Journal of Computer Science

Original Research Paper

Framework for Containers Orchestration to handle the Scientific Workloads using Kubernetes

Manish Kumar Abhishek, D. Rajeswara Rao and K. Subrahmanyam

Department of Computer Science, Koneru Lakshmaiah Education Foundation, Vaddeswaram, India

Article history
Received: 27-03-2022
Revised: 11-06-2022
Accepted: 22-07-2022

Corresponding Author:
Manish Kumar Abhishek
Department of Computer Science, Koneru Lakshmaiah Education Foundation, Vaddeswaram, India
Email: manish.9623727629@gmail.com

Abstract: The lightweight nature of Operating System virtualization give birth to the containers, and their high efficiency for application deployment prefers its usage in Cloud computing. Containers encapsulate and bundle the required dependencies for application development and deployment in libs packages as a single entity. The nature of containers eases application migration within the High-Performance Computing environment to handle the scientific workloads smoothly. Singularity containers are aimed to be only for High-Performance Computing (HPC) applications. The existing HPC workload managers are good with container monitoring, scheduling, and resource management, but container orchestration is always a concern. This study proposes a framework that will ease container orchestration using Kubernetes to handle the scientific applications workload. HPC cluster is built using interconnected containers in a private cloud environment. The architecture is derived based on the required configuration for containers to deploy the application and scheduling jobs.

Keywords: Cloud Computing, High-Performance Computing, Containers, Containers Orchestration, Resource Manager, Kubernetes

Introduction

Cloud computing is widely used across enterprise applications in the research area. It provides scalability, elasticity, fault tolerance, and secure infrastructure to develop and deploy applications. Different types of cloud models such as public, private, and hybrid; all are based on virtualization to serve the computing resources and instances to the individuals. Containerization is an emerging technology used to provide computing instances that replaces virtual machines in cloud computing to provide computing instances. It is based on Operating System (OS) virtualization and is easy to use for application deployment. Scientific applications with higher workloads are data-intensive and need high-performance computing resources in the Graphics Processing Unit (GPU), Central Processing Unit (CPU), and network. The cluster has multiple containers, where each container refers to single application bundling for the deployment. The number of applications increases results in a higher number of containers, raising the concern about its orchestration efficiency. The parallel computing application can be on the map-reduce technique, where the input data is divided into multiple tasks and gets executed in parallel. Once tasks get completed, all will be merged as a single output. To execute these tasks, we need multiple containers. It is going to be similar to microservices architecture. The auto-scaling of the infrastructure is required as big data applications are based on parallel computation. If there is a requirement to add a new library or jar based on application type, it will be challenging with the currently deployed application. The containers are the best solution to handle this kind of situation. The new version of the application or library will be added, and the computing instance will be launched without interruption. In the HPC cluster, the jobs hold huge workloads specific to the underlying nature of infrastructure. These jobs will be submitted in a queue in the form of batches and will be served on a first come, first basis. The resource and scheduler manager will decide to trigger these waiting jobs to get considered. These jobs will get assigned to the nodes based on the availability of resources and job priorities. The job priorities are defined in terms of numbers which refers to Critical, High, normal, and low priority, respectively. The higher the count, the lower the priority.

Container orchestration is an automated operation effort to manage its life cycle, including provisioning, monitoring, application deployment, and scalability (Al Jawarneh *et al*., 2019). It's mainly about scheduling, resources, and service management. Kubernetes provides a

platform to orchestrate the containers. Still, for scientific workloads, the requirement is quite different, which needs to be handled efficiently as all the running jobs are not going to consume all the assigned resources. Instead, another job can consume unused resources. In this study, we propose a framework that will leverage the existing resource manager and Kubernetes to fill the gap in container orchestration to handle the scientific workloads in the cloud environment. This framework will be based on the hybrid architecture of the cloud environment and HPC cluster, which internally uses Kubernetes. This study contributes to the high-performance computing area where we need to handle applications with a high volume of scientific workloads. The proposed framework is scalable and flexible, as Kubernetes is handling the job submission First, we will describe the existing resource managers, Kubernetes, and the related work. Next, we have captured the proposed framework and its architecture. Following the same, we evaluate the use cases and capture the results. Later, we covered the discussion and the conclusion of the research work.

