ORNL/TM-2024/3388

High Performance Computing and Quantum Computing Integration Framework Architecture and Requirements Document

Amir Shehata
Thomas Naughton
In-Saeng Suh

August 2024

ORNL/TM-2024/3388

National Center for Computational Sciences

HIGH PERFORMANCE COMPUTING AND QUANTUM COMPUTING INTEGRATION FRAMEWORK ARCHITECTURE AND REQUIREMENTS DOCUMENT

Amir Shehata
Thomas Naughton
In-Saeng Suh

August 2024

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831
managed by
UT-BATTELLE LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

Table of Contents

ABSTRACT 1
1. Quantum Motivation 2
    1.1 Terms & Definitions 2
    1.2 Quantum Advantage for Specific Algorithms 2
    1.3 Quantum Simulators 3
        1.3.1 Advantages of Quantum Simulators 3
2. Project Overview 3
    2.1 Terms & Definitions 4
    2.2 HPC/QC Integration Space 5
3. HPC/QC Integration Usage Patterns 5
4. HPC/QC Integration Models 6
    4.1 Single Quantum Computing Resource 6
    4.2 Per-Job Quantum Computing Resource 6
    4.3 Per-Process Quantum Computing Resource 7
5. HPC/QC Integration Framework Overview 8
    5.1 Terms & Definitions 9
    5.2 High-level System Diagram 12
    5.3 Quantum Operational Phases 12
        5.3.1 Resource Reservation 12
        5.3.2 Quantum Resource Pairing 14
        5.3.3 Quantum Task Execution 15
        5.3.4 Dynamic Simulation Environment 16
    5.4 HPC ecosystem Integration 18
6. Data Model 18
7. Plugin Architecture 19
    7.1 Application Integration 19
    7.2 Quantum Platform 20
        7.2.1 Standardized API 21
        7.2.2 Quantum Task Format 21
        7.2.3 Software Framework 21
    7.3 External Platform Integration 22
8. Justification 22
9. References 24
10. Requirements 25
    10.1 Requirement Metadata 25
        10.1.1 Requirement Description 25
        10.1.2 Project Acronym 25
        10.1.3 Requirement Category 25
    10.2 Resource Management Frontend 26
    10.3 Resource management backend 28
    10.4 Quantum tasks 31
    10.5 Quantum task manager 31
    10.6 Quantum platform manager 32
    10.7 Quantum runtime controller 34
    10.8 Plugin architecture 35

ABSTRACT

The HPC/QC Requirements Document presents a comprehensive framework for integrating High Performance Computing (HPC) and Quantum Computing (QC). The framework proposed in this document supports a hybrid quantum-classical computing model that provides application flexibility, and resource abstraction and standardization.

Key highlights

- Emphasis on the potential of quantum computing to revolutionize specific algorithms and applications.
- Advantages of quantum simulators in mitigating challenges related to limited qubit coherence times and error-prone computational platforms.
- Outlines various models which an HPC/QC Integration framework needs to support.
- Outlines the Integration of HPC and QC through a comprehensive framework, (a) including resource management, (b) dynamic simulation environment, and (c) intelligent resource allocation.
- The framework's plugin architecture, allowing for seamless integration of new quantum simulators and hardware as well as (d) Applications.
- Emphases on the standardization of access to quantum platforms and the uniform reporting of resources, enabling seamless integration and interaction within the HPC/QC environment.
- Intelligent resource allocation and dynamic simulation environment for optimizing quantum task execution within the broader HPC context.

The document positions the proposed framework as a strategic choice for organizations seeking to integrate quantum computing capabilities into their existing HPC infrastructure, with a focus on hybrid quantum-classical computing, resource abstraction, and intelligent resource allocation.

