

Dynamic Solutions for Hybrid Quantum-HPC Resource Allocation

Original

Dynamic Solutions for Hybrid Quantum-HPC Resource Allocation / Rocco, Roberto; Rizzo, Simone; Barbieri, Matteo; Bettonte, Gabriella; Boella, Elisabetta; Ganz, Fulvio; Iserte, Sergio; Peña, Antonio J.; Sandås, Petter; Scionti, Alberto; Terzo, Olivier; Vercellino, Chiara; Vitali, Giacomo; Viviani, Paolo; Frassinetti, Jonathan; Marzella, Sara; Ottaviani, Daniele; Colonnelli, Iacopo; Gregori, Daniele. - (2025), pp. 34-40. (2025 IEEE International Conference on Quantum Computing and Engineering (QCE) Albuquerque, NM (USA) 30 August 2025 - 05 September 2025)
[10.1109/QCE65121.2025.10289].

Availability:

This version is available at: 11583/3003240 since: 2025-09-22T10:45:40Z

Publisher:

IEEE Computer Society

Published

DOI:10.1109/QCE65121.2025.10289

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

IEEE postprint/Author's Accepted Manuscript

©2025 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

Dynamic Solutions for Hybrid Quantum-HPC Resource Allocation

Roberto Rocco^{*§}, Simone Rizzo^{*}, Matteo Barbieri^{*}, Gabriella Bettonte^{*}, Elisabetta Boella^{*}, Fulvio Ganz^{*}, Sergio Iserte^{††}, Antonio J. Peña^{††}, Petter Sandås^{††}, Alberto Scionti[†], Olivier Terzo[†], Chiara Vercellino^{†¶}, Giacomo Vitali^{†¶}, Paolo Viviani[†], Jonathan Frassinetti[‡], Sara Marzella[‡], Daniele Ottaviani[‡], Iacopo Colonnelli^{**}, Daniele Gregori^{*}

^{*}*E4 Computer Engineering*, Scandiano, Italy [†]*LINKS Foundation*, Torino, Italy

^{††}*Barcelona Supercomputing Center (BSC-CNS)*, Barcelona, Spain [‡]*CINECA*, Casalecchio di Reno, Italy

[¶]*Politecnico di Torino*, Torino, Italy ^{**}*Università di Torino*, Torino, Italy

[§]roberto.rocco@e4company.com

Abstract—The integration of quantum computers within classical High-Performance Computing (HPC) infrastructures is receiving increasing attention, with the former expected to serve as accelerators for specific computational tasks. However, combining HPC and quantum computers presents significant technical challenges, including resource allocation. This paper presents a novel malleability-based approach, alongside a workflow-based strategy, to optimize resource utilization in hybrid HPC-quantum workloads. With both these approaches, we can release classical resources when computations are offloaded to the quantum computer and reallocate them once quantum processing is complete. Our experiments with a hybrid HPC-quantum use case show the benefits of dynamic allocation, highlighting the potential of those solutions.

Index Terms—HPC, Quantum Computing, Dynamic Resource Management, Malleability

I. INTRODUCTION

Quantum Computing (QC) technologies have evolved significantly, making quantum utility look attainable soon. As performance improves, the research community has begun to explore how quantum computers might fit into the broader computing continuum. Quantum systems are particularly well suited for addressing specific classes of NP-hard problems [1]. This feature makes them ideal candidates to complement traditional High-Performance Computing (HPC) systems. In this emerging paradigm, it is widely anticipated that quantum processing units (QPUs) will serve as accelerators for computationally intensive, exponentially scaling workloads within scientific applications [2]–[4], while HPC systems will provide the necessary scalability to support hybrid quantum-classical applications [5].

Bringing QC into HPC environments holds promise for boosting progress across many scientific domains. Beck *et al.* have recently identified a set of applications spanning various

scientific fields, which would highly benefit from an HPC-QC integration [3]. These applications include quantum many-body dynamics for simulating quantum systems, continuum mechanics simulations using quantum linear solvers for problems like fluid dynamics, quantum-enhanced machine learning for developing advanced models, and quantum optimization to tackle complex optimization problems. Early examples of hybrid HPC-QC algorithms have begun to appear in the literature. Vercellino *et al.* developed a graph coloring code relevant in several industrial contexts by combining a parallel branch-and-bound algorithm with a quantum routine that solves the maximum independent set problem to sample potential coloring solutions [6]. Similarly, Kim and Suh considerably enhanced the performance of their hybrid optimization algorithm for metamaterial design by parallelizing via Message Passing Interface (MPI) both the machine learning phase preceding and the wave-optics simulation phase following the quantum approximate optimization algorithm [7].

For all these reasons, a heterogeneous HPC-QC cluster is an appealing prospect. However, realizing such integration presents significant challenges due to the disparity in technological maturity between HPC and QC, their fundamentally different architectural paradigms, and the substantial imbalance in resource availability between classical and quantum systems [8]. Among these challenges, efficient resource management and allocation stand out as critical concerns [5], [8]–[10]. In particular, dedicated scheduling strategies are essential to prevent resource under-utilization, especially in scenarios where only one or two QPUs are available per cluster [11]–[13]. A recent study analyzed and discussed potential solutions for efficient resource allocation in HPC-QC integration, proposing malleability to minimize computational resource waste [14]. A malleable job is capable of dynamically adjusting its resource allocation at runtime. In the HPC-QC context, this would enable a job to release classical HPC resources during quantum computation phases and reallocate them once computation returns to the classical phase.