Resource Managers

Conventional resource managers can manage the resources in extensive data analytics systems (Medel *et al*., 2018). The admin can configure the jobs with their expiry and specific configurations in GPU, memory, the number of processors, and the network. The resource allocation by the resource manager is done based on the granularity of the job node, which can be shared across jobs. There are mainly five job types: Malleable, moldable, rigid, evolving, and adaptive. The dynamic jobs responsible for big data set intensive applications are going to be of type adaptive one that will become adaptive at runtime based on the changes in resource allocation needs. The type requires the detailed formats of defined resources of rigid jobs where there is a need for a longer execution time for applications.

The efficient orchestration of containers will benefit the deployed high-performance computing application in terms of resource control (Rodriguez and Buyya, 2019). It will allocate the CPU/GPU to a container which will reframe the interference of other containers and help in scheduling by discovering the defined policies. It also distributes the load, provisions the containers in case of any failure or crashes, and helps in the application's scalability. The orchestration includes resource management, application management, and scheduling. There are multiple existing research managers described in the upcoming section.

Existing Resource Managers

A. Viewpoint

It is one of the most commonly used HPC-based resource managers. Using its flexible user interface, users can manage resources efficiently. Moreover, it is a visual-based interface specifically designed to increase the productivity of individuals (www.aspsys.com). Therefore, its dashboard is self-sufficient and explanatory to the users. The feature covers the job submission, application interface, script builder, job status details, file manager, 2Dimensional (2D)/3Dimensional (3D) visualizations, and throughput sessions using pre-defined templates.

B. SLURM

It refers to Simple Linux Utility for Resource Management (SLURM), a job and resource scheduling manager to manage the small and large HPC systems on top of the Linux operating system (https://slurm.schedmd.com/overview.html). There is no need for kernel modification. For the computation of nodes, it allocates exclusive and non-exclusive access. Once resources get allocated, jobs will be pushed into the queue, where they will get served one by one. Once the current job finishes, another job will be picked from the queue.

C. Torque

It refers to the Tera-scale Open-source Resource (https://adaptivecomputing.com) and QUEue manager (TORQUE), one of the most commonly used resource managers. It is used to control the HPC running jobs and nodes. It provides the high logging-based job scheduling user interface, the generated data after the completion of the job, and fault tolerance. It is highly scalable and used in mainframes and supercomputers for research work. The most commonly used commands are:

- Qstat – for checking the job's current status
- Qdel – delete a job once it gets finished
- Qsub – for job submission

D. Maui

It is a cluster resource manager that is highly optimized and easily configurable. It is a job scheduler with advanced features and is used to schedule the jobs based on the policies, fair share, dynamic priorities, and user interface to manage the resources for massive clusters. It helps to improve machine performance, ranging from smaller ones to higher processor-based clusters in teraflops (Zitzlsberger *et al*., 2018).

Figure 1 shows the TORQUE-based resource management for the HPC cluster having multiple containers to deploy the applications using the OS virtualization using the underlying infrastructure holding physical servers. The jobs on the computing nodes will be queues and served once resources are allocated based on their availability. It shows the general mechanism of managing the computing resources in the HPC cluster, where jobs are typically queued, and resources will get allocated once they get freed. The queue refers to the nodes partitioned in the form of different groups. TORQUE internally holds one primary node and the other multiple computing nodes. The job is submitted first to the primary node, and it is passed to the pbs server after allocating an id.

Later this job will be served to the scheduler. The scheduler will add this job to the queue based on a first-come, first-served basis, and the resource allocation and job priority will be defined. Then, based on the node's availability, the job details are returned to the pbs server. The first node will become the mother, and the other will become the sister node. The pbs server will get the resource allocation details from the job. It will pass the job control to the sister's nodes, installed on the superior, and instructed to provide the job on the multiple computing nodes. The superior node will monitor and manage the resource allocation to the containers. Whether success or error, the job status will be reported to the pbs server.

