Running Kubernetes Workloads on Rootless HPC Systems using Slurm

Jonathan Decker, Sören Metje and Julian Kunkel
Institute for Computer Science, Universität Göttingen,
Goldschmidtstraße 7, 37077 Göttingen, Germany
e-mail: jonathan.decker@uni-goettingen.de | soerenmetje@yahoo.de | julian.kunkel@gwdg.de

***Abstract*—Kubernetes has become a widespread orchestrator for cloud workloads but with increasing demand for compute the need arises to also access HPC environments that are operated via batch schedulers such as Slurm. A number of solutions for combining Slurm and Kubernetes are available, which can be categorized further based on the interaction between Slurm and Kubernetes that they provide. In this paper, we consider the use case of utilizing an existing Slurm cluster to run Kubernetes workloads. For this, we introduce a new solution called Kind Slurm Integration (KSI) based on Kind and rootless Podman and compare it based on performance, usability and maintainability to the existing solutions Bridge Operator and High-Performance Kubernetes (HPK). We found that Bridge Operator provides native performance as it effectively submits Slurm jobs through a Kubernetes interface and that HPK provides good performance by creating almost feature complete Kubernetes clusters on top of Apptainer. KSI on the other hand is able to provide fully functional Kubernetes clusters inside Slurm jobs but lacks behind in network performance. Overall, we conclude that more work is needed to run Kubernetes workloads under Slurm without missing out on features or performance.**

***Keywords*-Kubernetes; HPC; Container; Slurm; Cloud.**

I. Introduction

Kubernetes has established itself as a widespread solution for orchestration of cloud workloads [1][2] and is used for various workloads including service computing, running large amounts of micro services, as well as batch jobs, such as data analytics or machine learning. However, batch jobs would fit better into HPC environments where powerful high-performance compute and networking resources are available. HPC workloads are commonly scheduled using a batch scheduler such as Slurm [3] but Kubernetes itself can also be used for scheduling HPC jobs using a batch scheduler such as Volcano [4] and have already been scaled to large clusters using appropriate workarounds [5]. Nevertheless, while Kubernetes brings a large array of features, its virtualization layers incur a performance overhead compared to bare metal performance [6], which one could achieve with Slurm.

Users might want to bring their Kubernetes workloads into Slurm-based HPC environments to benefit from the reduced overhead compared to a regular Kubernetes cluster or to gain access to additional compute hardware, which could also include specialized hardware only available in HPC environments. Rewriting Kubernetes workloads to be executable in Slurm may require significant effort and expertise with the scheduling systems. However, various approaches and implementations exist for combining Slurm and Kubernetes enabling users to dynamically move workloads between cloud and HPC environments.

As there have been various efforts to combine Kubernetes and Slurm, we consider the definition by Wickberg of Schedmd [7] who defines four categories from the perspective of Slurm for such approaches.

- **Over**: The entire Kubernetes environment exists within a Slurm job and is therefore temporary as it is fully removed once the job completes.
- **Distant**: Compute nodes are part of either a Kubernetes or a Slurm cluster and may be moved between the clusters.
- **Adjacent**: Slurm and Kubernetes utilize some form of plugins or bridging tools to cooperate but can still be used individually.
- **Under**: Kubernetes runs a Slurm cluster within its own environment across one or more pods.

Given the above use case of running Kubernetes jobs in an existing Slurm environment, this fits the **Over** or **Adjacent** model. After investigating existing solutions that implement either of these models we found various approaches that provide the **Adjacent** model but no system for having a Kubernetes cluster running within a Slurm job as described in the **Over** model. Therefore, we present **Kind Slurm Integration (KSI)** [8], an implementation of the **Over** model based on Kubernetes in Docker (Kind) [9]. We systematically evaluate and compare KSI to existing solutions including **Bridge Operator** by IBM [10], **WLM-Operator** by Sylabs [11], **kube-slurm** by Kalen Peterson [12] and **High-Performance Kubernetes (HPK)** [13].

Our evaluation consists of a review of the state of the respective projects with regard to features and maintainability as well as a performance analysis to determine the overhead incurred by the respective approach. For this purpose, we benchmarked the solutions based on workload startup time, CPU compute performance, memory throughput, storage throughput, network latency and network throughput, and compared the results to bare metal. We found that not all of the implementations listed above were able to pass a minimal functionality test. For those that passed, no significant differences in CPU compute performance, memory throughput and storage throughput were found. While our own solution, KSI, is outperformed by the others in terms of startup time and network performance, it still provides the most complete support for Kubernetes features compared to the others.

Overall this paper contributes a systematic evaluation of existing approaches that implement the **Adjacent** or **Over** model to combine Slurm and Kubernetes, the design and implementation of a proof-of-concept for KSI and a final

overview of the features and limitations of the evaluated approaches. The work shown in this paper is based on the master's thesis of one of the authors [14].

The remainder of the paper is organized as follows: In Section II the various implementations for integrating Slurm and Kubernetes are discussed. The methods for benchmarking and comparing the solutions as well as the design of KSI are discussed in Section III. The results of the evaluation are given in Section IV. Finally, Section V provides the conclusion and outlook for future work.

II. Related work

To properly distinguish various approaches for combining Slurm and Kubernetes, we discuss the four models along with notable examples. We also cover related approaches that do not use either Slurm or Kubernetes and then have a more in-depth look at the implementations, which we evaluated in this paper.

