volume types to the FileManager pod, along with setting the "runAsUser" and "runAsGroup" values.

Batch Job: In this study, for batch job services, integration is performed in a loosely-coupled structure with the existing HPC cluster system, as presented in Sect. 3. That is, the batch job pod created in the Kubernetes cluster receives user input through a web browser and generates a job script file. The created job script file is submitted to the job scheduler's master with the user's UID, and the actual job is executed on the existing HPC cluster. However, the executed results can be verified by the user through the batch job pod.

5 Conclusion

Due to the fundamental differences between HPC application services and cloud application services, tightly-coupled integration approaches have clear restrictions, causing hesitation in their adoption by administrators. To address these challenges, this paper presented MyKSC, which provides a unified user interface based on a loosely-coupled architecture. MyKSC is applied to `Nurion`, a supercomputer operated by KISTI, allowing `Nurion` users to use new cloud application services such as Jupyter and RStudio, along with traditional HPC application services like MPI, through MyKSC.

References

1. Beltre, A.M., Saha, P., Govindaraju, M., Younge, A., Grant, R.E.: Enabling HPC workloads on cloud infrastructure using kubernetes container orchestration mechanisms. In: 2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC), pp. 11–20. IEEE (2019)
2. Dockendorf, T., Baer, T., Johnson, D.: Early experiences with tight integration of kubernetes in an HPC environment. In: Practice and Experience in Advanced Research Computing, pp. 1–4 (2022)
3. Gerhardt, L., et al.: Shifter: containers for HPC. In: Journal of physics: Conference Series, vol. 898, p. 082021. IOP Publishing (2017)
4. Godlove, D.: Singularity: simple, secure containers for compute-driven workloads. In: Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning), pp. 1–4 (2019)
5. Kozhirbayev, Z., Sinnott, R.O.: A performance comparison of container-based technologies for the cloud. Futur. Gener. Comput. Syst. **68**, 175–182 (2017)
6. Le, E., Paz, D.: Performance analysis of applications using singularity container on SDSC comet. In: Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact, pp. 1–4 (2017)
7. Papadimitriou, G., Vahi, K., Kincl, J., Anantharaj, V., Deelman, E., Wells, J.: Workflow submit nodes as a service on leadership class systems. In: Practice and Experience in Advanced Research Computing, pp. 56–63 (2020)
8. Park, J.W., Hahm, J.: Container-based cluster management platform for distributed computing. In: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), p. 34. The Steering Committee of The World Congress in Computer Science, Computer ... (2015)
9. Priedhorsky, R., Randles, T.: Charliecloud: unprivileged containers for user-defined software stacks in HPC. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–10 (2017)
10. Smith, P.M., et al.: The "geddes" composable platform-an evolution of community clusters for a composable world. In: 2020 IEEE/ACM International Workshop on Interoperability of Supercomputing and Cloud Technologies (SuperCompCloud), pp. 33–38. IEEE (2020)
11. Torrez, A., Randles, T., Priedhorsky, R.: HPC container runtimes have minimal or no performance impact. In: 2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC), pp. 37–42. IEEE (2019)
12. Zhou, N., et al.: Container orchestration on HPC systems through kubernetes. J. Cloud Comput. **10**(1), 1–14 (2021)
13. Zhou, N., Zhou, H., Hoppe, D.: Containerisation for high performance computing systems: Survey and prospects. IEEE Trans. Softw. Eng. **49**, 2722–2740 (2022)