UNDERSTANDING THE OPPORTUNITIES OF APPLYING KUBERNETES SCHEDULING CAPABILITIES IN HIGH PERFORMANCE COMPUTING

Ioan-Mihail STAN[1], Ștefan-Dan CIOCÎRLAN[2], Răzvan RUGHINIȘ[3]

High Performance Computing and cloud computing are two paradigms of distributed systems, which until recently were based on different governance models. As the adoption of containers became an anchor between the two, the scheduling methods adopted by the industry began to inherit common elements from both. Thus, the current case study aims to analyze and classify architectural models from literature in the context of hybrid configurations. It proposes a taxonomy for the classification of HPC-Kubernetes hybrid configurations and provides improvements to the solutions identified. From the perspective of the cloud provider, Kubernetes has been selected for its versatility and for the optimized methodologies implemented for intelligent management of containerized workloads.

.

Keywords: HPC, Kubernetes, Docker

1. Introduction

High Performance Computing is one of the most prolific concepts in Information Technology and one of the drivers of innovation. From the early stages of Covid-19 pandemics, companies and institutions brought together an incredible number of resources for running studies on the new virus. The Covid19 HPC Consortium[4] shared freely around 6.4 million CPU cores, 603 Petaflops and 4.9k GPUs for running Covid-19 related projects. Still a rigid infrastructure, HPC has begun to adopt the container as a processing unit, inspired by the benefits they have brought to the cloud.

Another important game changer and innovator in the distributed systems market is Kubernetes, a container orchestrator, cloud-enabler, with a consistent adoption rate worldwide. As announced by CNCF[5] (Linux Foundation), there are

[1] PhD(c). Eng., Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: ioan.stan@upb.ro

[2] PhD(c). Eng., Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: stefan_dan.ciocirlan@upb.ro

[3] Prof., PhD., Eng., Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: razvan.rughinis@cs.pub.ro

[4] https://covid19-hpc-consortium.org/

[5] https://www.cncf.io/announcements/2022/02/10/cncf-sees-record-kubernetes-and-container-adoption-in-2021-cloud-native-survey/

currently about 5.6 million developers who have adopted Kubernetes globally. With a well-developed control plan and an ecosystem built around technology, Kubernetes outperforms other competitors and becomes an industry standard.

As both distributed systems-based platforms focus their efforts on containers, the current paper proposes a classification of HPC-Kubernetes hybrid implementations and the way in which the important capabilities of both systems have merged. This state-of-the-art article aims to analyze the implementations found in the literature, to group them and to provide punctual improvements of the proposed workflows. Section 2 presents a brief overview of some concepts and emerging technologies in the space of distributed systems. Section 3 reveals the classification taxonomy and shows improvements in current implementations and Section 4 concludes the aspects previously presented.

2. Background

Docker [1] is a containerization engine that furnishes industry with a way to pack applications and dependencies in executable artifacts able to provide the same running experience on heterogeneous systems. It aims to cover the gap caused by the lack of consistency between the development environment and production environment. As many containerization engines on the market, Docker uses two important Kernel native capabilities to run containers: namespaces and cgroup. The isolation taxonomy proposed covers six out of seven namespaces available: mnt, pid, net, uts, ipc and user and excludes more recently added time namespace.

The Docker community started the Open Container Initiative, a governance organization that provides industry with guidelines and standards with the aim of interoperability [2]. Founded in 2015, the project maintains two important specifications, one related to the containerization runtime and one related to the container image definition, both extremely relevant for interoperability. Thus, the modularity enforced brought the opportunity to replace the Docker native runc runtime with other, more secure solutions [3][4]. In addition, image build for Docker can now be easily transferred, with no extra effort, to other containerization engines implementing OCI standards [5].

For HPC however, the most common containerization engine in use is Singularity [5]. Contrary to Docker's isolation taxonomy, Singularity proposes one namespace isolation with mount (mnt) and inherits the entire user context from the underlying node. This reflects the way HPC nodes are configured as they usually unify the development and execution environments by enforcing the same access policy on both. Therefore, a container executed via an HPC task/job is running with the same user, shares the same process table, same network and so on. The only component that is detached from the underlying node is the

collection of artifacts embedded within the container image, that will no longer be required to exist on the host. Thus, this may optimize the configuration process since the toolset no longer needs to be installed before running a task/job. Singularity is OCI compliant, therefore, it can run pre-existing Docker images with no refactoring effort.