A. Models for Integrating Slurm and Kubernetes

The four categories for combining Slurm and Kubernetes defined by Wickberg of Schedmd [7] are **Over**, **Distant**, **Adjacent** and **Under** as defined in Section I.

a) ***Distant*** *model:* Notable implementations include [15] and [16], which both implement systems for dynamically changing the partitioning of a node pool between a Kubernetes and Slurm cluster.

b) ***Under*** *model:* Contributions have been made in [17], [18] and [19], in which Slurm is being run as a set of Kubernetes pods. A significant project in this category is Slinky [20] by Schedmd who had created a Slurm Kubernetes bridge implementation as a proof-of-concept before creating Slinky. The proof-of-concept implementation followed the **Adjacent** model, was not functional and has since then been removed from public access.

c) ***Adjacent*** *model:* Approaches in this category are relatively diverse in their approaches including Bridge Operator [10], WLM-Operator [11] and HPK [13]. Each of these approaches is discussed in more detail in Subsection II-C.

d) ***Over*** *model:* There are no notable implementations of this model except for KSI [8], which is presented in detail in Subsection III-B.

B. Other Approaches for Integrating HPC and Cloud

While this work focuses on combining Slurm and Kubernetes it should be noted that there are alternative approaches to running HPC workloads through a cloud interface. For example, as mentioned in Section I, Volcano [4] is an extension for the Kubernetes scheduler, which implements features such as batch and gang scheduling. This enables the execution of batch workloads as shown in [21][22].

Another notable approach is **hpc-connector** [23] presented in [16], which enables the submission of jobs through an arbitrary cloud interface to be executed via Slurm. This approach can be considered similar to Bridge Operator but is not bound to Kubernetes but also lacks deeper integration with any specific cloud platform to enable advanced features.

Finally, there is [24] who integrated TORQUE [25] with Kubernetes, enabling scheduling of HPC workloads through Kubernetes to TORQUE similar to the **Adjacent** model.

C. Implementations for *Adjacent* Slurm and Kubernetes

1) WLM-Operator: Sylabs Inc. had developed the WLM-Operator [26] and Singularity-CRI [27] with Singularity-CRI providing a Kubernetes-compatible implementation of the Container Runtime Interface for Singularity [11]. The WLM-Operator implements a Kubernetes operator that is able to interface with Slurm such that Slurm nodes become visible in Kubernetes as virtual nodes.

Moreover, it provides a Custom Resource Definition (CRD) in Kubernetes called *SlurmJob*, which enables the submission of Slurm jobs through Kubernetes. When submitting a SlurmJob, a dummy pod is created in Kubernetes and the actual job is submitted to Slurm to be run in a Singularity container. The results are then collected through another pod via a shared storage before closing the dummy pod once the job completes.

However, on December 30th 2020, both WLM-Operator and Singularity-CRI projects have been archived with no further development planned.

2) Bridge Operator: IBM had developed Bridge Operator [28] in order for a Kubernetes cluster to be able to access external compute resources including Slurm clusters [10]. Bridge Operator implements a Kubernetes operator and provides the *BridgeJob* CRD, which accepts all the details required to launch a Slurm job including the remote URL of a Slurm cluster, what resources to request and a remote storage configuration.

For each BridgeJob, the Bridge Operator starts a monitoring pod and submits the job to Slurm. The monitoring pod regularly updates a Kubernetes ConfigMap with the current status and fetches the job output. The creators of Bridge Operator have also demonstrated how to run Kubeflow workloads through BridgeJobs [29], however, as these jobs are converted to Slurm jobs, the Kubernetes pods are not directly being run in Slurm.

3) HPK: HPK [30] is presented in [13] and [31] as a way to run Kubernetes workloads on Slurm through Apptainer [32]. It is deployed as a single Apptainer container that runs the Kubernetes control plane and a custom implementation of virtual Kubelet [33], which presents an entire Slurm cluster as a single node in the cluster. Whenever a new pod is to be scheduled, it submits a job through Slurm for the pod to be started as a container using Apptainer.

For the container networking to function, it relies on Flanneld service [34] to be installed on the nodes and the Flannel-CNI plugin [35] to be installed for Apptainer. However, only headless services without cluster IPs are supported as the additional layer of load balancing is not possible with the used networking stack. Moreover, the command `kubectl exec`, which is used to execute commands inside Kubernetes pods, is not supported.

4) Kube-Slurm: The kube-slurm project [12] provides a tool for controlling Kubernetes resources using Slurm jobs. When deploying, Slurm and Kubernetes must both be installed on the same set of nodes with kubectl available on all nodes. Once

deployed, users can submit Slurm jobs, which get scheduled by the tool as Kubernetes pods onto the nodes selected by the Slurm scheduler.

The deployment can also be completed with the **Under** model by having Slurm run within Kubernetes but still using Slurm to schedule the pods. Nevertheless, due to the way the access is provided to the Slurm scheduler, all users receive the same access to the Kubernetes cluster making this approach unfit for multi-user setups with potentially malicious users.

III. Methodology

This work focuses on approaches for combining Kubernetes and Slurm that allow running workloads on an existing Slurm cluster following the **Over** or **Adjacent** model and investigates the suitability of the existing solutions. For that purpose we define the following research questions:

RQ1 Can workloads be submitted using Kubernetes tooling, e.g., `kubectl`?
RQ2 Can workloads be scheduled and executed on machines managed by an existing Slurm cluster without root access?
RQ3 Can workloads be executed across multiple machines in parallel?
RQ4 What is the performance overhead imposed by the tool?
RQ5 Is the tool easy to operate for the end user?
RQ6 Is the tool well maintained?