In this work, we explore for the first time the use of *malleability* in hybrid HPC-QC jobs to ensure efficient resource utilization. We also examine the potential of achieving

Funded by the European Union – NextGenerationEU: ICSC National Centre, CN00000013, MUR Act n. 1031 - 17/06/2022; and Barcelona Zettascale Lab, promoted by the Ministry for Digital Transformation and the Civil Service, within the framework of the Recovery, Transformation, and Resilience Plan. BSC researchers are also supported by the Dpt. of Research and Universities of the Government of Catalonia to the AccMem (Code: 2021 SGR 00807). Antonio J. Peña was partially supported by the Ramón y Cajal fellowship RYC2020-030054-I funded by MCIN/AEI/10.13039/501100011033 and by “ESF Investing in Your Future”.

effective resource management through a *workflow*-based approach. It is worth noting that, although the use of Workflow Management Systems (WMSs) has been proposed for some time [15], there are still few examples of hybrid HPC-QC workloads that leverage WMSs to optimize resource usage in such heterogeneous environments [16], [17]. As such, our use case implementation based on StreamFlow [18] introduces original elements and can help promote adopting this approach within the HPC-QC context.

To support this study, we parallelized an existing classical-quantum application to perform clustering aggregation of data leveraging different algorithms [19], [20]. The application provides the ideal use case to show the interaction between HPC and QC resources since the clustering methods are executed in parallel on HPC resources, and the aggregation occurs on the QC resource. We test our application on an *ad-hoc* cluster comprising three classical compute nodes, with an additional classical node functioning as a quantum emulator. We then extend the application to support malleability by integrating the Dynamic Management of Resources (DMR) framework [21]. Finally, we evaluate time-to-solution and resource usage across three scenarios: (1) no hybrid resource management; (2) hybrid resource management using a workflow-based approach implemented with the StreamFlow engine [18]; and (3) hybrid resource management with malleability.

Our experiments demonstrate that workflow-based and malleability-based approaches significantly reduce resource usage, effectively minimizing waste. Furthermore, we show that the malleability approach optimizes resource utilization in general and results in shorter time-to-solution than the workflow-based method. Notably, this temporal gain becomes increasingly pronounced as the computational load on the cluster grows.

Overall, the contributions of this paper are the following:

- We overview the design of hybrid HPC-QC applications starting from classical-quantum ones, evaluating different resource access schemes on a practical example;
- We apply a novel solution based on malleability to address the issue of resource allocation in an HPC-QC environment with limited QC resources;
- We compare our results across three test scenarios: without hybrid resource management, with workflow-based management, and with malleability.

This paper is structured as follows. Section II illustrates, through a concrete example, the transformation of a classical-quantum application into an HPC-QC workload, highlighting the benefits of integrating HPC and QC in real-world scenarios; Section III overviews the proposed approaches to deal with hybrid resource allocation; Section IV presents our experimental campaign, along with a discussion of the results. Finally, Section V concludes the article.

II. TOWARDS HPC-QC: CLUSTERING AGGREGATION

In recent years, a growing number of hybrid applications have emerged, combining classical computing and QC to

tackle complex scientific problems. Part of the computation is typically executed on a classical CPU, often single-threaded, while a quantum computer handles specific sub-tasks. Although QC is regarded as an extension of the frontier of HPC, few application examples in the literature take advantage of such a hybrid system. Building upon the works of Scotti *et al.* [20] and Li *et al.* [19], we propose a new hybrid HPC-QC application, and we later use it to evaluate our proposed resource management strategies. This algorithm performs clustering aggregation to determine the optimal number of clusters based on the output of multiple clustering methods. The application exhibits several key features that make it an attractive candidate for investigating dynamic quantum-HPC resource management: 1) the classical part is highly parallelizable and could effectively capitalize on parallel execution; 2) the application scales with a variable number of classical resources; and 3) it requires quantum resources only for a limited time. Thus, in this work, we adapt the clustering aggregation algorithm proposed in [20] to the HPC context. This adapted application serves as a use case to investigate the potential of a *malleability*-based approach and a *workflow*-oriented strategy for efficient resource allocation in hybrid classical-quantum environments.