In High Performance Computing jobs or tasks are stored in batch queues and executed one by one, in the entire clustered infrastructure, imprinting a similar experience as in running them on the local development node, but with a larger spectrum of resources available. The processes triggered during the workflow are executed within the same user context, therefore untrusted users get direct access to the underlying systems. Furthermore, each system is preloaded with the entire toolkit required for running each individual job, thus one new demanded library or tool needs to be deployed on each individual node. A way to overcome this operational problem is to run tasks/jobs with containers, entities that embed the main process with all its dependencies.

Kubernetes is a container orchestrator, a powerful layer that aggregates and coordinates multiple container engines in order to fulfill complex production scenarios [6][7]. It offers capabilities like self-healing, smart release management, equitable distribution of workload on nodes, scaling and so on. Compared to HPC, Kubernetes implements another paradigm. It is service-oriented, multi-tenant and has its own dedicated resource management system and scheduling mechanism [8].

3. Taxonomy

The aim of the current paper is to classify various HPC-Kubernetes integrations in a way to better express the opportunities that may arise with such a mix of paradigms. On one side, high performance computing implementations are less flexible, whereas Kubernetes and the cloud paradigm enforces a quick and transparent response to change. Gathering attributes from both models, one classification method is focused on the volatility of the ecosystem around the HPC-Kubernetes stack. Here one can observe two patterns based on the life cycle of the hybrid setup - predefined/static topologies and on-demand topologies

In classical HPC implementations, each infrastructure change requires reloading the distributed control plane to synchronize [9] the resource pool. Thus, prior to running any task, the entire infrastructure must be deployed and rigorously configured, with all the involved nodes running indefinitely, waiting for jobs. In cloud computing each resource can be spawned up on demand. Such methods can optimize the costs of running infrastructure, as control workload doesn’t need to be present all the time. Before going further and presenting some architectures matching the classifiers proposed, both categories can also support

two main subclasses. Thus, each existing hybrid implementation can be defined as fully managed by Kubernetes or partially managed by Kubernetes. Here, the focus is on the scheduling component of the control plane and less on the data/processing plane, as the latter is a concern of the type of tasks running within the infrastructure and not a generic grouping factor. In addition, the paper assumes that batch HPC tasks can be containerized with Kubernetes compatible technologies. The containerization engine must be OCI (Open Container Initiative) compliant and to implement CRI (Container Runtime Interface) directly or via an adapter (shim[6]).

With the adoption of Operators, Kubernetes can support the extension of the API and scheduling capabilities with additional control plane logic. Thus, in theory, developing the entire HPC scheduling, as a plug-and-play module, is achievable. However, the extensive complexity of HPC systems may not be feasible to fully fit inside the Kubernetes control plane. In this regard, most case studies fall under the partially managed Kubernetes category. The closest solution to a fully Kubernetes managed workflow has been presented by M. Piras et al [10] where the HPC Grid Manager is in charge to attach and detach worker nodes to an always-on Kubernetes cluster. A HPC partition is formed on demand, by attaching required nodes to the orchestrator control plane. The job scheduling is fully outsourced to Kubernetes control plane and triggered via the native Kubernetes Deployment mechanism. However, since the Grid Manager is still detached from Kubernetes and considered also an essential part of the scheduling mechanism, the solution is as well labeled as partially managed by the orchestrator. Furthermore, a Deployment object stays indefinitely in Kubernetes, thus, an external mechanism must monitor pods toward completion and delete the Deployment during the tear-down phase. For this case study, adopting batch jobs Kubernetes objects instead of deployments can improve the control routines.