RQ1, **RQ2** and **RQ3** define the functional requirements. For a solution to be a valid approach for utilizing a Slurm cluster through Kubernetes, it should answer yes to at least **RQ1** and **RQ2** with a yes to **RQ3** being desirable but not strictly required. Notably, these requirements do not include whether a solution must be able to run Kubernetes workloads or if it may run Slurm workloads through a Kubernetes interface. For example, Bridge Operator accepts Slurm workloads submitted through Kubernetes while HPK takes Kubernetes workloads submitted through a Kubernetes interface, with both executing the workloads on a Slurm cluster. Moreover, the distinction between the **Over** and **Adjacent** model breaks down to whether the deployment requires an existing component, such as a Kubernetes cluster, to be running before the Slurm job that will handle the target workload is submitted.

RQ4 is concerned with the performance cost of a given solution. Depending on the architecture and optimization a given implementation may cost additional compute power or delay the start of workloads, which should be minimal for an application to fully harvest the power of HPC machines.

RQ5 and **RQ6** cover the usability and maintainability of a given software providing an indication for the viability in productive use.

While **RQ1**, **RQ2** and **RQ3** can be answered as yes or no questions, **RQ4**, **RQ5** and **RQ6** require a graded answer. We use a three point scoring from + (positive) over ○ (average) to − (negative) to be able to quickly compare the results for multiple implementations. + is the best score, which is given if the implementation fulfills the requirements without any significant drawbacks. ○ is the middle score, which indicates that some limitations apply and − is the lowest score, which applies if significant shortcomings exist.

A. Selection of Implementations to Evaluate

In Section II-C we had introduced the WLM-Operator, Bridge Operator, HPK and Kube-Slurm. Before starting our evaluation we performed a minimal functionality test and found that the latest version of WLM-Operator is no longer functional on recent operating systems. Despite our best efforts and reaching out to Sylabs, we were unable to reproduce the minimal examples in the repository. Therefore, WLM-Operator can be considered retired and we will not further consider it.

Kube-Slurm requires the installation of a Kubernetes cluster on all nodes as part of its deployment, which violates **RQ2** that it must be able to operate without root access. Therefore, we will not further consider Kube-Slurm.

This only leaves HPK and Bridge Operator as viable targets for further evaluation along with KSI, which is introduced in the next section. However, when testing Bridge Operator we ran into a number of issues, which we reported on Github and created a pull request [36] with our code adjustments.

B. Kind Slurm Integration (KSI) Design

Our main objectives of designing another approach for combining Slurm and Kubernetes were that it should follow the **Over** model and support all Kubernetes features. Following the **Over** model, KSI can be run strictly inside Slurm jobs without relying on external components. This was important to us, as our use case involved a multi-user HPC system in which the users of KSI would not be able to deploy a control plane outside of Slurm jobs as it is required by HPK. Moreover, as we could not find any existing projects employing the **Over** model, we consider this a research gap.

We utilized rootless Kind [9] via its experimental Podman support to create a script that receives a Kubernetes workload, initializes a cluster inside a Slurm job, executes the workload and then closes the cluster as the Slurm job ends. Before settling on rootless Kind, we also considered Minikube [37], K3D [38] and Usernetes [39] but found rootless Kind to be the most suitable.

Kind [40] was developed with local development and automatic testing of Kubernetes in mind. It can deploy a fully functional Kubernetes cluster on a single node by deploying a "node" image, which internally runs another container runtime from which all containers belonging to the cluster are run. From the perspective of the host system, only a single container is running for the control plane node of the cluster. Moreover, by deploying multiple "node" images on the same host, Kind can simulate a multi-node cluster.

For operating KSI inside of Slurm jobs without access to root permissions, which are required for regular container operation, we employ rootless Kind [9]. Rootless Kind relies on a container runtime, in our case Podman, which uses Cgroups v2 features to run rootless containers. Besides Cgroups v2, Podman also relies on the shadow-utils package, which provides subuids and subgids for user namespaces.

However, Kind itself is not designed for multi-node clusters across multiple physical machines or VMs, so in order to achieve **RQ3**, we would require a tool such as Kilo [41] or Liqo [42]. With these it is possible to aggregate multiple Kubernetes cluster into a single cluster by representing each cluster as a virtual Kubelet in the main cluster. With this KSI could be deployed across a number of nodes in a multi-node Slurm job and all the worker nodes would use Kilo or Liqo to register with the cluster on the main node, which in turn would be able to schedule work across all nodes. When using this approach, each node has to run KSI and initialize its own Kubernetes cluster, which includes running all control plane components, before joining together via Kilo or Liqo to form a single cluster. We have not implemented this feature for the version of KSI under evaluation in this paper, but expect that the same overhead, in terms of CPU and memory consumption by the control plane components that applies to a single node running KSI, would then also apply to each individual node in such a multi-node setup.

In order to deploy KSI, the nodes must provide a recent Linux operating system with support for Cgroups v2, rootless Podman must be set up, as well as slirp4netns [43], to provide networking for rootless Podman. Since the experiments for this work have concluded, Podman 5.0 [44] was released, which uses pasta [45] as its default rootless network driver instead of slirp4netns. Workloads can be configured and embedded via `run-workload.sh`, which sets up the Kubernetes cluster when submitted via Slurm. `srun -N1 /bin/bash run-workload.sh example-workload.sh` would set up a single node cluster and then run the workload described in `example-workload.sh`. The workload script should internally use kubectl to create the required Kubernetes resources and then wait for the workload to finish. Upon completion of the workload script, the cluster is stopped and removed such that the Slurm job is also closed.