Given a dataset, our C++ application, whose source code is publicly available at [22], begins by executing three widely used clustering algorithms: k-means [23], DBSCAN [24], and hierarchical clustering [25] (see Figure 1). Each algorithm instance is assigned to a separate HPC node using MPI, which enables parallel execution and results in significant performance improvements over the serial baseline, assuming multiple compute nodes are available. The outputs of the three clustering algorithms are then combined to construct an undirected graph, with the constraint that a valid clustering must consist of non-overlapping clusters, as described in [20]. In this formulation, each valid clustering corresponds to an independent set in the graph, and identifying a maximum independent set yields a unique and robust consensus clustering that mitigates the individual limitations of the underlying algorithms, as shown in [19]. Consequently, the clustering aggregation task reduces to solving a Maximum Independent Set (MIS) problem, which can be formulated as the minimization of the following function:

$$f_{MIS}(x) = - \sum_{i \in \mathcal{V}} w_i x_i + \lambda \sum_{(i,j) \in \mathcal{E}} x_i x_j. \quad (1)$$

where λ is the penalty factor, namely a corrective factor to dissuade the algorithm from choosing overlapping clusters, w_i are weights assigned to each cluster, and $x_i \in \{0, 1\} \forall i$. Notably, Equation (1) represents a QUBO problem.

In practice, the QUBO problem is formulated as a weighted adjacency matrix. The diagonal elements of this matrix contain weights w_i proportional to the dimension of the corresponding cluster, so that the final solution remains balanced in terms of cluster sizes, discouraging both massive and tiny clusters. The off-diagonal entries are non-zero only for pairs of clusters that

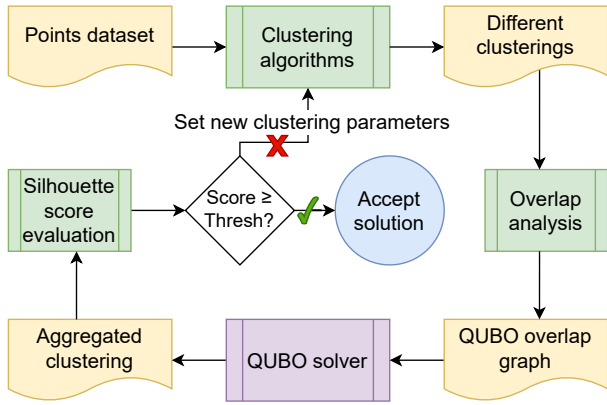


Fig. 1. Data and processes of the target application. Yellow shapes are data between processes, green boxes are classical processes, and purple boxes are processes suitable to be performed by quantum resources. The application loops until the Silhouette score of a certain clustering reaches a user-defined threshold or until the maximum number of loop iterations is reached.

share common data points (i.e., overlapping clusters), and are assigned a value equal to the number of distinct clusters λ .

The weighted adjacency matrix is the input to the quantum portion of our application (i.e., the *quantum phase*). A QUBO problem can be solved using various quantum computing architectures, including analogic quantum computers (e.g. Neutral Atoms, Quantum Annealers), and gate-based quantum computers, with the latter utilizing the Quantum Approximate Optimization Algorithm (QAOA). In this work, we solve it via Simulated Annealing (SA) [26]. After finding a solution to the QUBO problem, MPI rank 0 computes the silhouette score to assess the quality of the result. The silhouette value yields between -1 and 1 and stands as a metric for determining the similarity of an object to its assigned cluster (cohesion) versus other clusters (separation). A high silhouette value indicates that the object is well-suited to its cluster and distinct from neighboring clusters. By evaluating the silhouette score, we choose whether to repeat the algorithms with different configurations to aim for better quality or to stop the execution. In particular, we terminate the loop when we achieve a silhouette score greater than 0.8 . To prevent an infinite loop and excessive quantum resource usage, we also set an upper bound of 10 loop iterations, after which the best result obtained up to that point is chosen.

III. TOOLS FOR HYBRID RESOURCE ALLOCATION

The structure of the application flow defines two segments that must run on different hardware. From an HPC center perspective, providing access to multiple applications with this pattern is non-trivial, as quantum resources are currently scarce. The difficulty behind resource allocation comes from the imbalance between the requests for quantum resources and their availability, which inevitably introduces a bottleneck in the system. To ease the bottleneck, we must ensure that the users access scarce resources properly to avoid reserving them when unneeded, thus blocking the execution for all the others. In general, requesting both classical and quantum resources in

a SLURM-based fashion at the beginning of the execution and for its entire duration is inefficient, as the application uses only one type of resource at any time. On the other hand, reserving only classical resources and relying on a Representational State Transfer (REST) Application Programming Interface (API) to access the quantum node, as is commonly done in today's cloud-based quantum systems, is not efficient: while we wait to schedule the quantum offloaded portion, we are holding classical resources without using them.

The difficulty in maintaining efficiency comes from the discrepancy between the changing resource requirements of hybrid applications and the staticity of resource allocation. This problem is not new and already features some theorized solutions [3]–[5], [13]–[15]. Out of all the proposed approaches, we think that a *workflow* implementation of the execution flow and enabling dynamicity through *malleability* can yield significant benefits regarding efficient resource allocations. In the following subsections, we describe these two approaches, presenting some tools we leverage to implement those functionalities.

A. Workflow

A workflow approach divides an application into tasks and executes them according to the dependencies between them [27]. Describing complex, large-scale applications as workflows dramatically simplifies their transition from classical execution to hybrid HPC-QC settings. Various workflow engines have been developed to orchestrate the execution of applications across heterogeneous computing environments (e.g., Nextflow [28], PyCOMPSs [29], and Rigoletto [17]).