The infeasibility of fully integrating HPC control processes into Kubernetes stems from the complexity of the input information the scheduler receives. As C. Iacopo et al describe in [11], organizing tasks on the appropriate infrastructure nodes is more than a matter of resources available and load. In HPC, data localization is an important input parameter to any scheduling heuristic, especially in cases where the jobs/tasks are data-intensive or require ingesting extremely large datasets. Movement of data may not be always feasible, thus, running a job in the proximity of data required can also be weighted for scheduling decisions.

[6] https://kubernetes.io/docs/tasks/administer-cluster/migrating-from-dockershim/check-if-dockershim-deprecation-affects-you/

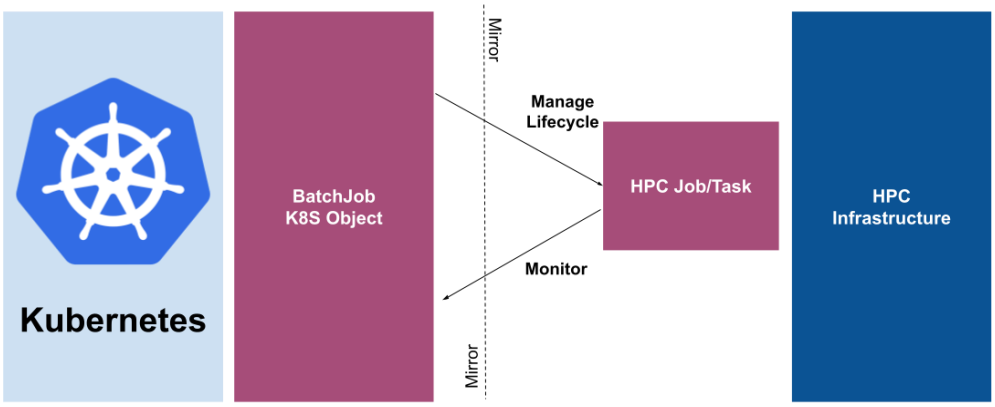

Fig 1. HPC-connector concept architecture

Furthermore, many hybrids HPC-Kubernetes implementations follow the STSC or MTSC model, where one/many task objects trigger one container. STSC (Single Task Single Container) and MTSC (Multiple Tasks One Container) are adaptations of the Flynn taxonomy (as defined in [11]) and describe how HPC-Kubernetes hybrid objects must enforce, isolate and manage the execution of HPC tasks/jobs. S. Lopez-Huguet et al. [12] follows the STSC/MTSC model where the hpc-connector (Fig. 1) monitors the lifecycle of a Kubernetes job and replicates the same behavior within the HPC infrastructure. Therefore, the software running inside the container can interpret Operating System signals, initiated by the orchestrator control plane and apply the same directives to the HPC scheduler. One possible problem with such an approach is when the underlying containerization engine or Kubernetes decide to instantly kill a pod due to lack of memory available or if the pod bypasses the memory limits enforced - OOMKill. Thus, the software may not be able to further capture signals and therefore any link between a cluster job and HPC job may be lost or interrupted.

As described by [11], the most challenging part in an HPC-Kubernetes ecosystem is to manage STMC (Single Task Multiple Containers) and MTMC (Multiple Tasks Multiple Containers) workloads. For STMC, Kubernetes natively supports parallelism for its batch job objects, where multiple instances of the same software solution run simultaneously towards completion. Similar behavior can be achieved by deploying the kube-batch[7] [10] controller and running kube-batch specific objects. The coordination on the workload consumption must be made internally, within the distributed containerized application. The MTMC paradigm requires more complex scheduling and isolation mechanisms which can be also fulfilled by current Kubernetes capabilities. However, such deployments may

[7] https://github.com/kubernetes-sigs/kube-batch/blob/master/README.md

require multiple Kubernetes objects. Based on the use case, one may need to enforce complex network policies to isolate communication topology or deploy Kubernetes services to ensure inter-pod communication. In some cases, the architecture may require intermediary solutions like Message Queues or Caching systems in order to ensure asynchronous data consumption[8]. Another assumption that must be made in order to fully align with the taxonomy proposed by case study [11] is that a Kubernetes pod must contain only one application container, whereas extra containers are only supplements following one of the industry specific design patterns: Sidecar, Adaptor or Ambassador [13].