The produced KSI code, documentation and workload examples are released under the GPL-3.0 license on Github [8].

C. Performance Evaluation

To assess the performance overhead to answer **RQ4** we have broken down our benchmarking into the following factors:

- Startup time: Measured with a dummy workload
- CPU compute performance: Measured with Sysbench [46]
- Memory throughput: Measured with Stream [47]
- Storage throughput: Measured with Fio [48]
- Network latency: Measured with Netperf [49]
- Network bandwidth: Measured with iPerf3 [50]

We consider these as representative factors for user workloads that might be run through any of the tools under study.

As the bare metal baseline we run the benchmarks through Slurm without Kubernetes. All benchmarks were run on two machines with hardware specifications as shown in Table I. On the nodes we used software versions as shown in II. The Kubernetes version v1.27.3 is the most recent version at the time of the experiments and was used for the external cluster for the Bridge Operator as well as by KSI. HPK, however, is pinned to v1.25.0 in its code base. Furthermore, we disabled SELinux, swap and write caching to more clearly measure the respective factors.

TABLE I. HARDWARE SPECIFICATIONS OF THE BENCHMARK MACHINES.

CPU	Intel(R) Xeon(R) CPU E5-2695 v3
CPU Sockets	2
Cores per socket	14
Threads per core	2
Total threads	56
RAM	24 DIMMs DDR4 16 GB 1866 MHz
Total RAM	384.00 GB
Storage	1 Verbatim Vi550 S3 SATA Revision 3.2 SSD
Total storage	128.00 GB
Network interface	QLogic BRCM 10G/GbE 2+2P 57800-t rNDC

TABLE II. SOFTWARE VERSIONS OF THE BENCHMARK MACHINES.

Linux OS	CentOS Stream 9
Slurm	23.02.5
Podman	4.6.1
slirp4netns	1.2.2-1
Kind	0.20.0
Kubectl	v1.28.2
Kubernetes	v1.27.3
HPK Kubernetes	v1.25.0
shadow-utils	2:4.9-8

D. Project State Evaluation

Evaluating the maintainability and usability of software has been studied extensively [51][52] with many tools and methods having been proposed. For this work, in order to answer **RQ5** and **RQ6**, we have to consider what methods to employ.

In order to grade usability we consider the state of the available documentation as well as the difficulty of setting up and operating the respective tools for an assumed non-expert user based on our own experience of working with the tools during this study. For grading maintainability we reviewed the state of the code bases based on its complexity, whether it has been kept up-to-date and how well issues are being addresses. Moreover, we consider that a code base that does a comparatively simple job while relying on more well maintained dependencies is itself more maintainable than a larger code that has more moving parts that may require maintenance.

We acknowledge that more sophisticated methods are available but consider our approach sufficient to compare three projects on a three point grading schema.

IV. RESULTS

The commit hashes of the implementation versions used in our tests are as follows:

- Bridge Operator: `56334fa57caf2de28df6ff76df8a6e6232021421`
- HPK: `a902acbf2436e8a85a4620fddfa5745523f443d4`
- KSI: `780ef3a0562ad4bb12611f9ef43fa743fe0277d0`

A. Functional Requirements

For all Bridge Operator, HPK and KSI, the answer to **RQ1**, **RQ2** and **RQ3** is yes with the exception that KSI requires

an additional integration with Kilo, Liqo or a similar tool to support multi-node execution, which has not been implemented yet.

B. Project State

a) Bridge Operator: When submitting a workload through Bridge Operator, it requires the user to create an instance of the CRD BridgeJob. With that no understanding of Slurm by the user is required. However, the available documentation for Bridge Operator is limited with some examples not working such that a patch was required to make it work [36]. While the project itself depends only on the Slurm REST API giving it a stable foundation, the project itself seems abandoned with no activity after late 2022. Due to this, we rate it ○ in both usability and maintainability. We would have rated + for usability if the examples were all functional and for maintainability if the project was actively being maintained.

b) HPK: Similar to Bridge Operator, HPK can be controlled directly through `kubectl` without additional understanding of Slurm by the user. Nevertheless, while its documentation is also limited, after we had completed our experiments [30] was released along with `v0.1.2` of HPK containing a number of bug fixes. This shows that the project is being actively developed, moreover, when we ran into issues, we quickly received community support from the maintainers. With this we rate the maintainability as + and the usability as ○ because in addition to the points mentioned above, HPK does not support certain Kubernetes features, most notably `kubectl exec` and services, such that users need to work around these limitations.

c) KSI: Unlike the other two tools, KSI is started via Slurm as it has no active component outside of the Slurm job. Its usage is documented with several examples and it depends on Podman and Kind, which are both well maintained projects. The project delivers a feature complete Kubernetes cluster via a set of scripts making it both usable and maintainable so we rate KSI + for both factors. However, as KSI is our own creation, we cannot claim that this evaluation is unbiased and should be regarded as such.

C. Performance

The benchmarking scripts, as well as the raw test data, are available online [53]. Each benchmark was repeated 10 times to minimize random error with the standard deviation shown in the graphs.

TABLE III. WORKLOAD STARTUP TIME, LOWER IS BETTER.