Quantum Framework (QFw)

- Aims to harness the potential of quantum computing through a generic approach for executing quantum tasks on various quantum platforms.
- Provides flexibility, allowing users to leverage any circuit composition framework like Pennylane or Qiskit while maintaining standardization in the form of a common text-based format for circuit description (e.g., QASM 2.0 [5], QIR [3,4].
- Offers an MPI-based [2] mechanism to manage communication with the quantum platform, facilitating seamless integration into existing HPC environments without requiring significant changes in the application programming paradigm.
- Provides mechanisms for reserving both HPC and quantum resources in a cohesive manner, ensuring successful job submission by identifying required quantum resource types and configuring underlying resources accordingly.

1. QUANTUM MOTIVATION

In recent years, the field of quantum computing has shown remarkable advancements, demonstrating its potential to revolutionize certain types of algorithms and applications. While quantum computing holds great promise for solving specific problems exponentially faster than classical computers, its widespread adoption for general computing remains a future prospect. In the foreseeable future, quantum computing is anticipated to coexist and collaborate with classical High-Performance Computing (HPC) environments to harness its unique advantages.

1.1 TERMS & DEFINITIONS

Term	**Definition**
High Performance Computing (HPC)	HPC involves the use of advanced computing systems and parallel processing techniques to efficiently handle complex computations and large datasets, enabling significantly faster and more powerful computational capabilities than traditional computing environments.
Quantum Computing (QC)	Quantum computing is a field of computing that leverages principles from quantum mechanics, such as superposition and entanglement, to perform computations using quantum bits (qubits).
Qubits	Unlike classical bits, which can be either 0 or 1, qubits can exist in multiple states simultaneously, enabling quantum computers to explore parallel possibilities and potentially solve certain problems more efficiently than classical computers.
Noisy Intermediate-Scale Quantum (NISQ)	Quantum computers with ~50 - ~100 qubits may be able to perform tasks that surpass the capabilities of today's classical digital computers, but noise in quantum gates will limit the size of quantum circuits that can be reliably executed.
Quantum Computer Simulator	a classical computer simulating the behavior of a quantum computer.
Quantum Platform/Quantum Resources	the underlying hardware or software technology used to implement a quantum computer or quantum simulator.

1.2 QUANTUM ADVANTAGE FOR SPECIFIC ALGORITHMS

Quantum computing excels in solving problems that involve complex mathematical operations, optimization challenges, and cryptographic tasks. Algorithms such as Shor's algorithm for integer factorization and Grover's algorithm for unstructured search showcase the quantum advantage by outperforming their classical counterparts in terms of efficiency and computational speed.

1.3 QUANTUM SIMULATORS

In the landscape of quantum computing Noisy Intermediate-Scale Quantum (NISQ) devices have emerged as powerful yet have limited qubit coherence times and are error-prone computational platforms. These devices, featuring a limited number of qubits, qubit coherency issues and susceptibility to environmental noise, present challenges in ensuring the correctness of quantum computations. As quantum algorithms become more intricate, the need for error correction becomes pronounced, introducing complexity and demanding sophisticated algorithms to maintain accuracy. Quantum simulators play a pivotal role in mitigating these challenges by providing a controlled and error-free environment for researchers to develop, test, and debug quantum algorithms.

1.3.1 Advantages of Quantum Simulators

1. **Error-Free Development:** Quantum simulators allow researchers to develop quantum algorithms in an error-free environment, free from the coherency issues and noise prevalent in NISQ devices. This facilitates a more straightforward and precise algorithmic development process.
2. **Testing and Debugging:** Simulators provide a platform for rigorous testing and debugging of quantum algorithms. Researchers can simulate various scenarios, inputs, and conditions to analyze the behavior of quantum circuits, identify potential errors, and refine their algorithms before transitioning to real quantum hardware.
3. **Algorithm Optimization:** Quantum simulators enable researchers to optimize algorithms without the constraints of error correction. This iterative process helps fine-tune quantum circuits, enhancing performance and paving the way for more efficient quantum algorithms.
4. **Preparing for Real Hardware Testing:** Quantum simulators serve as a crucial step in the development pipeline, allowing researchers to thoroughly prepare for the transition to real hardware. By addressing correctness issues in a simulated environment, researchers can strategically plan and optimize their algorithms before subjecting them to the inherent challenges of NISQ devices.
5. **Access and Availability:** While real quantum hardware may have limitations in accessibility, quantum simulators are widely available and easily accessible to researchers. This democratization of simulation resources empowers a broader community of scientists and developers to engage in quantum algorithm research.