The StreamFlow WMS [18] has been explicitly designed with *hybrid workflows* in mind. A hybrid workflow can be expressed as (W, L, \mathcal{M}) , where $W = (S, P, \mathcal{D})$ is a directed bipartite graph representing a standard workflow (with steps S , ports P , and dependency links \mathcal{D}), L is the set of available execution locations, and $\mathcal{M} \subseteq (S \times L)$ is a mapping function [30]. This abstraction is general enough to encode workflow models specifically targeting classical-quantum applications. For example, in [16], workflow designers are required to identify a set $Q \subseteq S$ of *quantum candidates*, i.e., steps that support a quantum implementation that is *functionally equivalent* to the classical one [16]. At runtime, the WMS chooses between the classical and quantum implementation, depending on the availability of quantum resources. This technique can be encoded as a hybrid workflow by identifying a subset of *quantum locations* $L_Q \subseteq L$ and defining a *quantum mapping* function $\mathcal{M} \subseteq (Q \times L_Q)$. The principal advantage of this more general approach is that it allows a seamless transition from specialized HPC-QC orchestrators to X-QC settings, including cloud-QC and HPC-cloud-QC.

Implementation-wise, StreamFlow augments the Common Workflow Language (CWL) [31] open standard with a topology of deployment locations and a `LOOP` extension to model iterative workflows [32]. The orchestration plane leverages pluggable connectors to support several execution environments, from HPC queue managers to cloud infrastructures.

Developing a new plugin to support a given quantum device is just a matter of extending the StreamFlow Connector base class to integrate the device’s APIs, whether web-based REST APIs or SLURM-managed queues.

These features make StreamFlow a natural fit for implementing our HPC-QC application as a hybrid workflow. The resulting model is a three-step pipeline that computes and combines the clusterings (step s_C), aggregates them by solving a QUBO problem (step s_Q), and evaluates the resulting silhouette score (step s_S), s.t. $S=\{s_C, s_Q, s_S\}$. Each stage is implemented as a CWL CommandLineTool object, and the coordination logic is encoded as an iterative CWL workflow that repeats the whole process until the silhouette score reaches a configurable threshold. The execution environment is composed of three classical locations (HPC nodes) and one quantum location, s.t. $L=\{l_1^c, l_2^c, l_3^c, l_1^q\}$. The resulting mapping function is $\mathcal{M}(s_C)=\{l_1^c, l_2^c, l_3^c\}$, $\mathcal{M}(s_Q)=l_1^q$, and $\mathcal{M}(s_S)=l\in\{l_1^c, l_2^c, l_3^c\}$. Note that this example can also be modeled in the Cranganore *et al.* [16] framework by defining $Q=\{s_Q\}$ and $L_Q=\{l_1^q\}$.

B. Malleability

Malleability refers to the ability of a parallel application to dynamically adjust its [33] resource allocation, such as the number of computing nodes or processes, during runtime. This flexibility enables jobs to grow or shrink in response to changing system conditions, improving overall resource utilization and reducing wait times [34]. Malleability is an active area of research in traditional HPC. Among the several tools available to aid the implementation of malleability [35], the DMR framework [21] provides a high-level API designed to incorporate malleability into HPC applications seamlessly. DMR is a well-known tool in the community [36], has been thoroughly evaluated across a variety of use cases [37]–[39], and has also been extended with novel functionalities to support broader capabilities [40], [41]. DMR establishes communication between the Parallel Distributed Runtime (PDR), the Resource Management System (RMS), and the Parallel Performance Monitor (PPEM) to enable transparent process and resource management.

At its core, DMR enables passive dynamic resource management through the following process:

- During execution, DMR periodically checks when jobs are ready for reconfiguration at predefined synchronization points within the application.
- DMR determines whether a reconfiguration is necessary, considering available resources and performance metrics in collaboration with the RMS and the PPEM.
- DMR orchestrates the updates in resource allocations (interacting with the RMS) and the number of processes (through the PDR).
- Finally, the execution resumes seamlessly from the re-configuration point with the new job shape.

Listing 1 shows the structure of the implemented HPC-QC malleable code described in Section II. First, the DMR environment is initiated and provided with the program-specific

```

1 void main() {
2   MPI_Init();
3   DMR_INIT (restart(it0, quantum));
4   for (auto it = it0; it < ITERS; it++) {
5     if (!quantum) {
6       while (!resources && !timeout)
7         DMR_EXPAND (checkpoint(it, quantum));
8       //Execute Classical Computation
9       quantum = true;
10      DMR_MINIMIZE (checkpoint(it, quantum));
11    }
12    if (rank_id == 0)
13      //Offload Quantum Computation
14      quantum = false
15    }
16  MPI_Finalize();
17 }

```

Listing 1. Scheme of how malleability is adopted in the user code.

restart function, used by new processes after reconfigurations (line 3). The main loop (line 4) has two stages: classical and quantum. At the beginning of each iteration, if the execution has to run the classical stage, a reconfiguration can be scheduled (line 7). After this stage, only the root MPI process will be kept active, relinquishing the remaining resources (line 10). The reconfiguration requires an application-specific checkpoint function to save the current state, which comprises the current loop iteration number and a flag that specifies whether we can proceed to the quantum stage. This state is then restored by the restart function. Eventually, the execution continues in the quantum stage (line 13) without wasting classical resources.