Integration Approach	Startup Time [s]
bare metal	0.141
Bridge Operator	2.725
HPK	2.497
KSI	53.921

1) Startup Time: The startup delays given in Table III have negligible standard deviation and show that Bridge Operator and HPK start a workload in 2 to 3 seconds while KSI requires almost one minute. This result is as expected since Bridge Operator and HPK already have an active Kubernetes cluster running before they submit their Slurm job while KSI has to set up a Kubernetes cluster from scratch. Considering that HPC workloads often run for multiple hours, one minute extra start up time is not great but acceptable. Due to this we rate Bridge Operator and HPK with + and KSI with ○.

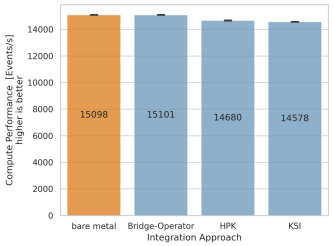

Figure 1. CPU compute performance results using Sysbench.

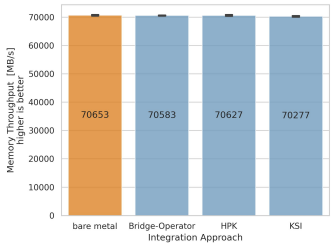

Figure 2. Memory throughput results using Stream.

2) Compute Performance: Figure 1 shows that Bridge Operator and bare metal are effectively on the same performance level while HPK and KSI are slightly lower than bare metal (2.7% and 3.4%, respectively). The difference arises due to the virtualization overhead and additional active components running for HPK and KSI but is overall negligible such that we rate all approaches with +.

3) Memory Performance: Figure 2 shows similar to Figure 1 only minor differences for HPK and KSI due to the additional active components and virtualization such that we also rate all with + here.

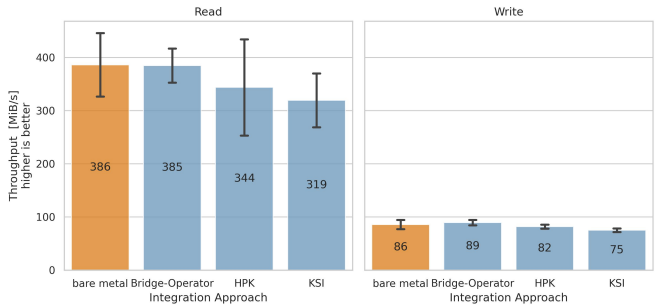

Figure 3. Storage throughput results using Fio and sequential operations.

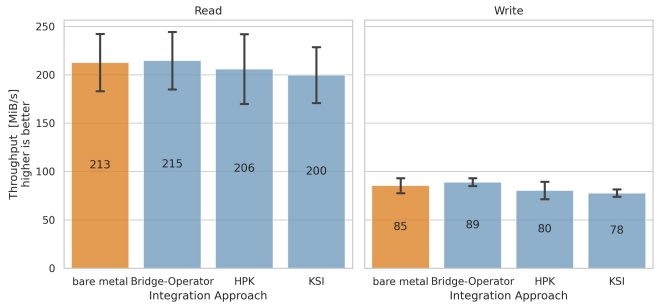

Figure 4. Storage throughput results using Fio and randomized operations.

4) Storage Performance: The sequential read and write shown in Figure 3 shows a similar pattern as the random read and write shown in Figure 4 with Bridge Operator being on the same level as bare metal and HPK and KSI lacking behind. More specifically HPK is about 11% slower in sequential and 5% slower in random reading and KSI is overall 17% slower in reading and 13% slower in writing than bare metal. These differences can also be attributed to the additional virtualization and overall resource consumption. While 17% slower reading is not good we rate it still as acceptable so HPK and KSI are rated as ○ and Bridge Operator as +.

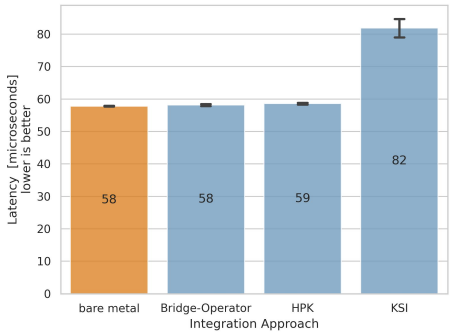

Figure 5. Network latency results using Netperf.

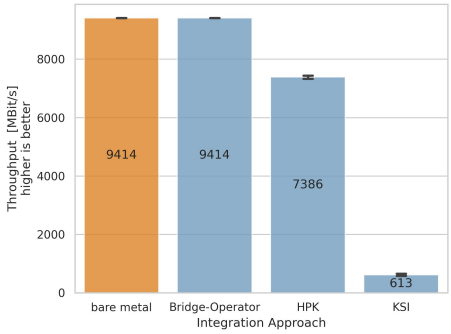

Figure 6. Network bandwidth results using iPerf3.

5) Network Performance: For these benchmarks, the respective tool executed a workload containing a test client that executed the network benchmark against a server running on the other of the two nodes in our test setup. What is shown as the bare metal latency and throughput are therefore the latency and peak throughput between the two nodes. Figure 5 shows network latency with all solutions on the same level except for KSI, which is 42% slower than bare metal. In Figure 6 the network throughput is even worse for KSI with HPK already being 21% slower than bare metal, KSI is 93.5% slower. As KSI operates via rootlesss Podman, it uses slirp4netns as its driver, which according to the Podman documentation [54] results in degraded performance compared to rootful Podman networking. Since our experiments concluded, pasta had replaced slirp4netns as the default network driver for rootless Podman, which promises better performance but initial tests could not show a significant overall improvement [55]. Our ratings are + for Bridge Operator, ○ for HPK and − for KSI.

