Serverless Computing for Dynamic HPC Workflows

Vijay Thurimella[1, 2], Philipp Raith[2], Rolando P. Hong Enriquez[2], Anderson Andrei Da Silva[2],
Gourav Rattihalli[2] Ada Gavrilovska[1], Dejan Milojicic[2]
[1]*School of Computer Science*, *Georgia Institute of Technology*, Atlanta, USA
[2]*Hewlett Packard Labs*, *Hewlett Packard Enterprise*, Milpitas, CA
vthurimella@gatech.edu, philipp.raith@hpe.com, rhong@hpe.com, da-silva@hpe.com,
gourav.rattihalli@hpe.com, ada@cc.gatech.edu, dejan.milojicic@hpe.com

***Abstract*—Containers have become an important component for scientific workflows, enhancing reproducibility, portability, and isolation when coupled with workflow management systems. However, integrating containers with these systems can be complex, potentially hindering wider adoption. Serverless platforms offer a solution by providing a layer of abstraction over container orchestrators, simplifying management while introducing event-driven capabilities. This paper presents a novel integration of serverless with workflow management systems to optimize scientific workflow execution. Our approach leverages serverless functions to dynamically provision containers for workflow tasks, resulting in up to 30% faster execution. We found that performance can be further improved by reusing containers between multiple different tasks that were provisioned by the serverless platform. These findings demonstrate the utility of combining specialized container orchestration with established workflow management to streamline scientific computing, improve resource utilization, and accelerate time-to-results. Serverless' event-driven architecture enables efficient resource scaling, aligning with the dynamic nature of scientific workloads.**

I. Introduction

Reproducibility, portability, and performance isolation are essential for high-performance computing (HPC) workflows in shared environments [1], [2]. It has been recognized that containerization goes a long way in meeting these requirements. Containers offer significant advantages in cluster configuration for workflow execution. They provide a portable, self-contained environment that encapsulates all necessary dependencies, libraries, and configurations. This portability allows tasks within a workflow to run on any node in the cluster that has a container runtime installed, regardless of the underlying system configuration. In heterogeneous clusters, where nodes may have different operating systems or software versions, containers mitigate the dependency hell problem [3]. Without containers, managing software dependencies across diverse nodes can be complex and error-prone, often leading to compatibility issues and workflow failures. Containers effectively standardize the execution environment, ensuring consistency across the cluster and simplifying overall cluster management. This approach not only enhances workflow reproducibility but also improves scalability and resource utilization, as tasks can be more flexibly distributed across available nodes.

There has been previous work [4] in integrating containers with workflow management systems in order to achieve these properties. However, these integration do not take full advantage of serverless computing ability for managing, reusing and scaling containers. Serverless computing offers significant advantages when integrated with workflow managers, particularly in terms of scalability. Unlike traditional infrastructure, serverless platforms can automatically scale containers up or down based on the current workload. This dynamic scaling capability is especially beneficial for scientific workflows that often have varying computational demands across different stages of the workflow. Workflow managers can leverage this dynamic scale to efficiently handle bursts of activity or periods of low demand without manual intervention. This scalability, combined with the ease of deployment and management, makes serverless an attractive option for enhancing the performance and efficiency of workflow management systems in scientific computing environments. In addition to its ability to scale containers, serverless platforms can also reuse containers. Previously, each workflow task would create a new container, but with serverless, multiple tasks can share the same container between invocations. This reduces the overhead associated with managing containers.

This paper demonstrates the potential of integrating serverless computing with traditional workflow management systems to achieve a fine-grained trade-off between execution time and performance isolation. By combining serverless and workflow management, we achieve greater flexibility and performance. Our approach leverages serverless functions to dynamically provision containers for workflow tasks, resulting in considerable faster execution. We found that performance can be further improved by reusing containers through serverless platforms. In summary, we make the following contributions:

- **Integration of Serverless with Workflow Management Systems**: Proposed and demonstrated integrating Knative, an open-source serverless platform, with the Pegasus workflow management system to enhance container orchestration and introduce serverless capabilities.
- **Efficient Container Management and Provisioning**: Developed methods to containerize workflow tasks and register them with Knative, leveraging its capabilities for efficient container distribution and management. This includes pre-staging containers on worker nodes or deferring downloads until task invocation, thus significantly reducing execution time.
- **Enhanced Data Accessibility and Transparent Invocation**: Implemented strategies to ensure seamless access to data for containerized tasks. Modified workflows to invoke containerized functions via HTTP requests, leveraging Knative's pre-registered containers.
- **Experimental Validation and Performance Insights**:

979-8-3503-5554-3/24/$31.00 ©2024 IEEE
DOI 10.1109/SCW63240.2024.00262

Conducted experiments demonstrating the viability of the approach, showing a trade-off between performance isolation and execution efficiency.