2. PROJECT OVERVIEW

The integration of quantum computing into HPC environments represents a strategic approach for unlocking the potential of quantum algorithms while maintaining the reliability and versatility of classical computing. This overview sets the stage for exploring the detailed requirements and considerations essential for realizing the quantum advantage within the broader landscape of high-performance computing. Furthermore, Quantum Simulators play a vital role in the NISQ era by providing a controlled environment for quantum algorithm development, testing, and debugging. Their ability to circumvent errors and coherency issues allows researchers to refine and optimize quantum algorithms before deploying them on real, error-prone quantum hardware, contributing to the advancement of quantum computing capabilities.

Integrating quantum computing into HPC ecosystems creates a symbiotic relationship, where classical systems handle traditional tasks, and quantum processors address specific problems for which they are uniquely suited. A practical and efficient approach involves adopting a hybrid quantum-classical computing model. In this model, classical computers manage day-to-day computations, while quantum processors focus on solving problems where they demonstrate a clear advantage. Task-specific quantum algorithms can be seamlessly integrated into existing HPC workflows, allowing organizations to leverage the strengths of both classical and quantum computing paradigms.

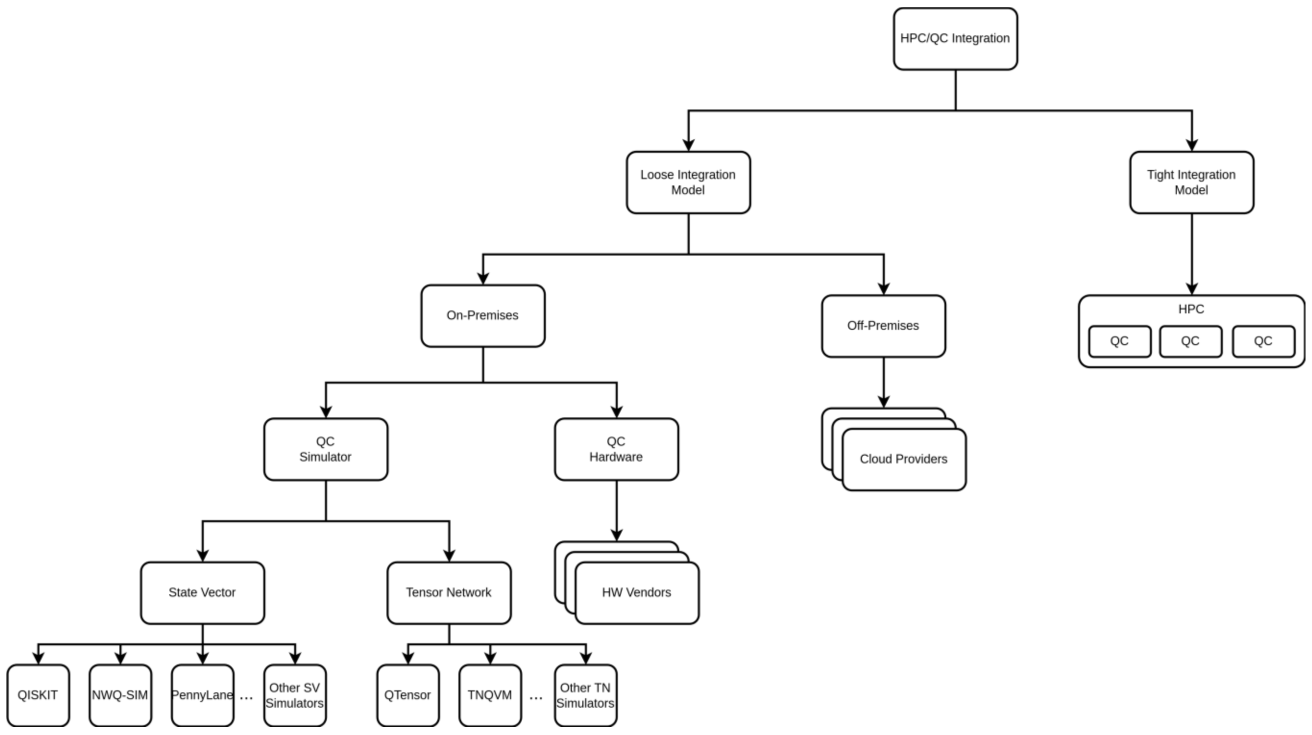
Given these considerations, it becomes imperative to establish a framework which:

- Facilitates the integration of Quantum Hardware into HPC environments
- Facilitates the integration of Quantum Simulators within the HPC environments.
- Expandable to allow for seamless integration of new simulators and hardware through the development of plugins tailored to the framework's requirements.
- Provides a layer of abstraction to the application, enabling it to access quantum platforms, whether hardware or software, in a standardized manner.
- Provides the application the ability to express resource requirements.
- Intelligently selects the available quantum resource based on application provided configuration.

2.1 TERMS & DEFINITIONS

Term	**Definition**
State Vector Simulator	A state vector Simulator in quantum computing is a simulation tool that represents the quantum state of a system using a state vector. It calculates the evolution of this vector over time, allowing researchers to model and analyze quantum algorithms and circuits.
Tensor Network Simulator	A tensor network Simulator in quantum computing is a simulation tool based on tensor network representations. It leverages tensor network diagrams to efficiently represent and compute the quantum states of complex systems. This approach is particularly useful for simulating large quantum systems where an explicit representation of the full state vector is impractical.

2.2 HPC/QC INTEGRATION SPACE



3. HPC/QC INTEGRATION USAGE PATTERNS

Hybrid HPC/QC applications typically involve classical logic intertwined with Quantum logic. Quantum Tasks (QT) will need to be built, compiled, optimized and run on either real quantum hardware or a quantum simulator; henceforth referred to as quantum platform or quantum resource. In some cases, they may be run multiple times to collect a probabilistic result. Results may or may not influence the following iterations. After the quantum portion is completed, classical logic may continue making use of the quantum results.

Application patterns can be summarized as follows:

1. **In-Sequence Processing:** Applications that require low-latency communication and control decisions over the quantum platform for successful execution of a quantum task. These applications demand individual outcomes calculated by the quantum processing unit (QPU) to be transmitted to the HPC and processed during the remaining run time of the quantum computation
2. **Single-Circuit:** Applications where the underlying quantum task remains static during execution and does not change its state based on intermediate measurement outcomes. These programs may be executed repeatedly to generate a distribution of measurements and a resulting statistical characterization, but such choices do not change the requirements placed on the control of the quantum platform. These circuits can be large and complex.
3. **Ensemble-Circuit:** Applications that require the execution of multiple circuit instances to generate a distribution of measurements and a resulting statistical characterization.

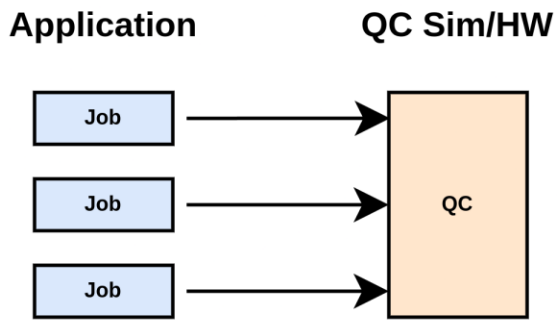
Circuit instances can be issued independently; however, aggregation of the results from all circuit instances is necessary for post-processing to complete the calculation.