IV. EXPERIMENTAL CAMPAIGN

We conducted a series of experiments by executing our hybrid clustering application under different configurations to evaluate the impact of the solutions described in Section III. In the following paragraphs, we present the experimental setup and overview of the campaign, analyzing its results.

Experimental setup. We set up a dedicated cluster to test our approaches, using SLURM version 23.02.7. The cluster represents a plausible HPC-QC integration scenario at scale, comprising two partitions: a log-in and a master node. The first partition, named *compute*, consists of three CPU-only nodes. Each compute node contains two AMD EPYC 7543 CPUs and 256 GB of DDR4 memory. The second partition, named *quantum*, acts as a quantum emulator. This node contains two AMD EPYC 7282 CPUs, 512 GB of DDR4 memory. However, due to the memory requirements of the clustering algorithms, we launch our application requesting one or more classical nodes and one task per node. The log-in node and the master node are virtualized x86_64 machines with 8 GB of memory each. Every machine in the described cluster runs on Red Hat Enterprise Linux 9.4, using kernel version 5.14. The application source code is compiled with GNU Compiler Collection version 11.4.1. The distribution of classical workloads across the compute partition is achieved with OpenMPI version 5.0.6. Submitting quantum jobs is carried out with HyperQueue [42] version 0.21.1. In particular,

HyperQueue allocates a node from the SLURM quantum partition and launches the QUBO solver on it. StreamFlow version 0.2.0.dev12 is used to validate the workflow approach. **Towards quantum.** In our pipeline, the quantum phase is formulated as a QUBO problem. We currently solve it using Simulated Annealing on a classical node, which mimics the behavior of a Quantum Annealing algorithm. Since the problem core formulation remains unchanged, substituting the classical node, initially a placeholder, with an actual quantum device requires no modifications to the proposed configurations or methodologies. With real quantum hardware, such as a superconducting quantum processor, a quantum annealer, a neutral atom device, or a photonic quantum computer, the QUBO problem can be tackled using inherently quantum approaches. It is essential to note that different quantum technologies exhibit distinct characteristics and execution times. For instance, a neutral atom device would typically require minutes to complete a job (also considering register preparation), whereas a superconducting device operates on a much faster timescale, in the order of seconds. To simulate the performance of a neutral atom device, we introduce an artificial delay of some minutes in those runs, allowing us to analyze and understand the potential benefits and limitations of dynamic resource allocation in such systems. The duration of these artificial delays is determined based on the quantum machines of interest, considering the typical preparation and execution times of their average quantum job.

A. Experimental Results

We execute a set of comparative experiments to evaluate the behavior of a system running hybrid HPC-QC workloads. We begin by analyzing a configuration that adopts a traditional resource management strategy, allocating classical resources for the entire duration of the hybrid job and offloading the quantum subroutine to the emulated QPU using HyperQueue. This configuration is referred to as the *baseline* throughout the rest of the paper, as it resembles the current offloading scheme diffused with cloud-based quantum machines. We then compare the baseline results with those of the two alternative approaches for hybrid resource management discussed in Section III: *workflow* and *malleability*. As for the baseline, the malleability offloads the quantum task via HyperQueue, whereas the workflow approach independently allocates a node from the quantum partition only when required. For all three configurations, we evaluate and compare the following metrics:

- Wall time: the time employed to execute the whole simulation, considering also the time spent by SLURM to initialize and finalize the job(s);
- Classical resource usage in terms of node-seconds: the sum of the product between the number of classical nodes used in an interval and the length (in seconds) of the interval (we disregard here the quantum resources because their usage is constant across the three scenarios);

The dataset used in the clustering application consists of 80,000 2D points generated via `make_blobs` from scikit-

learn¹. The application code is written to permit reproducible results. In particular, given the described dataset, each workload run ends after the fourth loop iteration, as the combined clusterings achieve a silhouette score over 0.8. We profile two versions of the application to better understand how different quantum technologies might suit specific scheduling approaches. A first version executes quantum jobs in a fraction of a second, thus akin to superconducting QPUs. A second version adds an artificial delay of two minutes during the quantum job, trying to mimic a generic neutral atom QPU.

We start from the executions with no resource contention, i.e. with no other jobs in the cluster queue, and with the two-minute-long quantum jobs, i.e. reproducing the behavior of a neutral atoms machine. We average the metrics from five runs for each strategy. Table I shows the result of the first experiment. The baseline approach is the fastest one, but it is less efficient regarding resource usage. The workflow approach performs poorly in terms of wall time since it asks SLURM for resources at every step, and the overhead of the WMS slows it down. Conversely, it is the best regarding resource usage with minimal node-second consumption. The malleability approach acts as a compromise between the other two. In the absence of resource contention, both malleability and workflow approaches primarily conserve valuable computational resources with a negligible impact on time-to-solution.