The results highlight the benefits of combining Knative with Pegasus for large-scale scientific computing, enhancing both performance and resource utilization.

II. Preliminaries

A. Containers

Containers have improved software deployment by providing lightweight, portable, and self-sufficient units that package applications and their dependencies [5]. They provide a consistent environment for applications to run across different computing infrastructures, from development to production. The foundation of container technology lies in Control Groups [6] (cgroups), a Linux kernel feature that enables fine-grained control over resource allocation and isolation for processes. Cgroups allow for management of system resources. This maintains performance isolation in multi-tenant environments. Beyond limiting resources, cgroups enable the assignment of different priorities to process groups.

1) Isolation: Resource isolation, a key feature of containers, ensures that applications within a container have access only to a predefined set of resources. This isolation prevents resource contention between multiple applications. It also improves security by limiting the potential impact of compromised applications. Resource isolation can allow scientists to more accurately estimate the resource requirements, cost and execution time of their workflows by providing predictable performance metrics and resource usage patterns.

2) Orchestration: Container orchestration refers to the automated management, deployment, scaling, and networking of containers. Kubernetes [7] is the standard for container orchestration in both industry and research environments. Kubernetes manages containerized applications with features for automatic deployment and scaling based on defined rules or metrics. It optimizes resource utilization in dynamic workloads and provides advanced service discovery and load balancing, facilitating container communication and stable network traffic distribution.

B. Serverless

Traditionally, virtual machines (VMs) were the primary compute units. Serverless computing offers a finer-grained approach, allowing users to allocate resources precisely for each task, thus achieving better resource utilization. This model, characterized by automatic scaling and dynamic resource management, eliminates the need for provisioning and maintaining servers. By efficiently managing containers, including reuse, serverless platforms reduce cold start times and enhance performance, especially for short-lived tasks. This abstraction enables developers to focus on application logic, accelerating development and reducing operational overhead.

Knative [8], a serverless platform built on Kubernetes, provides the necessary functionality for building event-driven applications. With components like *Serving* and *Eventing*, Knative offers automatic scaling, efficient resource utilization through scale-to-zero, and flexible event management. Kubernetes [7], while powerful, lacks specialized container management features tailored for scientific workflows. To address this, we propose integrating Knative with Pegasus to optimize scientific workflow execution, leveraging Knative's event-driven capabilities and elastic scaling for improved performance and resource utilization.

C. Workflow Management Systems

Workflow Management Systems (WMS) are used for automating and orchestrating scientific computations [9]. By managing task dependencies, data transfer, and resource allocation, WMS's simplify distributed computing, enabling scientists to concentrate on research rather than infrastructure.

Pegasus [10] is a widely adopted WMS that highlights the capabilities of this technology. Pegasus specializes in mapping abstract workflow descriptions onto distributed computing resources. It separates the workflow description from the execution environment to create portability across different computing platforms. Pegasus does this by having the workflow developer describe all of the tasks in the workflow language as transformations. After registering the task as a Transformation developers can invoke the task in the workflow by creating a Job of the Transformation.

It offers fault-tolerance mechanisms, including task retry and checkpoint/restart capabilities, which can be very helpful for long-running scientific experiments. Pegasus also performs workflow restructuring and task clustering to improve execution efficiency and reduce redundancy, to optimize resource usage.

III. Motivation

High-Performance Computing (HPC) environments often use shared clusters to maximize resource efficiency. However, maintaining *performance isolation* in these clusters is crucial to ensure that individual workloads receive predictable and consistent performance, regardless of other users' activities. This prevents resource contention and guarantees fair resource allocation. While shared clusters optimize overall resource utilization, they introduce significant challenges related to performance isolation. Workflows on shared clusters frequently face resource contention, where individual tasks within a workflow can monopolize resources, causing starvation for other users or tasks. This issue becomes especially problematic when a single task occupies an entire node, potentially disrupting the execution of multiple workflows.

Containers have emerged as a promising solution to this challenge by providing isolated execution environments for each task. This isolation prevents resource monopolization and ensures fair resource allocation across multiple users and workflows. However, traditional workflow management systems which were designed primarily for static resource allocation, struggle to effectively manage containerized workloads. This mismatch between containerization technology and legacy workflow management systems hinders the full potential of containerization in HPC environments.