4. HPC/QC INTEGRATION MODELS

To satisfy the hybrid HPC/QC Application usage patterns outlined above, the following integration models can be envisioned:

1. Single quantum computing resource
2. Per-Job quantum computing resource
3. Per-Process quantum computing resource

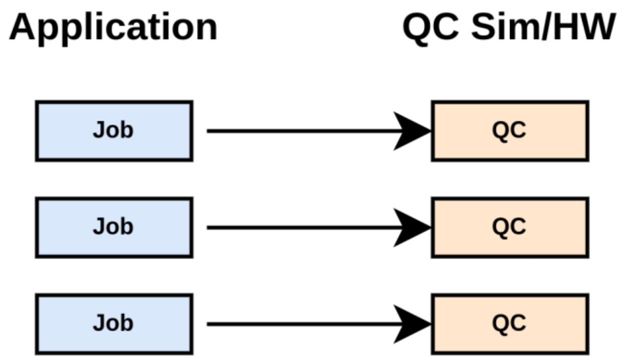
4.1 SINGLE QUANTUM COMPUTING RESOURCE



In this model a single quantum platform is used by all HPC jobs. This can be implemented by:

1. **Cloud resource model.** Quantum circuits are posted to remote QC resources. Cloud model presents several challenges, including latency and co-scheduling difficulties. Cloud resources are managed by external entities which cater to a variety of users and might not match local resource allocation policies.
2. **On-premises QC resource model.** Quantum circuits are posted to on-premises quantum platform.
3. **Long running QC Simulator model.** A long running QC simulator can be running on HPC and used by multiple jobs to perform Quantum computation

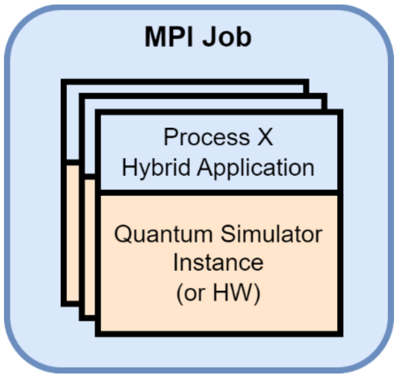
4.2 PER-JOB QUANTUM COMPUTING RESOURCE



In this model each job allocation is either paired with a quantum hardware resource or a set of HPC nodes dedicated for a quantum simulation environment.

- In this model each job will have a dedicated Quantum Resource.
- In the near term, this will not be feasible with real Quantum hardware due to availability.
- It is feasible to have a quantum simulator available for each job.
- A resource management system can allocate enough HPC nodes and partition them such that a subset is dedicated to the HPC job, and a subset is dedicated to the Quantum Simulator.
- It is possible to partition the nodes dedicated to Quantum Simulation, such that they run multiple Quantum Simulator instances.
    - Current Quantum Simulators parallelize single circuit execution. However, if multiple circuits are submitted for execution they are run sequentially.
    - For applications which use the "ensemble-circuit" pattern, they will benefit from running circuits in parallel. Therefore, having multiple Quantum Simulators running in parallel will be useful
    - This sub-version of the model can be extrapolated for the "Single quantum computing resource" simulator model described above. There is nothing preventing the HPC allocation dedicated for the quantum simulation to be partitioned among multiple simulators. Multiple circuits can be submitted and can run in parallel. In fact, this seems to be the preferred mode of operation.

4.3 PER-PROCESS QUANTUM COMPUTING RESOURCE



In this model each HPC node has a dedicated quantum resource. This parallels the existing approach where multiple MPI processes can run on a single node and have access to the integrated GPUs for acceleration. The per-process quantum computing resource model is not feasible in the near future with real quantum hardware, due to technology maturity. However, HPC/QC hybrid applications use this model via quantum simulators. The hybrid application steps include:

- An HPC allocation is reserved.
- The application runs on the allocation.
- Multiple MPI processes run on each node of the application.
- Each MPI process initializes its own quantum simulator when it starts up.

- Each MPI process uses a quantum framework (such as Pennylane or Qiskit) to build, compile, optimize and execute circuits.

This model benefits small circuit execution, as each node (or MPI process) has a dedicated simulator. However, there are several disadvantages to this approach:

- **Oversubscribe resources.** If there are multiple processes running on a single node, each process will spawn its own simulator. This may put undue strain on allocated computing resources.
- **Large circuit execution.** Large circuit execution may need a simulator running on multiple HPC nodes. In this model it can not be parallelized, since each simulator is restricted to one node
- **Compute resource usage.** Depending on the simulator being used, the compute resources on the node might not be used efficiently. For example, some simulators do not use GPUs.
- **Lack of circuit parallelization.** If an MPI process wants to run an ensemble of independent circuits, it can not run them in parallel due to simulator restriction. Each simulator can run one circuit at a time.