For our second experiment, we run two concurrent workloads using all the approaches, again under a queue empty from other submissions and by emulating two-minutes-long quantum jobs. Table I contains the results of this second experiment, averaged over five runs each. Note that the wall time here refers to the elapsed time between the beginning of the first starting workload and the completion of the latest ending workload, thus considering two complete end-to-end simulations. The baseline approach is, in this case, the worst-performing one. The other approaches can interleave their execution, finishing earlier and using fewer resources, as shown in Figure 2. The difference between malleability and workflow results resides in the need for the former to have at least one MPI process to remain active at all times, even during the quantum phase when computations are offloaded to the QPU. While this may seem like an overhead, it offers a clear advantage: when the code returns from the quantum phase, execution can resume immediately, allowing the simulation to proceed even if not all originally requested resources are available.

We then execute the experiments above without artificial delays in the quantum phase. The results of these new experiments can be seen in Table II. For baseline and workflow, the wall time and resource usage of the dual concurrent execution are almost double the values from the single execution. This means the classical resources are highly contested, with little space to optimize scheduling. The malleability solution, on the other hand, completes both simulations in a

¹https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html

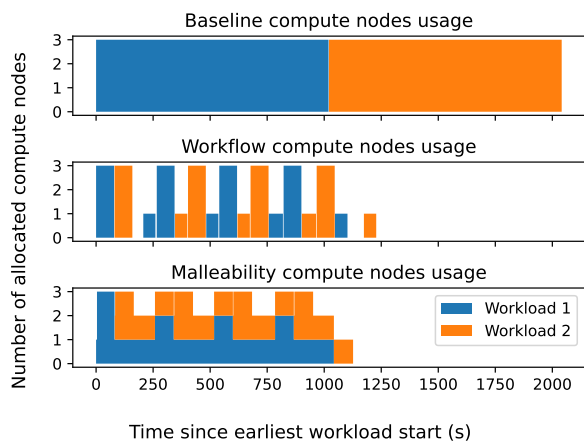


Fig. 2. Timeline of cumulative usage of classical nodes on our SLURM compute partition when launching two concurrent workloads (in blue and orange, respectively) with two minutes long quantum jobs. Each subplot refers to a different scheduling approach, i.e. baseline, workflow, and malleability.

TABLE I
EXECUTIONS WITH 2 MINUTES LONG QUANTUM JOBS.

Execution Type	Mode	Wall time [seconds]	Resource usage [node-seconds]
Baseline	Single	1019.58 \pm 0.85	3058.74 \pm 2.56
Workflow	Single	1057.80 \pm 6.02	1161.20 \pm 6.94
Malleability	Single	1029.06 \pm 1.54	1647.75 \pm 1.54
Baseline	Double	2038.43 \pm 0.96	6115.30 \pm 2.89
Workflow	Double	1226.00 \pm 1.58	2324.00 \pm 3.39
Malleability	Double	1127.65 \pm 1.18	2817.73 \pm 1.27

time comparable to that of a single execution (648.61 s vs 549.60 s), demonstrating its ability to manage and adapt to varying resource demands dynamically. However, this result comes from the intrinsic properties of the algorithms executed during the classical phase (with k-means taking significantly less time than the other two), so they do not represent a trend. In general, scenarios involving short quantum jobs and no resource contention should show limited resource savings when using either workflow or malleability approaches.

B. Discussion

Experimental data suggests that malleability and workflow are well-suited for hybrid classical-quantum workloads with long-running quantum phases. The specific choice between a workflow or a malleability approach depends on what a user values the most: if resource usage is crucial, workflows

TABLE II
EXECUTIONS WITH SHORT (< 1 SECOND) QUANTUM JOBS.

Execution Type	Mode	Wall time [seconds]	Resource usage [node-seconds]
Baseline	Single	539.44 \pm 0.53	1618.33 \pm 1.60
Workflow	Single	569.00 \pm 3.94	1148.00 \pm 1.87
Malleability	Single	549.60 \pm 1.86	1168.29 \pm 1.81
Baseline	Double	1076.98 \pm 1.79	3230.95 \pm 5.37
Workflow	Double	1089.00 \pm 1.00	2324.00 \pm 4.24
Malleability	Double	648.61 \pm 2.08	1622.63 \pm 1.05

allow for using resources very efficiently; when time to solution is more critical, a malleability approach could be more appropriate, especially in HPC clusters with long queue times. Thus, even if somewhat anticipated, these results highlight the relevance of this work as part of a still limited but growing set of practical efforts addressing hybrid HPC-QC scheduling (e.g., [43]–[45]). Furthermore, while our current metrics do not account for queue times, incorporating this factor would further highlight the advantage of malleable jobs in reducing queuing delays, enabling users to obtain results more quickly.

We note that neither of the proposed approaches is a drop-in replacement inside an application with fully static resource allocation. Workflows require the programmer to create modular applications instead of monolithic ones. This would likely improve the quality of the code, but the user would need to understand a dedicated workflow language to glue the application parts together. Conversely, we argue that malleability approaches have low entry barriers since the offered interface is user-friendly for programmers, but they have to manage a code base with more complex state management and execution with variable resources.