In response to these challenges, Kubernetes and Knative have emerged as compelling platforms for containerized workflow execution. Kubernetes provides robust orchestration and management capabilities for containerized applications, while Knative offers serverless functions and automatic scaling. These tools are well-suited for dynamic resource allocation, performance isolation, and effective reuse of containers, which are essential for efficient workflow execution in shared HPC clusters.

A. Portability

Containers offer significant advantages in cluster configuration for workflow execution. They provide a portable, self-contained environment that encapsulates all necessary dependencies, libraries, and configurations. This portability allows tasks within a workflow to run on any node in the cluster that has a container runtime installed, regardless of the underlying system configuration. In heterogeneous clusters, where nodes may have different operating systems or software versions, containers mitigate the dependency hell problem [3]. Without containers, managing software dependencies across diverse nodes can be complex and error-prone, often leading to compatibility issues and workflow failures. Containers effectively standardize the execution environment, ensuring consistency across the cluster and simplifying overall cluster management. This approach not only enhances workflow reproducibility but also improves scalability and resource utilization, as tasks can be more flexibly distributed across available nodes.

B. Container Reuse

Container reuse is a key advantage of serverless computing in scientific workflows. By maintaining warm containers, serverless platforms can significantly reduce the overhead associated with container startup and teardown, which is particularly beneficial for workflows comprising numerous short-duration tasks. This approach allows subsequent tasks to leverage pre-initialized runtime environments, dramatically decreasing overall execution time.

To quantify this benefit, we conducted an experiment comparing Docker and Knative performance for executing multiple small tasks. Each task involved matrix multiplication: reading two input matrices from disk, computing their product, and writing the result back to disk. We varied the total number of sequentially executed tasks and measured the overall execution time. In the Docker setup, each task ran in a new container, executed from the command line using docker run. For Knative, tasks were run on a 4-node cluster where the input data was stored on the node and the HTTP request was sent via a python script. Knative tasks experienced a cold start, which Figure 1 shows to be 1.48 seconds, but operated within the same container structure, enabling reuse.

The results from this experiment, illustrated in Figure 1, show a performance improvement as task count increases. Docker's total time grew rapidly, far outpacing the actual computation time due to container creation and destruction overhead. Knative's total time increased more slowly, remaining closer to the execution time even at higher task counts. At 160 tasks, Docker's total time reached about 100 seconds, while Knative's was around 78 seconds - a clear efficiency gain. Notably, the execution times of individual tasks increased as more tasks were executed, but these times remained similar between Knative and Docker, highlighting that the performance difference stems from container management overhead rather than computation. This difference is due to Knative's ability to reuse container structures, reducing cumulative container lifecycle management overhead. The efficiency is particularly valuable in scientific workflows with numerous short-duration tasks. These findings suggest serverless platforms offer a promising solution for optimizing containerized workflow execution in HPC environments, especially for workflows comprising many small tasks.

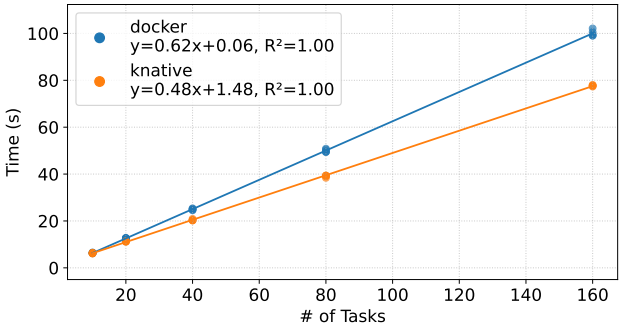

Fig. 1: Docker's execution time increases rapidly due to per-task container overhead, while Knative scales more efficiently by reusing containers. Analysis of the regression slopes indicates Knative can reduce overall execution time by up to 30% compared to Docker.

C. Scaling

Serverless computing offers significant advantages when integrated with workflow managers, particularly in terms of scalability. Unlike traditional infrastructure, serverless platforms can automatically scale resources up or down based on the current workload, ensuring optimal resource utilization. This dynamic scaling capability is especially beneficial for scientific workflows that often have varying computational demands across different stages. Workflow managers can leverage this dynamic scale to efficiently handle bursts of activity or periods of low demand without manual intervention. This scalability, combined with the ease of deployment and management, makes serverless an attractive option for enhancing the performance and efficiency of workflow management systems in scientific computing environments.

Our experiment demonstrates how Knative, a serverless platform, can process parallel tasks more efficiently than traditional container execution on HTCondor.

We designed a workflow consisting of multiple parallel matrix multiplication tasks, leveraging our integration of Pegasus and Knative. This integration was crucial for the experiment, as attempting to run concurrent Knative tasks without HTCondor's queuing ability caused the virtual machine to crash.