Related to data locality, there are situations where data transfer from one location to a remote one is mandatory. As proposed by C. Iacopo et al and described by case study [11], the workload is distributed among both Kubernetes and HPC jobs, where the consumers are hosted by Kubernetes pods, while producers are kept in HPC infrastructure. Similarly, I. Stan et. al. [14] enforces a tailored system where data processed by the ATLAS Experiment at CERN HPC infrastructure for specific Machine Learning jobs is further transferred and visualized in Kubernetes, on a demand basis. In both use cases, based on the amount of data that must be transferred, the underlying infrastructure must be also prepared to achieve quick transfer rates. On one side, as proposed by A.M. Beltre et. al. [15] using InfiniBand where possible, increases the overall performance and reduces the execution times of jobs/tasks. On the other hand, the container technology has also a significant stake in the overall performance. In most cases, Singularity behaves better and, in addition, enforces a corresponding security policy [15][16] for the execution of HPC jobs/tasks.

Based on previous use cases, there is another classification that can be applied on hybrid HPC-Kubernetes solutions regarding the precedence of the two main components in the job execution pipeline. In the case studies [10][11], the HPC components precede and trigger Kubernetes data flow, while in cases like [12] and [17][18], the pipeline is initiated in Kubernetes and continued by the HPC scheduling mechanism. N. Zhou et. al. presents in [17][18] an alternative of using the scheduling mechanism of Kubernetes to determine which HPC partition/queue fits the current job/task demand. The implementation uses the concept of Virtual Kubernetes Node (virtual kubelets), where each abstracted node hides an HPC partition/queue and reports the set of resources available to the Kubernetes control plane. The Kubernetes master node analyzes the number of resources available and schedules the workload to the appropriate partition/queue. Since the HPC infrastructure is heterogeneous and sometime segregated per category of tasks/jobs [19], the scheduling mechanism can make use of Node Affinity, Taints and Tolerations, or Node selection functions to properly select the HPC partition/queue in charge to run the current job/task archetype. Thus, as

[8] https://kubernetes.io/docs/tasks/job/fine-parallel-processing-work-queue/

presented by V. Dakic et al [19], a segregation model can be based on the underlying computing architecture. Such classification methodology can be supported and implemented in Kubernetes through a mix of node labeling and node selector or affinity.