V. CONCLUSION

In this work, we analyze the resource allocation challenges in the emerging HPC-QC landscape and propose a novel solution based on malleability alongside a workflow-based approach. To our knowledge, malleability has not previously been explored in the context of HPC-QC, yet it holds significant potential for improving efficiency. We demonstrate the benefits of malleability and workflow strategies through a representative HPC-QC application, showing clear advantages over a traditional, statically allocated baseline. Our experiments show that, under resource contention, malleability reduces execution time by approximately $\approx 44\%$ compared to the baseline and by $\approx 8\%$ compared to the workflow approach. Regarding resource consumption, malleability achieves a $\approx 54\%$ reduction in node-seconds compared to the baseline while incurring a $\approx 17.5\%$ overhead compared to the workflow approach.

Managing a current HPC-QC cluster is not trivial, as the classical and quantum partition differences go beyond the simple programming model. As the maturity of the quantum software stack grows, the scientific community is also considering the issues coming from resource allocation. The present work contributes to this effort by leveraging the current state of the art in supercomputing systems to bridge the gap until more advanced, integrated frameworks become available. While we do not claim to offer a one-size-fits-all solution, we propose dynamic resource allocation strategies, both demonstrating performance improvements over the baseline.

Future work in this area includes extending this work by considering a larger-scale execution environment with a complex resource contention scenario, possibly addressing an actual quantum machine.