Additionally, HTCondor provided the necessary queuing and scheduling capabilities for managing many concurrent tasks.

Figure 2 illustrates the results of our experiment, comparing the execution time of native, Knative, and container-based approaches as the number of parallel tasks increases. The plot reveals that Knative scales more efficiently than traditional containerized execution on HTCondor, especially as the task count grows.

This performance advantage stems from two key factors. First, Knative allows multiple tasks to be co-located within the same container, reducing overhead associated with container creation and destruction. Second, Knative's automatic scaling functionality dynamically creates new containers and distributes them across worker nodes when a single container cannot service all concurrent tasks efficiently.

The linear regression lines in the plot quantify these differences. Native execution shows the lowest slope (0.28), followed closely by Knative (0.30), while container execution on HTCondor has a significantly steeper slope (0.96). This indicates that as the number of parallel tasks increases, the execution time for Knative grows at a rate similar to native execution, while traditional containerized execution experiences a much faster increase in execution time.

These findings highlight the potential of serverless platforms like Knative to optimize the execution of highly parallel workflows in HPC environments. By efficiently managing container resources and automatically scaling to meet demand, Knative offers a promising solution for improving the performance of scientific workflows with large numbers of concurrent tasks.

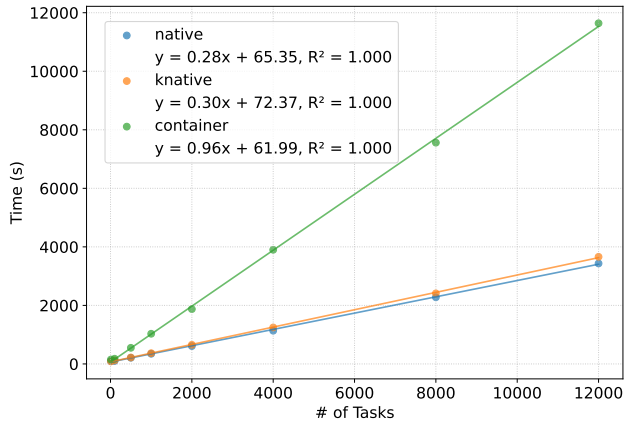

Fig. 2: Native execution scales best for parallel tasks, with Knative close behind and traditional containerization slowest, as evidenced by their regression slopes. This highlights serverless computing's efficiency in managing containers in parallel workflows, performing much closer to native execution than traditional containerization.

It is important to note that this speedup comes at the cost of added workflow file management complexity. Since the serverless function's execution location is determined when the function is invoked, the workflow can't know where to move the file to. To enable the function to access the input files for the task, the files need to be stored in a location that is accessible to the function, such as a shared file system.

IV. Prototype Implementation

Unlike previous approaches that developed custom solutions for workflow management systems to address container challenges [4], this paper advocates using serverless platforms for container management. We specifically looked at integrating Pegasus and Knative. Knative is attractive because it simplifies container orchestration and provides serverless functionality for dynamic resource provisioning. We demonstrate this approach by transforming Pegasus workflow tasks into containerized functions managed by Knative. To accomplish this, we addressed several key challenges:

1) **Task Containerization and Registration**: We developed a method to containerize tasks and register them with Knative prior to workflow execution. This process involves encapsulating tasks in HTTP event listeners and registering them with Knative before the workflow runs. The task registration with the serverless system was done manually before the execution of the workflow.
2) **Container Provisioning**: Our system utilizes Knative's capabilities for efficient container management and distribution. During **function registration**, Knative provisions pods that download the necessary containers to the assigned nodes. Users can customize this process by setting the fields in the function registration metadata, "*autoscaling.knative.dev/min-scale*" to specify the number of worker nodes that should download the container ahead of time. Setting "*autoscaling.knative.dev/initial-scale*" to zero defers container downloads until a task is actually invoked on a worker, this would be similar to how Pegasus passes the container to the worker node at the time of execution of the job
3) **File Management in Serverless**: To ensure containerized tasks can access their input files, we implemented a strategy where input data is sent in the function invocation as part of the invocation network request. This evaluation strategy is similar to a pass by value when making a function call. Output data is similarly returned from the serverless invocation call. The resulting output is then written to a file by the wrapper script that is the replaced job in the executable workflow.
4) **Transparent Task Invocation**: We modified the workflow to invoke the containerized functions via network requests, synchronously waiting for completion. This involved replacing the original task in the executable workflow with a new task that invokes the pre-registered container in Knative, passing the same parameter by value to invoke the serverless function. The critical path of execution now has been extended as a wrapper task is now scheduled by HTCondor onto a worker node and that worker node will synchronously send a request to the serverless function. In the case of large data, inputs if the serverless invocation is scheduled on a different worker node redundant data movements occurs from the

submit node, to the first worker node and then to the execution node of the serverless invocation.