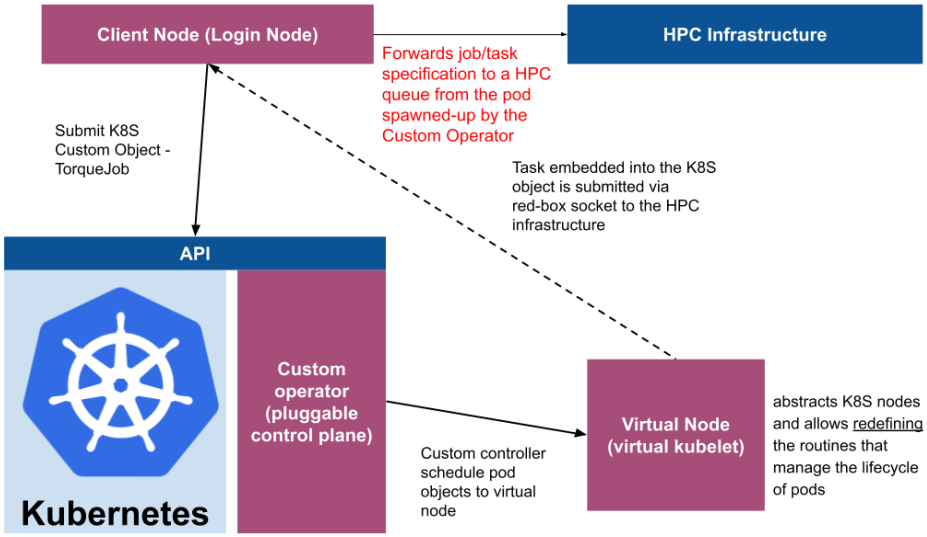

Fig 2. Torque-operator concept architecture

As originally announced, all cases studied can be grouped in two main categories: static/dedicated topologies vs on-demand topologies. Part of them inherit the dynamics of cloud topologies where resources are no longer dedicated to a specific workflow but shared in order to leverage the cost and to balance the utilization rates. Here the discussion can be further extended as the paradigm itself has multiple valences. In case study [14], pods are spawned-up, on request, from a service catalog, in order to expose data processed by the HPC infrastructure. In case study [11] StreamFlow covers a more extensive use case where the solution supports hybrid workflows, implements various connectors towards multiple processing systems including Occam HPC setup and Kubernetes and ensures data transfer among technologies. Therefore, if required, StreamFlow connects to a multi-tenant Kubernetes cluster and executes specific jobs, based on the demand. The Kubernetes cluster can be further utilized for other purposes and workflows. Case study [9] proposes a different approach where nodes are attached, on demand, to a Kubernetes control plane and removed after the execution of the task/job. Depending on the number of nodes available another method to achieve similar behavior would be to attach all the HPC nodes to the control plane and to cordon them if not required for the current execution. Currently the Kubernetes control plane supports up to 5000 nodes attached. With the Tolerations and the

Affinity mechanisms, the attach/detach mechanism can be avoided also, and therefore, exclude the need to outsource such function to a Grid Management System.

Solutions like the one proposed by N. Zhou et al. [17][18] (Fig 2.) goes under the predefined-static topology classifier as it requires a precise structure. The system includes a bridge/login node settled between the two separated infrastructures (HPC and Kubernetes cloud) and various virtual kubelet nodes that rely on the HPC manager API to report available resources. The most interesting approach for the on-demand paradigm consists of the case study [9] where C. Cerin et al not only borrow the execution capacity from a Kubernetes cluster but also maintain the HPC control plane/manager in pods on top of an existing on-premises cloud implementation. As an improvement to the proposed architecture, cluster node preparation for HPC workload can be done via Daemon Set objects with escalated privileges and filtered via Tolerations. Thus, a privileged Daemon Set pod can prepare the underlying node and attach it to the containerized control plane once deployed in Kubernetes.

6. Conclusions

The case study aimed to define a classification model for Kubernetes implementations in High Performance Computing, bringing some specific improvements for those found in literature. The classification methodology comprises a simplified structure on three layers that aims to cover most implementations and to provide a better visibility of the integration methods used - Fig 3.

The three layers taxonomy – fully-managed by Kubernetes, Kubernetes to HPC and HPC to Kubernetes - that define the form of communication between the control planes of the two distributed infrastructures is doubled by another dimensional axis, which represents the volatility of the hosting infrastructure – fixed/pre-defined and provisioned on a demand basis.

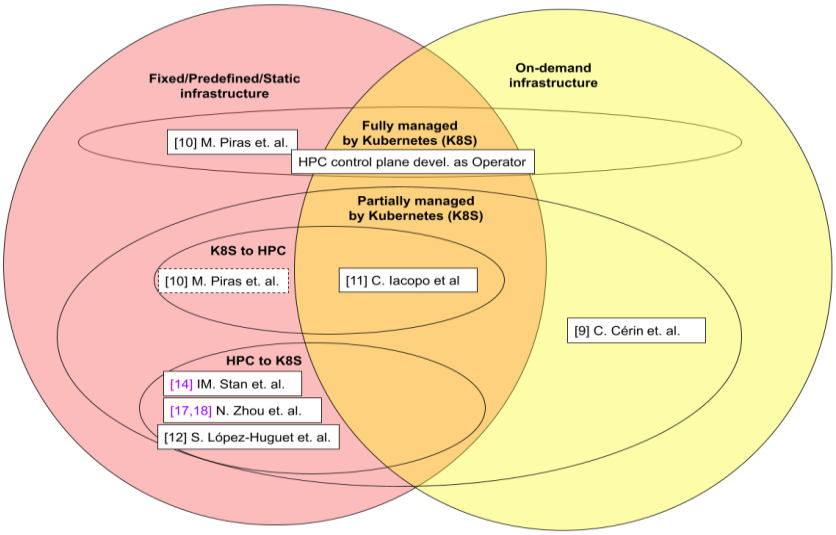

Fig 3. Hybrid HPC-Kubernetes system classification model

This observation perspective nuances the opportunities for cost optimization in terms of resource consumption.