D. Evaluation

TABLE IV. PROJECT ASSESSMENT REGARDING QUALITY REQUIREMENTS. * SELF-EVALUATION OF KSI IS NOT UNBIASED.

Project	RQ4 Startup	RQ4 Comp.	RQ4 Storage	RQ4 Net.	RQ5 Usab.	RQ6 Maintainab.
B-O	+	+	+	+	○	○
HPK	+	+	○	○	○	+
KSI	○	+	○	−	+*	+*

Table IV summarizes the ratings we have assigned throughout this section with Startup, Compute, Storage and Network performance all aiming at **RQ4** and Usability and Maintainability aiming at **RQ5** and **RQ6**, respectively.

Bridge Operator has shown performance close or identical to bare metal, which is as expected since it effectively submits a Slurm job through Kubernetes and does not start additional software in that Slurm job. This brings some limitations as it is not actually running a given workload using Kubernetes. Nevertheless, it has presented itself as a valid approach for extending a Kubernetes cluster via access to a Slurm cluster.

HPK provides a good middle ground for running Kubernetes jobs on Slurm with some performance deficiencies compared to bare metal. If WLM-Operator would be functional, we would have probably seen similar performance to HPK as WLM-Operator is based on Singularity and HPK is based on Apptainer and both projects still share the majority of their implementation. While HPK does not support all Kubernetes features, e.g., services and `kubectl exec` are not supported, it provides a solid choice for natively running Kubernetes workloads through Slurm.

KSI is functionally the most complete Kubernetes environment within a Slurm job and requires no external parts to be started and kept running outside of it. This comes with performance costs, as KSI shows the weakest performance in all benchmarks, especially in startup time and networking. The slow startup time is understandable as KSI has to bootstrap the Kubernetes control plane and cannot rely on an existing Kubernetes cluster. For network performance, KSI relies on slirp4netns, which is known for causing performance degradation [54]. While other network drivers are available they are necessarily a silver bullet to resolve this issue [55].

All the projects suffer from being either only proof-of-concept implementations, not having being maintained or not being properly documented such that none of them provide a production ready solution for running Kubernetes inside of Slurm jobs.

V. CONCLUSION AND FUTURE WORK

In this paper, we analyzed the state of solutions for combining Slurm and Kubernetes with the goal to enable dynamic computation between either environment. We focused on a subset of the available solutions to support our use case of running Kubernetes workloads on an existing Slurm cluster.

For this purpose, we introduced our own solution KSI, which is based on Kind and rootless Podman and is able to deploy a fully functional Kubernetes cluster inside a Slurm job. From the available solutions we took a closer look at Bridge Operator, HPK and KSI and found that they fulfill our functional requirements, except for KSI, for which multi-node support has not been implemented yet. We further evaluated their performance and reviewed the state of their respective implementations.

We found that Bridge Operator delivers effectively bare metal performance equal to directly running a job through Slurm as this is effectively what Bridge Operator does. HPK established itself as a middle ground solution, providing an almost fully functional Kubernetes cluster inside a Slurm job with minor performance overhead. On the other hand, our solution KSI showed slightly higher overhead compared to HPK and significantly less network throughput. With this KSI provides the most feature complete Kubernetes clusters but should not be considered for workloads that significantly rely on network throughput compared to other factors. Still, if a user's workload relies on Kubernetes networking features such as services, from the evaluated solutions, only KSI can support this.

We conclude that the problem of running Kubernetes inside Slurm workloads is not fully solved as it either comes at the cost of performance or reduced feature sets with only unmaintained or proof-of-concept solutions available.

The next steps for KSI include evaluating different network drivers to mitigate its biggest short coming and to extend it to support multi-node workloads while exploring alternative approaches based on Minikube, K3D or Usersnetes.

REFERENCES

[1] "CNCF Annual Survey 2023", CNCF, Apr. 9, 2024, [Online]. Available: https://www.cncf.io/reports/cncf-annual-survey-2023/ (visited on 2024.12.30).

[2] "9 Insights on Real-World Container Use | Datadog", [Online]. Available: https://web.archive.org/web/20230318234844/https://www.datadoghq.com/container-report/ (visited on 2024.12.30).

[3] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management", in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds., Berlin, Heidelberg: Springer, 2003, pp. 44–60, ISBN: 978-3-540-39727-4. DOI: 10.1007/10968987_3.

[4] "Volcano-sh/volcano", Volcano, Dec. 30, 2024, [Online]. Available: https://github.com/volcano-sh/volcano (visited on 2024.12.30).

[5] "Scaling Kubernetes to 7,500 Nodes", Jan. 2021, [Online]. Available: https://openai.com/blog/scaling-kubernetes-to-7500-nodes/ (visited on 2021.02.12).

[6] A. M. Beltre, P. Saha, M. Govindaraju, A. Younge, and R. E. Grant, "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)*, Nov. 2019, pp. 11–20. DOI: 10.1109/CANOPIE-HPC49598.2019.00007.

[7] T. Wickberg, "Slurm and/or/vs Kubernetes", Dec. 30, 2024, [Online]. Available: https://slurm.schedmd.com/SC23/Slurm-and-or-vs-Kubernetes.pdf (visited on 2024.12.30).

[8] S. Metje, "Kubernetes Slurm Integration based on Kind", 2023, [Online]. Available: https://github.com/soerenmetje/kind-slurm-integration.