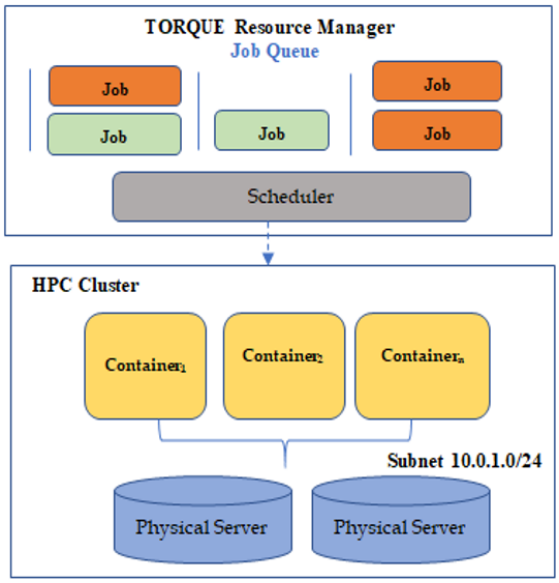

Fig. 1: Resource management using torque

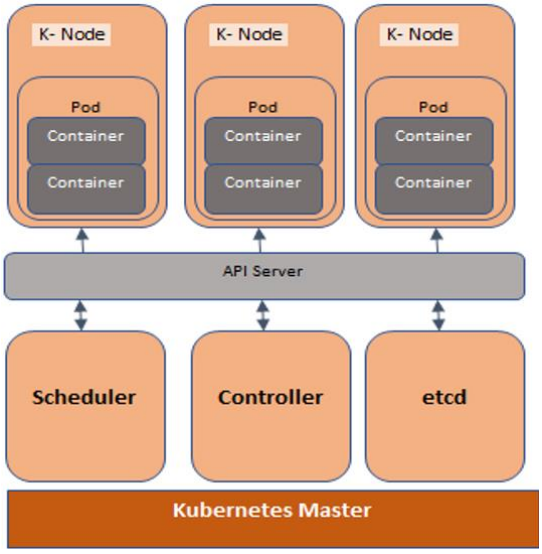

Fig. 2: Kubernetes using containers

E. Kubernetes

It is an open-source, most widely used platform for container orchestration (Jeffery *et al.*, 2021). It has its resource manager and job scheduler to allocate the resources and deploy the application bundled in containers. The deployed containers are called pods. The underlying container run time removed the Docker from the Kubernetes version 1.20 and onwards. It is fault-tolerance, highly scalable, efficient job scheduling, and a self-healed platform. It is based on the master and slave model. The primary node will be treated as a highly available fault-tolerant. The primary node will act as a control panel where slave nodes are the nodes where containers will get executed. Figure 2 shows the Kubernetes high-level component architecture diagram for container orchestration.

Related Work

Here, we are discussing a brief overview of our survey, which is done around the container's orchestration, which motivates us to develop this framework. Only a bunch of studies have been done as containerization is in its adopting phase for handling scientific workloads. The migration of computing resources at run time using HPC and Cloud Clusters is captured by Liu *et al.* (2018). IBM has also used the Spectrum Load Sharing Facility (LSF) workload manager to demo the Kubernetes pods execution (Liu *et al.*, 2018). Using the Grid Engine, Piras *et al.* (2019) have developed an approach to extend the Kubernetes on the top of HPC clusters (Piras *et al.*, 2019). A prototype has been proposed by Julian using the Moab scheduler (Julian *et al.*, 2016) to orchestrate the containers within the HPC environment. The python script is used to destroy the unused containers. Within the HPC cluster, an experiment has been performed by Wrede and Von Hof (2017) using the Docker Swarm. We propose a framework that will act as a bridge between the cloud and HPC systems using Kubernetes for container orchestration. It can be used for non-HPC applications, where GPU is not a requirement.

Proposed Model using Kubernetes

Our proposed framework is based on the model where the HPC Cluster will build using the cloud stack, i.e., private Cloud. The cluster is created using the provisioned computing instances in the form of high resource-based containers. As the scientific workloads need to be handled, GPU, network, CPU, and memory have been assigned accordingly. The Kubernetes will autonomously handle the container orchestration. Suppose there is a need to change the configuration or increase the higher resources at run time. In that case, the kubectl command will be run to deploy the new image of the application, which will provision the new container with defined resources and destroy the existing container only once the

new container is up and running. It is used to achieve the automatic deployment of containers and fault tolerance. Suppose the application breaches the higher limit at the run time, based on the prediction policy. In that case, a new container will be spawned with a threshold of memory in addition to what the previous container breaches the same based on the framework workflow design implementation. As Kubernetes is open source, we have leveraged its internal working mechanism to orchestrate the containers seamlessly in the cloud environment. We have used Linux Operating System-based containers to set up the HPC cluster and deploy the containers. The virtualization technology needs to be carefully chosen to handle the scientific workload in a cloud environment as it has its drawbacks to handle data-intensive applications. Using this framework, users can do changes in the Cloud-based execution stack and deploy their high data-sensitive application within the HPC cluster in the form of multiple containers to execute the tasks in parallel, similar to the map-reduce technique.