REFERENCES

- [1] R. Au-Yeung, N. Chancellor, and P. Halfmann, “Np-hard but no longer hard to solve? using quantum computing to tackle optimization problems,” *Frontiers in Quantum Science and Technology*, vol. 2, Feb. 2023.
- [2] M. Ruefenacht, B. G. Taketani *et al.*, “Bringing quantum acceleration to supercomputers,” *IQM/LRZ Technical Report*, https://www.quantum.lrz.de/fileadmin/QIC/Downloads/IQM_HPC-QC-Integration-Whitepaper.pdf, 2022.
- [3] T. Beck, A. Baroni *et al.*, “Integrating quantum computing resources into scientific hpc ecosystems,” *Future Generation Computer Systems*, vol. 161, pp. 11–25, 2024.
- [4] A. Elsharkawy, X. Guo, and M. Schulz, “Integration of quantum accelerators into hpc: Toward a unified quantum platform,” in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 01, 2024, pp. 774–783.
- [5] P. Mantha, F. J. Kiwit *et al.*, “Pilot-quantum: A quantum-hpc middleware for resource, workload and task management,” 2024.
- [6] C. Vercellino, G. Vitali *et al.*, “Bbq-mis: A parallel quantum algorithm for graph coloring problems,” in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 02, 2023, pp. 141–147.
- [7] S. Kim and I.-S. Suh, “Performance analysis of an optimization algorithm for metamaterial design on the integrated high-performance computing and quantum systems,” 2024.
- [8] A. Elsharkawy, X.-T. M. To *et al.*, “Challenges in hpcqc integration,” in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 02, 2023, pp. 405–406.
- [9] A. Shehata, P. Groszkowski *et al.*, “Building a software stack for quantum-hpc integration,” 2025.
- [10] P. Döbler and M. S. Jattana, “A survey on integrating quantum computers into high performance computing systems,” 2025.
- [11] M. Schulz, M. Ruefenacht *et al.*, “Accelerating HPC With Quantum Computing: It Is a Software Challenge Too,” *Computing in Science & Engineering*, vol. 24, no. 4, pp. 60–64, Jul. 2022.
- [12] K. A. Britt and T. S. Humble, “High-performance computing with quantum processing units,” *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, pp. 1–13, 2017.
- [13] N. Saurabh, S. Jha, and A. Luckow, “A Conceptual Architecture for a Quantum-HPC Middleware,” Aug. 2023.
- [14] P. Viviani, R. Rocco *et al.*, “Assessing the elephant in the room in scheduling for current hybrid hpc-qc clusters,” 2025.
- [15] R. Ferreira Da Silva, R. Badia *et al.*, “Workflows community summit 2022: A roadmap revolution,” Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States), Tech. Rep., 03 2023.
- [16] S. S. Cranganore, V. De Maio *et al.*, “Paving the way to hybrid quantum-classical scientific workflows,” *Future Generation Computer Systems*, vol. 158, pp. 346–366, 2024.
- [17] V. De Maio, D. Bork, and I. Brandic, “Rigoletto: A workflow definition language for hybrid quantum-classical scientific applications,” in *2024 26th International Conference on Business Informatics (CBI)*, 2024, pp. 40–49.
- [18] I. Colonnelli, B. Cantalupo *et al.*, “Streamflow: Cross-breeding cloud with HPC,” *IEEE Trans. Emerg. Top. Comput.*, vol. 9, no. 4, pp. 1723–1737, 2021.
- [19] N. Li and L. J. Latecki, “Clustering aggregation as maximum-weight independent set,” in *Neural Information Processing Systems*, 2012.
- [20] R. Scotti, G. Bettonte *et al.*, “A clustering aggregation algorithm on neutral-atoms and annealing quantum processors,” 2024.
- [21] S. Iserte, R. Mayo *et al.*, “DMRlib: Easy-coding and Efficient Resource Management for Job Malleability,” *IEEE Transactions on Computers*, vol. 70, pp. 1443–1457, Sep. 2020.
- [22] E. C. Engineering, “Hybrid classical-quantum clustering aggregation,” <https://github.com/E4-Computer-Engineering/clustering-mis>, 2025.
- [23] J. MacQueen, “Multivariate observations,” in *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, 1967, pp. 281–297.
- [24] M. Ester, H.-P. Kriegel *et al.*, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD’96. AAAI Press, 1996, p. 226–231.
- [25] D. Müllner, “fastcluster: Fast hierarchical, agglomerative clustering routines for randpython,” *Journal of Statistical Software*, vol. 53, no. 9, 2013.
- [26] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
- [27] F. Suter, T. Coleman *et al.*, “A terminology for scientific workflow systems,” *Future Generation Computer Systems*, vol. 174, p. 107974, 2026.
- [28] P. Di Tommaso, M. Chatzou *et al.*, “Nextflow enables reproducible computational workflows,” *Nature biotechnology*, vol. 35, no. 4, pp. 316–319, 2017.
- [29] E. Tejedor, Y. Becerra *et al.*, “Pycompss: Parallel computational workflows in python,” *The International Journal of High Performance Computing Applications*, vol. 31, no. 1, pp. 66–82, 2017.
- [30] I. Colonnelli, D. Medic *et al.*, “Introducing SWIRL: an intermediate representation language for scientific workflows,” in *Formal Methods - 26th International Symposium, FM 2024, Milan, Italy, September 9-13, 2024, Proceedings, Part I*, ser. Lecture Notes in Computer Science, vol. 14933. Springer, 2024, pp. 226–244.
- [31] M. R. Crusoe, S. Abeln *et al.*, “Methods included: Standardizing computational reuse and portability with the common workflow language,” *Communication of the ACM*, 2022.
- [32] I. Colonnelli, B. Casella *et al.*, “Federated learning meets HPC and cloud,” in *Astrophysics and Space Science Proceedings*, vol. 60. Springer, 2023, p. 193–199.
- [33] D. G. Feitelson, “Packing Schemes for Gang Scheduling,” in *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, ser. IPPS ’96. Berlin, Heidelberg: Springer-Verlag, Apr. 1996, pp. 89–110.
- [34] S. Iserte, I. Martín-Álvarez *et al.*, “Towards the democratization and standardization of dynamic resources with MPI spawning,” in *Proceedings of the International Conference on Parallel Processing & Applied Mathematics (PPAM). Best Paper Award*, 2024.
- [35] J. I. Aliaga, M. Castillo *et al.*, “A Survey on Malleability Solutions for High-Performance Distributed Computing,” *Applied Science*, vol. 12, pp. 1–32, May 2022.
- [36] A. Tarraf, M. Schreiber *et al.*, “Malleability in Modern HPC Systems: Current Experiences, Challenges, and Future Opportunities,” *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–14, Jun. 2024, conference Name: IEEE Transactions on Parallel and Distributed Systems.
- [37] S. Iserte and K. Rojek, “A Study of the Effect of Process Malleability in the Energy Efficiency on GPU-based Clusters,” *Journal of Supercomputing*, vol. 76, pp. 255–274, Oct. 2020.
- [38] S. Iserte, H. Martínez *et al.*, “Dynamic Reconfiguration of Non-iterative Scientific Applications: A Case Study with HPG-aligner,” *International Journal of High Performance Computing Application*, vol. 33, pp. 1–10, Aug. 2018.
- [39] S. Iserte, R. Mayo *et al.*, “DMR API: Improving Cluster Productivity by Turning Applications into Malleable,” *Parallel Computing*, vol. 78, pp. 54–66, Jul. 2018.
- [40] D. Huber, S. Iserte *et al.*, “Bridging the Gap Between Genericity and Programmability of Dynamic Resources in HPC,” in *ISC High Performance 2025 Research Paper Proceedings (40th International Conference)*, Jun. 2025, pp. 1–11.
- [41] S. Iserte, I. Martín-Álvarez *et al.*, “Resource optimization with MPI process malleability for dynamic workloads in HPC clusters,” *Future Generation Computer Systems*, p. 107949, Jun. 2025.
- [42] J. Beránek, A. Böhm *et al.*, “Hyperqueue: Efficient and ergonomic task graphs on hpc clusters,” *SoftwareX*, vol. 27, p. 101814, 2024.
- [43] R. Wille, L. Schmid *et al.*, “Qdmi - quantum device management interface: Hardware-software interface for the munich quantum software stack,” in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 02, 2024, pp. 573–574.
- [44] A. Esposito and U.-U. Haus, “Slurm heterogeneous jobs for hybrid classical-quantum workflows,” 2025.
- [45] M. Tejedor, B. Casas *et al.*, “Distributed quantum circuit cutting for hybrid quantum-classical high-performance computing,” 2025.