[9] "Kind – Rootless", [Online]. Available: https://kind.sigs.k8s.io/docs/user/rootless/ (visited on 2024.12.30).

[10] B. Lublinsky, E. Jennings, and V. Spišaková, "A kubernetes 'bridge' operator between cloud and external resources", in *2023 8th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*, 2023, pp. 263–269. DOI: 10.1109/ICCCBDA56900.2023.10154770.

[11] Staff, "Introducing HPC Affinities to the Enterprise: A New Open Source Project Integrates Singularity and Slurm via Kubernetes", Sylabs, May 7, 2019, [Online]. Available: https://sylabs.io/2019/05/introducing-hpc-affinities-to-the-enterprise-a-new-open-source-project-integrates-singularity-and-slurm-via-kubernetes/ (visited on 2024.12.30).

[12] K. Peterson, "Kalenpeterson/kube-slurm", Aug. 17, 2024, [Online]. Available: https://github.com/kalenpeterson/kube-slurm (visited on 2024.12.30).

[13] A. Chazapis, F. Nikolaidis, M. Marazakis, and A. Bilas, "Running kubernetes workloads on HPC", in *High Performance Computing*, A. Bienz, M. Weiland, M. Baboulin, and C. Kruse, Eds., ser. Lecture Notes in Computer Science, Cham: Springer Nature Switzerland, 2023, pp. 181–192, ISBN: 978-3-031-40843-4. DOI: 10.1007/978-3-031-40843-4_14.

[14] S. Metje, *Running Kubernetes Workloads on Rootless HPC Systems using Slurm*, GRO.data, Jan. 9, 2024. DOI: 10.25625/GDFCFP.

[15] F. Liu, K. Keahey, P. Riteau, and J. Weissman, "Dynamically negotiating capacity between on-demand and batch clusters", in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18, Dallas, Texas: IEEE Press, Nov. 11, 2018, pp. 1–11.

[16] B. Wu, M. Hu, S. Qin, and J. Jiang, "Research on fusion scheduling based on Slurm and Kubernetes", in *International Conference on Algorithms, High Performance Computing, and Artificial Intelligence (AHPCAI 2024)*, vol. 13403, SPIE, Nov. 18, 2024, pp. 476–485. DOI: 10.1117/12.3051639.

[17] G. Zervas, A. Chazapis, Y. Sfakianakis, C. Kozanitis, and A. Bilas, "Virtual clusters: Isolated, containerized HPC environments in kubernetes", in *High Performance Computing. ISC High Performance 2022 International Workshops*, H. Anzt, A. Bienz, P. Luszczek, and M. Baboulin, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2022, pp. 347–357, ISBN: 978-3-031-23220-6. DOI: 10.1007/978-3-031-23220-6_24.

[18] T. Menouer, N. Greneche, C. Cérin, and P. Darmon, "Towards an Optimized Containerization of HPC Job Schedulers Based on Namespaces", in *Network and Parallel Computing*, C. Cérin, D. Qian, J.-L. Gaudiot, G. Tan, and S. Zuckerman, Eds., Cham: Springer International Publishing, 2022, pp. 144–156, ISBN: 978-3-030-93571-9. DOI: 10.1007/978-3-030-93571-9_12.

[19] C. Cérin, N. Greneche, and T. Menouer, "Towards Pervasive Containerization of HPC Job Schedulers", in *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Sep. 2020, pp. 281–288. DOI: 10.1109/SBAC-PAD49847.2020.00046.

[20] "SlinkyProject/slurm-operator", SlinkyProject, Dec. 26, 2024, [Online]. Available: https://github.com/SlinkyProject/slurm-operator (visited on 2024.12.30).

[21] P. Liu and J. Guitart, *Fine-grained scheduling for containerized HPC workloads in kubernetes clusters*, Nov. 21, 2022. DOI: 10.48550/arXiv.2211.11487. arXiv: 2211.11487[cs].

[22] D. Medeiros, J. Wahlgren, G. Schieffer, and I. Peng, "Kub: Enabling elastic HPC workloads on containerized environments", in *2023 IEEE 35th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*,

ISSN: 2643-3001, Oct. 2023, pp. 219–229. DOI: 10.1109/SBAC-PAD59825.2023.00031.

[23] "PRIMAGE / hpc-connector · GitLab", GitLab, Feb. 22, 2023, [Online]. Available: https://gitlab.com/primageproject/hpc-connector (visited on 2025.01.02).

[24] N. Zhou *et al.*, "Container orchestration on HPC systems through Kubernetes", *Journal of Cloud Computing*, vol. 10, no. 1, p. 16, Feb. 22, 2021, ISSN: 2192-113X. DOI: 10.1186/s13677-021-00231-z.

[25] G. Staples, "TORQUE resource manager", in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06, New York, NY, USA: Association for Computing Machinery, Nov. 11, 2006, 8–es, ISBN: 978-0-7695-2700-0. DOI: 10.1145/1188455.1188464.

[26] "Sylabs/wlm-operator", Sylabs Inc., Nov. 5, 2024, [Online]. Available: https://github.com/sylabs/wlm-operator (visited on 2025.01.02).

[27] "Sylabs/singularity-cri", Sylabs Inc., Mar. 1, 2024, [Online]. Available: https://github.com/sylabs/singularity-cri (visited on 2025.01.02).

[28] B. Lublinsky, E. Jennings, and V. Spišaková, "A Kubernetes 'Bridge' operator between cloud and external resources", Jul. 6, 2022, arXiv: 2207.02531 [cs], [Online]. Available: http://arxiv.org/abs/2207.02531 (visited on 2025.01.02), pre-published.