Figure 3 shows the proposed model architecture to orchestrate the containers for HPC Cluster. The job scheduler will schedule the queued jobs based on their priority. The scheduling layer is responsible for effectively using the resources. It takes the user inputs in terms of replication job placement cost rates. It will predict the placement of containers based on the defined policy at run time. It checks the readiness of the deployed application bundled in the form of a container. It will periodically check the heartbeat of the container for long-running applications. If the status is 1/1, the container is up and running and ready to accept the requests. It also takes care of the auto-restart of the containers on failure or crash, its rollback mechanism, handling the image upgrade version, and the containers' colocation. Table 1 shows the comparison of the scheduler layer with Kubernetes and other existing container orchestration tools.

The resource allocator is responsible for resource management in terms of memory, CPU, GPU, and network. It avoids interference between the containers for getting the resources. The service management layer is responsible for facilitating the environment to develop and deploy the applications. It holds the labels, which are the metadata info associated with containers, and namespaces as a unique identifier to identify the containers, their dependencies, readiness check, and load balancing to distribute the incoming load. Table 2 compares the service management layer with Kubernetes and other existing container orchestration tools.

Evaluation

Here, we will evaluate our proposed framework-based model for container orchestration using Kubernetes to handle the scientific workloads in the cloud environment. The jobs are scheduled and managed by the job scheduler and resource allocator. We have set up the cloud environment using OpenStack (Wang and Zhang, 2017) and built the HPC cluster. We have deployed three applications that are highly data-intensive applications. Later, we changed some business logic implementation and maintained a new version of the image. Kubernetes Cluster is set up to deploy the application in the form of containers. We have defined the resource configurations of an application in the form of multiple image versions in YAML files. Kubernetes will orchestrate the containers and take care of their whole lifecycle. Figure 4 shows the used Kubernetes commands to show the deployments and labels for the nodes.

We have evaluated the different performance metrics using a hybrid model to compare container orchestration. The Message Passing Interface (MPI) launcher at the cluster level will provide the containers where the internal application that will be deployed using containers is of type MPI. We have used the MPI library (Sultana *et al.*, 2021). At first, we considered the requirement of time for deploying our proposed framework. Next, we performed the analysis for service management and scheduling to evaluate readiness time consumption and recover and reschedule in case of fault tolerance.

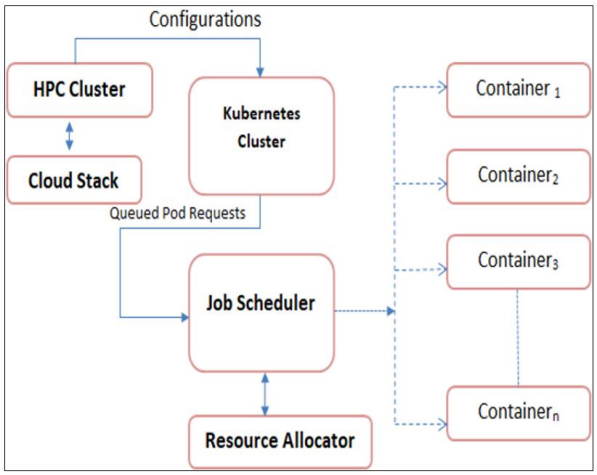

Fig. 3: Proposed model architecture in HPC cluster

Fig. 4: Kubernetes commands for container deployment and checking the label for nodes

Table 1: Comparison of scheduling layer

Scheduling layer	Mesos	Swarm	Kubernetes
Replication	Yes	Yes	Yes
Placement	Yes	Yes	Yes
Readiness	Yes	No	Yes
Rolling deployment	Yes	No	Yes
Colocation	No	No	Yes