V. Experimental Methodology

A. Software & Hardware Configuration

Our experimental setup consists of a cluster of four virtual machines. Each VM is equipped with 8 cores and 32 GB of RAM. The CPUs used are Intel(R) Xeon(R) Gold 6342 processors running at 2.80GHz. One VM serves as the submit node for Condor and hosts the Kubernetes control plane.

The software configuration for our study includes several key components. We utilize Pegasus version 5.0.7 for workflow management. The distributed computing environment is managed by HTCondor version 23.8.1 (BuildID: 742100, PackageID: 23.8.1-1.1). For container orchestration, we employ Kubernetes, with both the client and server running version v1.30.3. Knative, used for serverless deployments, is at version controller@sha256:2948cacc3. Our computational analysis is primarily conducted using Python version 3.10.12, with NumPy version 2.0.1 providing essential numerical computing capabilities.

B. Workflow Configuration

The computational task for each step involves matrix multiplication of two 350×350 matrices. These matrices are stored on disk and contain integers ranging from -100 to 100. The output of each task is another 350×350 matrix, which then serves as input for the subsequent task in the workflow.

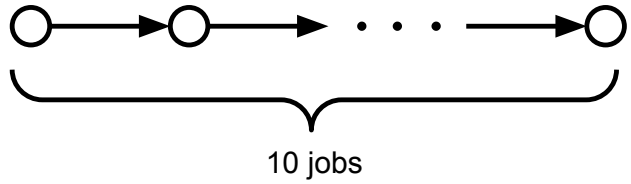

Fig. 3: This illustration depicts a single instance of a workflow used in our experimental evaluation. The workflow comprises a series of sequential jobs, each performing a matrix multiplication. These jobs are executed in one of three environments: natively on a worker node, within a container on a worker node, or in a container using the serverless platform.

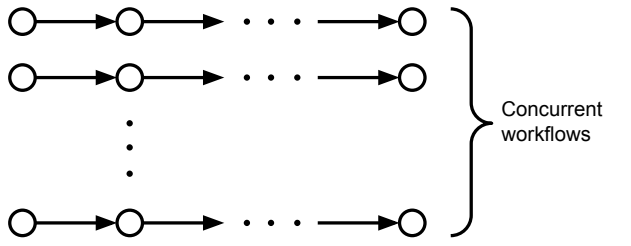

Fig. 4: A set of concurrent workflows each a sequential workflow shown in Figure 3.

C. Execution Environments

Our study compares serial workflows with matrix multiplication tasks across different execution platforms. We implement the task in three distinct execution contexts:

- **Setup 1**: native execution in Pegasus;
- **Setup 2**: traditional containerized execution in Pegasus;
- **Setup 3**: Knative-based serverless execution.

The task structure remains consistent across platforms: reading two input matrices from disk, performing matrix multiplication, and writing the result back to disk. Each workflow consists of 10 matrix multiply tasks, as shown in Figure 3. The distribution of tasks among these platforms is determined randomly before initiating the 10 workflows, ensuring a diverse mix of execution environments for the total 100 tasks. One instance of this experiment is graphically illustrated in Figure 4.

In Setup 1, natively executed tasks read two input matrices from disk, compute the result, and write the output matrix back to disk. In Setup 2, tasks are encoded as transformations in the Pegasus workflow with the container parameter set. These transformations are integrated into the workflow as a Job with two input files and one output file.

For Setup 3, we use Flask [11] to wrap the matrix multiplication task in an HTTP event listener. Tasks executed via Knative consist of two components: the original task wrapped in a Flask HTTP event listener, and a Python script that sends an HTTP event to invoke the task. The invocation script is encoded in Pegasus as a Transformation, replacing the previously containerized Job in the workflow. The containerized application is deployed on Knative *before* workflow execution.

We conduct experiments in two scenarios: containers distributed to workers, created and run before workflow execution, and containers distributed but not created before workflow execution. In the first scenario, we configure Knative to allow multiple requests per container, enabling container reuse across workflow requests.

For comparison, we implement traditionally containerized tasks in Pegasus. These requests are encoded in Pegasus as Transformations with the container dependency set. The associated container images contain the same Python code as the native tasks and are accessible via DockerHub.

D. Performance Metric

Our primary performance metric is the average execution time of the slowest workflow among the 10 concurrent runs. This approach allows us to assess the impact of different execution environments on overall workflow completion time.