[29] "Bridge-Operator/kubeflow at main · IBM/Bridge-Operator", [Online]. Available: https://github.com/IBM/Bridge-Operator/tree/main/kubeflow (visited on 2025.01.02).

[30] "CARV-ICS-FORTH/HPK", Computer Architecture and VLSI Systems (CARV) Laboratory, Dec. 26, 2024, [Online]. Available: https://github.com/CARV-ICS-FORTH/HPK (visited on 2025.01.02).

[31] A. Chazapis, E. Maliaroudakis, F. Nikolaidis, M. Marazakis, and A. Bilas, "Running Cloud-native Workloads on HPC with High-Performance Kubernetes", Sep. 25, 2024, arXiv: 2409.16919 [cs], [Online]. Available: http://arxiv.org/abs/2409.16919 (visited on 2025.01.02), pre-published.

[32] "Apptainer/apptainer", The Apptainer Container Project, Dec. 30, 2024, [Online]. Available: https://github.com/apptainer/apptainer (visited on 2025.01.02).

[33] "Virtual-kubelet/virtual-kubelet", virtual kubelet, Feb. 24, 2025, [Online]. Available: https://github.com/virtual-kubelet/virtual-kubelet (visited on 2025.02.24).

[34] "Flannel-io/flannel", flannel-io, Feb. 25, 2025, [Online]. Available: https://github.com/flannel-io/flannel (visited on 2025.02.25).

[35] "Flannel-io/cni-plugin", flannel-io, Jan. 31, 2025, [Online]. Available: https://github.com/flannel-io/cni-plugin (visited on 2025.02.25).

[36] "Fix #2 #3 #6 by soerenmetje · Pull Request #4 · IBM/Bridge-Operator", [Online]. Available: https://github.com/IBM/Bridge-Operator/pull/4 (visited on 2025.01.02).

[37] "Kubernetes/minikube", Kubernetes, Jan. 2, 2025, [Online]. Available: https://github.com/kubernetes/minikube (visited on 2025.01.02).

[38] "K3d-io/k3d", k3d, Jan. 2, 2025, [Online]. Available: https://github.com/k3d-io/k3d (visited on 2025.01.02).

[39] "Rootless-containers/usernetes", rootless-containers, Dec. 30, 2024, [Online]. Available: https://github.com/rootless-containers/usernetes (visited on 2025.01.02).

[40] "Kind", [Online]. Available: https://kind.sigs.k8s.io/ (visited on 2025.02.21).

[41] L. S. Marín, "Squat/kilo", Dec. 24, 2024, [Online]. Available: https://github.com/squat/kilo (visited on 2025.01.02).

[42] "Liqotech/liqo", LiqoTech, Dec. 27, 2024, [Online]. Available: https://github.com/liqotech/liqo (visited on 2025.01.02).

[43] "Rootless-containers/slirp4netns", rootless-containers, Feb. 23, 2025, [Online]. Available: https://github.com/rootless-containers/slirp4netns (visited on 2025.02.25).

[44] "Releases · containers/podman", GitHub, [Online]. Available: https://github.com/containers/podman/releases (visited on 2025.02.25).

[45] "Passt - Plug A Simple Socket Transport", [Online]. Available: https://passt.top/passt/about/ (visited on 2025.02.25).

[46] A. Kopytov, "Akopytov/sysbench", Jan. 2, 2025, [Online]. Available: https://github.com/akopytov/sysbench (visited on 2025.01.02).

[47] J. Hammond, "Jeffhammond/STREAM", Dec. 24, 2024, [Online]. Available: https://github.com/jeffhammond/STREAM (visited on 2025.01.02).

[48] J. Axboe, "Flexible I/O Tester", 2022, [Online]. Available: https://github.com/axboe/fio (visited on 2025.01.02).

[49] "HewlettPackard/netperf", Hewlett Packard Enterprise, Dec. 10, 2024, [Online]. Available: https://github.com/HewlettPackard/netperf (visited on 2025.01.02).

[50] "Esnet/iperf", ESnet: Energy Sciences Network, Jan. 2, 2025, [Online]. Available: https://github.com/esnet/iperf (visited on 2025.01.02).

[51] L. Ardito, R. Coppola, L. Barbato, and D. Verga, "A Tool-Based Perspective on Software Code Maintainability Metrics: A Systematic Literature Review", *Scientific Programming*, vol. 2020, no. 1, p. 8 840 389, 2020, ISSN: 1875-919X. DOI: 10.1155/2020/8840389.

[52] K. A. Dawood *et al.*, "Towards a unified criteria model for usability evaluation in the context of open source software based on a fuzzy Delphi method", *Information and Software Technology*, vol. 130, p. 106 453, Feb. 1, 2021, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2020.106453.

[53] S. Metje, "Soerenmetje/kubernetes-slurm-evaluation", Jun. 28, 2024, [Online]. Available: https://github.com/soerenmetje/kubernetes-slurm-evaluation (visited on 2025.01.02).

[54] "Podman/docs/tutorials/performance.md at main · containers/podman", GitHub, [Online]. Available: https://github.com/containers/podman/blob/main/docs/tutorials/performance.md (visited on 2025.01.03).

[55] "Rootless network performance (pasta vs slirp4netns) · containers/podman · Discussion #22559", GitHub, [Online]. Available: https://github.com/containers/podman/discussions/22559 (visited on 2025.01.03).