Table 2: Comparison of the service management layer

Scheduling layer	Mesos	Swarm	Kubernetes
Namespaces	Yes	No	Yes
Labels	Yes	Yes	Yes
Readiness	Yes	No	Yes
Load balancing	Partial	No	Yes

Table 3: Computing resources specification for HPC cluster using physical servers and container-based cluster

Computing resources	HPC cluster (physical server)	Container cluster
Operating system	Ubuntu 18.04.3	Ubuntu 18.04.3
Cores	24 (Per CPU 12 cores, Per node 2 CPU)	Yes
CPU frequency	Intel ® Xeon ® CPU E5-2630 v4, 2.20 GHz	Intel i7 9xx (Core i7 IBRS) 2.79 GHz
RAM	124 GB	8 GB
Number of nodes	4 (3 compute nodes)	4 (3 worker nodes)

Test Beds Specifications

The test bed specifications are defined using Table 3 which holds the specification for HPC Cluster using both physical as well as container-based, which is described as follows:

Use Case

We have executed three HPC applications App1, App2, and App3. We have increased the workload starting from App1, then higher in App2 and highest in App3, and performed the performance evaluation based on the testbeds specification as defined in Table 3. We have used the Singularity containers (Godlove, 2019). Docker is used to provisioning the containers using Kubernetes. It provides support for both types of containers, i.e. Singularity and Docker containers. Worker nodes are not going to be linked with the HPC cluster as they will be responsible for running the Docker containers. We have used one existing model for Kubernetes to establish communication among the Kubernetes pods, which are hosted on different hosts within the cluster. Using the existing model, there is no need for an overlay network to deploy Kubernetes within the system. It internally supports its network policy to provide isolation across Kubernetes namespaces. Only two processors will be used by each application on the node within the Container cluster. It will range from 2 processors to 24 in the count based on having the testbeds specifications.

Results

All three applications were first bundled in the form of an image to deploy them in the form of containers as Kubernetes pods. We have evaluated the provisioning time to deploy the application in containers using the existing orchestrator vs our proposed model using Kubernetes. When we compared the results, we found the heavy workload-based HPC application impacts the provisioning time within the container-based cluster. We have considered the experiments using multiple applications local images vs single application Docker image.

Using Fig. 5(a) and (b), shows that Kubernetes is consuming less time among all used container orchestration tools, i.e. Apache Mesos (Saha *et al*., 2018) and Docker Swarm (Marathe *et al*., 2019). The other two orchestration tools have comparable provisioning time, i.e., almost negligible. We initially kept the replicas as two, but later we increased these from 2 to 4 and later increased gradually to 80. We found a different set of results captured using Fig. 6(a) and (b), which shows the comparison of provisioning time for the application deployment using local image vs remote Docker image. Kubernetes took lower provisioning time with a local image whereas the highest time for a remote one. It is found that a longer time is taken due to communication between Kubernetes and Docker registry. The other orchestrator found it correlated in between. Apache Mesos provides a better result. We have gradually increased the replicas count to 80 and found that the results were linearly increasing in terms of provisioning time. Fig. 7 shows that our proposed model using Kubernetes is the best fit to restore the deployed applications during the failover of containers.