5. HPC/QC INTEGRATION FRAMEWORK OVERVIEW

The development of the Quantum framework (QFw) is aimed at harnessing the potential of quantum computing through a generic approach that can be utilized by hybrid HPC/QC applications for executing quantum tasks on various quantum platforms. QFw seeks to provide flexibility, allowing users to leverage any circuit composition framework like Pennylane [7] or Qiskit [6] while still maintaining standardization in the form of a common text-based format for circuit description (e.g., QASM 2.0, QIR). QFw offers an execution phase where built circuits are converted into this standardized format and handed off to QFw, which then executes them and returns results. This approach enables users to select any framework that suits their needs while still relying on a consistent circuit representation.

The framework offers an MPI-based mechanism to manage communication with the quantum platform. This approach facilitates seamless integration into existing HPC environments without requiring significant changes in the application programming paradigm. Additionally, the framework provides mechanisms for reserving both HPC and quantum resources in a cohesive manner, ensuring successful job submission by identifying required quantum resource types and configuring underlying resources accordingly.

In summary, the QFw shall abstract the underlying quantum platform, offering a cohesive approach for submitting quantum tasks and retrieving results. It shall feature a plugin architecture enabling the seamless integration of various quantum platforms while minimizing modifications to the application.

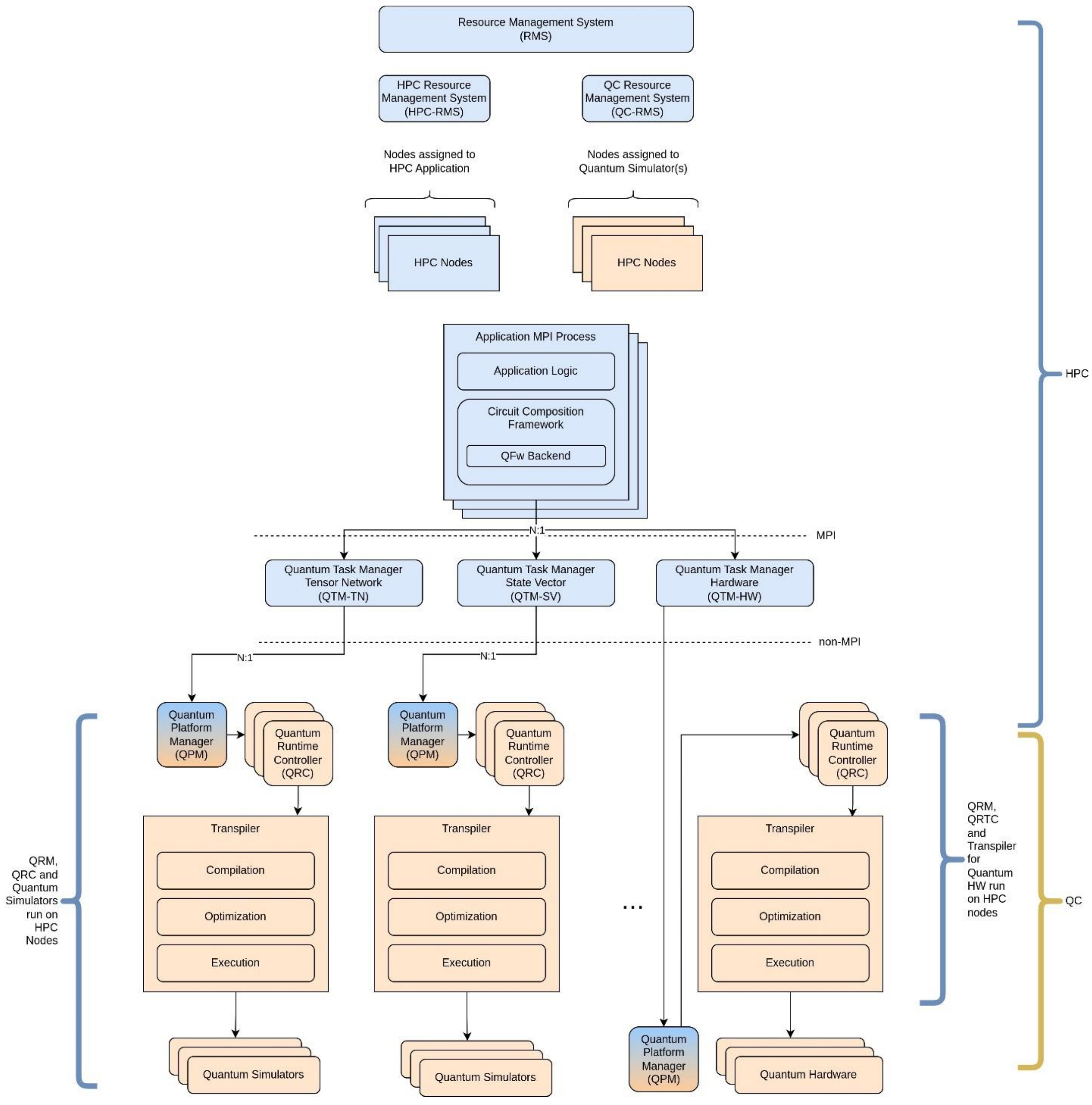
5.1 TERMS & DEFINITIONS

Term	Definition
Resource Management System (RMS)	Is an infrastructure component responsible for Allocating resources; both HPC and QC. SLURM is an implementation of this component.
Message passing Interface (MPI)	MPI is a standardized and portable message-passing system designed for parallel computing. It enables communication and coordination among processes in a parallel computing environment, allowing distributed memory systems to work collaboratively on a common task. MPI provides a set of functions, routines, and conventions for exchanging data and synchronization between processes in a parallel program. It is widely used in high-performance computing for building parallel applications that run on clusters, supercomputers, and other parallel architectures.