E. Key Observations

Our study builds on prior research that underscores the significance of container technologies in workflow management. We analyze Pegasus' container implementation and propose using Knative to achieve similar container distribution with greater flexibility. By adjusting Knative's autoscaling parameters, users can precisely control when containers are deployed. Setting the "*autoscaling.knative.dev/initial-scale*" parameter to 0 defers container downloads until task invocation, while a

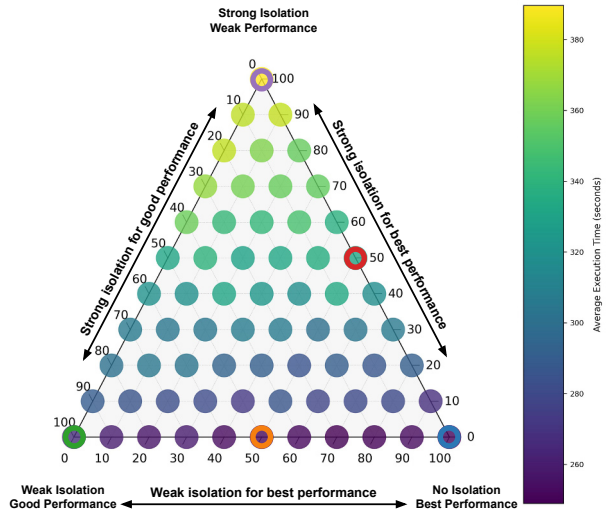

Fig. 5: Performance-isolation trade-off in concurrent workflow execution. The triangle illustrates how native execution achieves the best performance without isolation, individual containers offer strong isolation at a performance cost, and serverless containers balance good performance with weak isolation through container reuse.

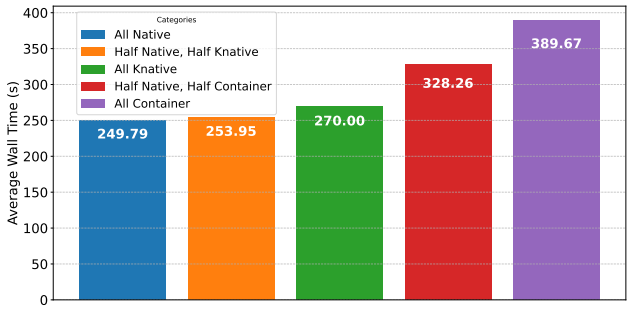

Fig. 6: Average makespan of five highlighted combinations from Figure 5, illustrating the trade-off between gaining performance isolation through containers and achieving better performance.

non-zero setting allows for pre-deployment before workflow execution. A challenge in adopting serverless computing for workflow container orchestration is managing files and specifying task execution locations. We address this by embedding file data in the network requests that invoke Knative tasks. Alternative strategies include using a storage service like Minio [12] or making files network-accessible from the submit node. In our approach, input and output files are transferred via network requests, streamlining data handling within the serverless framework.

VI. Experimental Evaluation

Our results demonstrate that by reusing containers in Knative, we achieve performance close to native execution while benefiting from container isolation. When running multiple tasks concurrently within the same container, we observe better performance compared to running one task per container in Knative.

Knative enables us to achieve various points along the spectrum between execution time and performance isolation using containers for each task. This performance benefit is achieved by reusing containers for multiple tasks (both concurrently and between different stages) and taking advantage of Knative's automatic scaling of containers.

Figure 5 illustrates this trade-off in a ternary plot. The topmost point represents the average makespan of the slowest workflow when all tasks run in separate containers, providing strong isolation. The bottom-left point shows the execution time when all tasks run in Knative containers, allowing only one request per container at a time but reusing the container structure for subsequent tasks. The bottom-right point represents workflows where all tasks run natively, achieving the best performance but lacking isolation. The makespan for each of the points that are highlighted in the figure are shown explicitly in figure 6.

The bar chart depicts five execution scenarios and their corresponding times. The fastest average makespan occurs when all tasks run natively in Pegasus, shown by the blue bar at 250 seconds. The second fastest scenario, represented by the orange bar, involves half of the tasks running on Knative and half running natively. The green bar represents all tasks running on Knative, achieving a performance of 1.08x times native execution. The red bar shows half the tasks in containers in Pegasus and half natively. The slowest execution, represented by the purple bar, occurs when all tasks run in traditional containers.

These figures highlight the performance benefits of using serverless computing for container reuse and scaling. The primary reason for the improvement is that containers do not need to be transferred to the execution node before each task, as Knative can cache and quickly create containers prior to execution and since Knative is able to keep a container between the execution of multiple tasks no container creation or destruction happens.