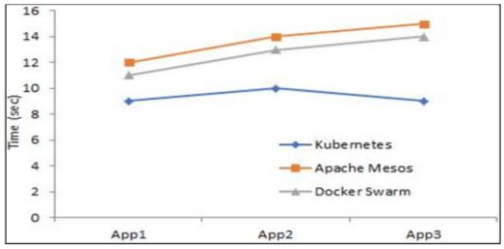

Fig. 5(a): Applications provisioning time using a local private image

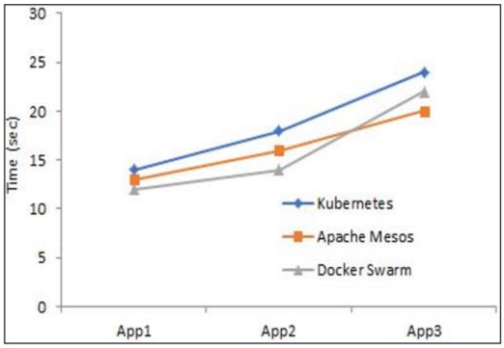

Fig. 5(b): Applications provisioning time using Docker registry remote image

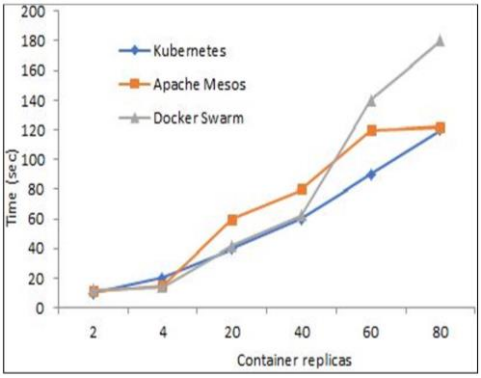

Fig. 6(a): Single Application provisioning time with multiple replicas using a local private image

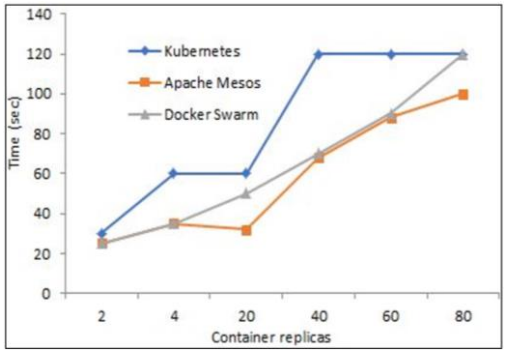

Fig. 6(b): Single Application provisioning time with multiple replicas using Docker registry remote image

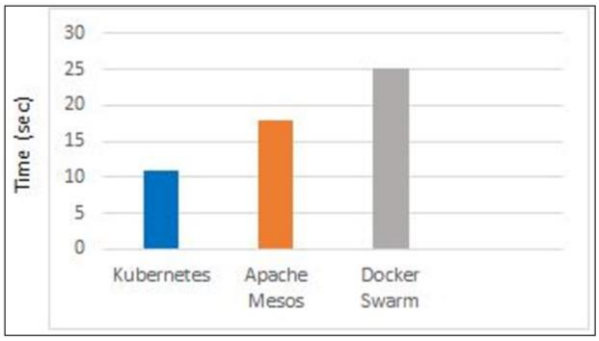

Fig. 7: Failover Time using proposed model vs existing container orchestration tools

Conclusion

The experimental results based on the defined use case highlight the advantages of our proposed model for the provisioning time to deploy the applications and the failover time. HPC-based applications tend to scale with the number of cores. We have proposed this architecture-based framework that fills the gap of HPC clusters using the containers-based cluster in the cloud environment. It will benefit where we need to handle the application with a high volume of scientific workloads. The proposed framework is scalable and flexible, as Kubernetes is handling the job submission. TORQUE has been used in the framework as a resource allocator, which takes care of the resource allocations to the containers. The higher provisioning time for remote images registered with the Docker registry using Kubernetes can be improvised by removing the communication overhead. We have proposed a framework for container orchestration to handle the scientific workloads using Kubernetes. This framework is scalable for HPC-based applications. It is the best fit for the provisioning time to deploy the application using the private local images and restore the services. It is flexible to adopt in Cloud vs non-cloud-based environment to orchestrate the containers using Kubernetes. In today's world, scientific research-based or Artificial intelligence-based applications revolve around the data (Li and Yao, 2022), and that cloud environment offers the HPC cluster using the containers. As the number of replicas grows, it needs to be handled efficiently and considered an exciting topic for future work.

Acknowledgment

I am thankful to the Koneru Lakshmaiah Education Foundation for allowing me to choose my research area as per my interest and my guide's optimistic nature, and other staff members who encouraged me to complete this research work.

Author's Contributions

Manish Kumar Abhishek: Considered the research framework, understanding of existing resource e managers, Kubernetes, container orchestration environment setup, performance result analysis, concluding framework, and preparing the manuscript.

D. Rajeswara Rao and K. Subrahmanyam: Advice on test bed specifications and participation to complete the final manuscript.

Ethics

The article is original and holds individual results for the research work. The corresponding author confirms that all of the other authors have read and approved the manuscript and that no ethical issues are involved.