The improvements suggested during the solutions analysis propose small but impactful changes in the architecture and design of the solutions found, especially in terms of replacing specific Kubernetes objects in certain use cases for optimizing the current workflows.

REFERENCES

[1]. *B. B. Rad, H. J. Bhatti, and M. Ahmadi*, An introduction to docker and analysis of its performance., International Journal of Computer Science and Network Security (IJCSNS) 17.3, 2017 pp. 228.

[2]. *L. Benedicic, et al.,* Sarus: Highly scalable Docker containers for HPC systems., International Conference on High Performance Computing. Springer, Cham, 2019.

[3]. *X. Wang, J. Du, and H. Liu,* Performance and isolation analysis of RunC, gVisor and Kata Containers runtimes., Cluster Computing, 2022 pp. 1-17.

[4]. *I. Mavridis, and H. Karatza*., Orchestrated sandboxed containers, unikernels, and virtual machines for isolation-enhanced multitenant workloads and serverless computing in cloud. Concurrency and Computation: Practice and Experience, 2021 e6365.

[5]. *G. M. Kurtzer, V. Sochat, and M. W. Bauer.,* Singularity: Scientific containers for mobility of compute., PloS one 12.5, 2017 e0177459.

[6]. *F. H. B. Megino, et al.,* Using Kubernetes as an ATLAS computing site., EPJ Web of Conferences. Vol. 245. EDP Sciences, 2020.

[7]. *A. P. Ferreira, and R. Sinnott.,* A performance evaluation of containers running on managed kubernetes services., 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, 2019.

[8]. *Z. Wei-guo, M. Xi-lin, and Z. Jin-zhong.*, Research on Kubernetes' Resource Scheduling Scheme., Proceedings of the 8th International Conference on Communication and Network Security., 2018.
[9]. *C. Cérin, N. Greneche, and T. Menouer*. Towards pervasive containerization of HPC job schedulers., 2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). IEEE, 2020.
[10]. *M. E. Piras, et al.,* Container orchestration on HPC clusters., International Conference on High Performance Computing. Springer, Cham, 2019.
[11]. *I. Colonnelli, et al.,* StreamFlow: cross-breeding cloud with HPC., IEEE Transactions on Emerging Topics in Computing 9.4, 2020 pp.1723-1737.
[12]. *S. López-Huguet, et al.,* Seamlessly managing HPC workloads through Kubernetes., International Conference on High Performance Computing. Springer, Cham, 2020.
[13]. *B. Burns, and D. Oppenheimer.,* Design patterns for container-based distributed systems., 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)., 2016.
[14]. *I. Stan, S. Padolski, and C. J. Lee,* Exploring the self-service model to visualize the results of the ATLAS Machine Learning analysis jobs in BigPanDA with Openshift OKD3., EPJ Web of Conferences. Vol. 251. EDP Sciences, 2021.
[15]. *A. M. Beltre, et al.,* Enabling HPC workloads on cloud infrastructure using Kubernetes container orchestration mechanisms., 2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC). IEEE, 2019.
[16]. *I. Stan, D. Rosner, and Ş. Ciocîrlan.*, Enforce a global security policy for user access to clustered container systems via user namespace sharing., 2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet). IEEE, 2020.
[17]. *N. Zhou, et al*., Container orchestration on HPC systems., 2020 IEEE 13th International Conference on Cloud Computing (CLOUD). IEEE, 2020.
[18]. *N. Zhou, et al.,* Container orchestration on HPC systems through Kubernetes., Journal of Cloud Computing 10.1, 2021 pp. 1-14.
[19]. *V. Dakic, M. Kovac, and J. Redzepagic.,* Optimizing Kubernetes performance, efficiency and energy footprint in heterogenous HPC environments, Annals of DAAAM & Proceedings 10.2, 2021.