Through this experiment, we demonstrate that Knative offers a flexible middle ground, allowing users to balance performance isolation and execution efficiency according to their specific needs. These findings suggest that careful container management and task allocation can lead to significant performance improvements in scientific workflows, potentially achieving both the benefits of containerization and performance comparable to native execution.

VII. Related Work

Previous research has explored the integration of containers with scientific workflows to enhance reproducibility, portability, and performance. Moreau et al. [2] emphasize the importance of creating consistent software environments, particularly within complex, collaborative projects. They offer practical recommendations for implementing containerization across diverse scientific fields and high-performance computing clusters. Sweeney and Thain [13] identified key chal-

lenges in incorporating containers into scientific workflows and proposed dynamic composition and image translation strategies to optimize network usage and runtime efficiency. Vahi et al. [14] highlight the challenges of using containers in distributed workflows and discuss Pegasus's container support. Zheng and Thain [4] investigate integrating containers into workflow systems using Makeflow, Work Queue, and Docker, exploring performance implications and container management strategies. Burkat et al. [15] evaluate the use of serverless containers for scientific workflows using AWS Fargate and Google Cloud Run, demonstrating their potential for running scientific applications. Basu Roy et al. [16] introduce Mashup, a hybrid approach combining serverless and traditional cloud computing for executing HPC workflows, achieving significant performance and cost improvements. Bruel et al. [17] argue that with the advent of HPC & AI and its use of accelerators, serverless provides a layer of abstraction that simplifies resource sharing for accelerators by allowing for fine-grained allocation.

While these studies provide valuable insights into containerization and serverless computing in scientific workflows, our work distinguishes itself by focusing on the integration of Knative with Pegasus to optimize workflow execution, leveraging Knative's unique strengths in event-driven orchestration and elastic scaling capabilities.

VIII. Conclusion

Our research demonstrates the potential of integrating serverless computing with traditional workflow management systems to achieve a fine-grained trade-off between execution time and performance isolation. By leveraging Knative's container management capabilities in conjunction with Pegasus, we have created a system that allows workflow developers to seamlessly incorporate containerized tasks into their scientific workflows without modifying their existing workflow descriptions.

This integration offers a range of options for balancing execution time and performance isolation. On one hand, developers can prioritize execution speed by reusing containers across multiple requests, approaching the efficiency of native execution while still benefiting from container isolation. On the other hand, they can opt for stronger isolation by using fresh containers for each task, accepting a slight performance cost. Between these two approaches lies a continuum of options available to workflow developers. This flexibility enables workflow developers to tune their workflows to the specific requirements of their projects, whether they prioritize execution time, performance isolation, or a balance between the two. By providing this versatility, the integration empowers developers to optimize their workflows according to their unique needs and constraints.

Our integration approach offers a transparent solution for workflow developers. By wrapping tasks in event listeners and deploying them as serverless functions, we enable the use of advanced container orchestration features without requiring modifications to existing workflow descriptions. This seamless adoption simplifies the integration of containerization and serverless computing in HPC workflows. However, it's important to note that this method introduces some redundant data movement, as the scheduler must send input data to the wrapper task before it reaches the invoked serverless functions. While this study did not focus on measuring communication requirements relative to previous methods, we recognize that data movement is a critical component of scientific workflow management systems. Our future work will include a comparative study of the communication overheads associated with our approach and existing methods. We are actively exploring solutions to automate data movement and reduce manual handling, aiming to further optimize workflow execution in distributed environments.

As scientific workflows continue to grow in complexity and scale, solutions that offer performance, ease of use and flexibility will become increasingly important.

IX. Future Work

In addition to our primary research directions, we are exploring different opportunities for integrating workflows and serverless to achieve better resource utilization and performance.

A. Comprehensive Workflow Evaluation

Our initial study uses a simple matrix multiplication workflow to isolate the effects of container orchestration and integration with traditional workflow management systems. While this approach provides valuable insights, we recognize the need for a more comprehensive evaluation. In future work, we plan to assess our serverless integration approach using more complex and dynamic scientific workflows. This expanded study will help validate the scalability and efficiency of our solution across a broader range of real-world scenarios.

B. Automated Integration

We are exploring the automation of integration between serverless architectures and scientific workflows, with a particular focus on streamlining the deployment, task rewriting of container functions, and minimizing data movement between traditional cluster and the integrated serverless system. This approach aims to automate both serverless function registration and workflow creation, potentially eliminating the need for developers to make manual code changes to their workflow.

C. Task Resizing

Task sizing in workflows can significantly influence execution efficiency. When multiple tasks are combined into larger units, resource allocators may face challenges in placing these tasks effectively. Conversely, breaking tasks into finer-grained components aligns well with serverless computing paradigms, which excel at allocating resources for very small tasks.