References

Al Jawarneh, I. M., Bellavista, P., Bosi, F., Foschini, L., Martuscelli, G., Montanari, R., & Palopoli, A. (2019, May). Container orchestration engines: A thorough functional and performance comparison. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)* (pp. 1-6). IEEE. https://doi.org/10.1109/ICC.2019.8762053

Godlove, D. (2019). Singularity: Simple, secure containers for compute-driven workloads. In *Proceedings of the Practice and Experience in Advanced Research Computing on Rising of the Machines (learning)* (pp. 1-4). https://doi.org/10.1145/3332186

Jeffery, A., Howard, H., & Mortier, R. (2021, April). Rearchitecting Kubernetes for the edge. In *Proceedings of the 4th International Workshop on Edge Systems, Analytics, and Networking* (pp. 7-12). https://doi.org/10.1145/3434770.3459730

Julian, S., Shuey, M., & Cook, S. (2016, July). Containers in research: initial experiences with lightweight infrastructure. In *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale* (pp. 1-6). https://doi.org/10.1145/2949550.2949562

Li, Y., & Yao, D. (2022). Dynamic Analysis of Deep Integration of Artificial Intelligence Based on High-Performance Computing for Ideological and Political Teaching Evaluation. *Mobile Information Systems*, *2022*. https://doi.org/10.1155/2022/4748544

Liu, F., Keahey, K., Riteau, P., & Weissman, J. (2018, November). Dynamically negotiating capacity between on-demand and batch clusters. In *SC18: International Conference for High-Performance Computing, Networking, Storage and Analysis* (pp. 493-503). IEEE. https://doi.org/10.1109/SC.2018.00041

Rodriguez, M. A., & Buyya, R. (2019). Container-based cluster orchestration systems: A taxonomy and future directions. *Software: Practice and Experience*, *49*(5), 698-719.

Marathe, N., Gandhi, A., & Shah, J. M. (2019, April). Docker swarm and Kubernetes in a cloud computing environment. In *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)* (pp. 179-184). IEEE. https://doi.org/10.1109/ICOEI.2019.8862654

Medel, V., Tolosana-Calasanz, R., Bañares, J. Á., Arronategui, U., & Rana, O. F. (2018). Characterizing resource management performance in Kubernetes. *Computers & Electrical Engineering*, *68*, 286-297. https://doi.org/10.1016/j.compeleceng.2018.03.041

Piras, M. E., Pireddu, L., Moro, M., & Zanetti, G. (2019, June). Container orchestration on HPC clusters. In *International Conference on High-Performance Computing* (pp. 25-35). Springer, Cham. https://doi.org/10.1007/978-3-030-34356-9_3

Saha, P., Beltre, A., & Govindaraju, M. (2018, July). Exploring the fairness and resource distribution in an apache Mesos environment. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)* (pp. 434-441). IEEE. https://doi.org/10.1109/CLOUD.2018.00061

Slurm Overview, its features, and how to use it.[https://slurm.schedmd.com/overview.html]

Sultana, N., Rüfenacht, M., Skjellum, A., Bangalore, P., Laguna, I., & Mohror, K. (2021). Understanding the use of message passing interface in exascale proxy applications. *Concurrency and Computation: Practice and Experience*, *33*(14), e5901. https://doi.org/10.1002/cpe.5901

Torque Resource Manager overview and its advantages. [https://adaptivecomputing.com/cherry-services/torque-resource-manager/]

VIEWPOINT, what it is, and how to use it. [https://www.aspsys.com/solutions/software-solutions/hpc-schedulers/]

Wang, L., & Zhang, D. (2017, June). Research on OpenStack of open-source cloud computing in colleges and universities' computer rooms. In *IOP Conference Series: Earth and Environmental Science* (Vol. 69, No. 1, p. 012140). IOP Publishing. https://doi.org/10.1088/1755-1315/69/1/012140

Wrede, F., & Von Hof, V. (2017, April). Enabling efficient use of algorithmic skeletons in cloud environments: container-based virtualization for hybrid CPU-GPU execution of data-parallel skeletons. In *Proceedings of the Symposium on Applied Computing* (pp. 1593-1596). https://doi.org/10.1145/3019612. 3019894

Zitzlsberger, G., Jansík, B., & Martinovič, J. (2018). Feasibility analysis of using the maui scheduler for job simulation of large-scale pbs based clusters. *IADIS International Journal on Computer Science & Information Systems, 13*(2). https://doi.org/10.33965/ijcsis_2018130204