If developers can successfully divide large tasks into smaller, more manageable units without substantially increasing inter-task communication and while leveraging parallelism, it could lead to improved resource utilization and enhanced performance across various workflows. This approach to task sizing has the potential to optimize both the allocation process and overall workflow execution efficiency.

D. Task redirection

Tasks can be scheduled onto compute nodes without full knowledge of current utilization levels. This can lead to situations where some nodes become overloaded while others remain underutilized, resulting in inefficient resource usage and slower overall workflow execution. Serverless computing offers a promising solution to this challenge by enabling dynamic and flexible task placement.

We plan to explore the idea of serverless redirection of tasks away from over-utilized nodes at runtime. This approach has the potential for improving resource utilization across the compute environment and also increasing execution performance.

REFERENCES

[1] R. Ferreira da Silva, K. Chard, H. Casanova, D. Laney, D. Ahn, S. Jha, W. E. Allcock, G. Bauer, D. Duplyakin, B. Enders, T. M. Heer, E. Lançon, S. Sanielevici, and K. Sayers, "Workflows community summit: Tightening the integration between computing facilities and scientific workflows," 2022.

[2] D. Moreau, K. Wiebels, and C. Boettiger, "Containers for computational reproducibility," *Nature Reviews Methods Primers*, vol. 3, no. 1, p. 50, 2023.

[3] K. Liu, K. Aida, S. Yokoyama, and Y. Masatani, "Flexible container-based computing platform on cloud for scientific workflows," in *2016 International Conference on Cloud Computing Research and Innovations (ICCCRI)*, pp. 56–63, 2016.

[4] C. Zheng and D. Thain, "Integrating containers into workflows: A case study using makeflow, work queue, and docker," in *Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing*, pp. 31–38, 2015.

[5] O. Bentaleb, A. S. Z. Belloum, A. Sebaa, and A. El-Maouhab, "Containerization technologies: taxonomies, applications and challenges," *The Journal of Supercomputing*, vol. 78, pp. 1144–1181, Jan 2022.

[6] R. Rosen, "Resource management: Linux kernel namespaces and cgroups," *Haifux, May*, vol. 186, p. 70, 2013.

[7] "Kubernetes," 2014. https://kubernetes.io/ [Accessed: 05.08.2024].

[8] "Knative: Kubernetes-based platform to build, deploy, and manage modern serverless workloads.," 2014. https://github.com/knative [Accessed: 05.08.2024].

[9] C. Schmitt, B. Yu, and T. Kuhr, "A workflow management system guide," 2023.

[10] E. Deelman, K. Vahi, M. Rynge, R. Mayani, R. F. da Silva, G. Papadimitriou, and M. Livny, "The evolution of the pegasus workflow management software," *Computing in Science & Engineering*, vol. 21, no. 4, pp. 22–36, 2019.

[11] M. Grinberg, *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

[12] MinIO, Inc., "MinIO: High Performance Object Storage." https://min.io/, 2024. Accessed: [Insert access date here].

[13] K. M. D. Sweeney and D. Thain, "Efficient integration of containers into scientific workflows," in *Proceedings of the 9th Workshop on Scientific Cloud Computing*, ScienceCloud'18, (New York, NY, USA), Association for Computing Machinery, 2018.

[14] K. Vahi, M. Rynge, G. Papadimitriou, D. A. Brown, R. Mayani, R. Ferreira da Silva, E. Deelman, A. Mandal, E. Lyons, and M. Zink, "Custom execution environments with containers in pegasus-enabled scientific workflows," in *2019 15th International Conference on eScience (eScience)*, pp. 281–290, 2019.

[15] K. Burkat, M. Pawlik, B. Balis, M. Malawski, K. Vahi, M. Rynge, R. F. da Silva, and E. Deelman, "Serverless containers – rising viable approach to scientific workflows," in *2021 IEEE 17th International Conference on eScience (eScience)*, pp. 40–49, 2021.

[16] R. B. Roy, T. Patel, V. Gadepally, and D. Tiwari, "Mashup: making serverless computing useful for hpc workflows via hybrid execution," in *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 46–60, 2022.

[17] P. Bruel, S. R. Chalamalasetti, A. Dhakal, E. Frachtenberg, N. Hogade, R. P. H. Enriquez, A. Mishra, D. Milojicic, P. Prakash, and G. Rattihalli, "Predicting heterogeneity and serverless principles of converged high-performance computing, artificial intelligence, and workflows," *Computer*, vol. 57, no. 1, pp. 136–144, 2024.