MPI Communication Group	A Communication Group refers to a set of processes that can communicate with each other. MPI allows the grouping of processes into subsets or groups, and communication operations, such as sending and receiving messages, are typically defined within these groups. Communication groups enable more flexible and efficient communication patterns among subsets of processes in parallel computing applications, allowing for targeted information exchange within specific subsets of the overall process ensemble.
Quantum Task (QT)	Is a task to run on a quantum platform. Currently, this is a quantum circuit.
HPC/Quantum Hybrid Framework (QFw)	The QFw, as detailed in this document, is designed to establish a standardized methodology for HPC/QC Applications to efficiently leverage Quantum Resources. These resources encompass both Quantum Hardware and Software Simulators. The primary goal is to enable the seamless execution of Quantum Tasks within a unified and consistent environment, promoting interoperability and ease of integration for applications relying on quantum computing capabilities.
HPC Resource Management System (HPC-RMS)	Is a subcomponent of RMS responsible for dealing with HPC resources.
QC Resource Management System (QC-RMS)	Is a subcomponent of RMS responsible for dealing with QC resources.
Quantum Task Manager - Tensor Network (QTM-TN)	Is an infrastructure component which handles Quantum Task submission to tensor network simulators.
Quantum Task Manager - State Vector (QTM-SV)	Is an infrastructure component which handles Quantum Task submission to state vector Simulators.
Quantum Task Manager - Hardware (QTM-HW)	Is an infrastructure component which handles Quantum Task submission to Quantum Hardware.
Quantum Platform	Is an infrastructure component which manages a specific quantum

Manager (QPM)	resource. It enforces the resource's management model, interfaces with its API, and acts as the gateway to all Quantum Task submissions.
Quantum Runtime Controller (QRC)	Is an infrastructure component responsible for preparing a QT to run on the QC resource.
Quantum Transpiler	A quantum transpiler is a tool or component in quantum computing that transforms a quantum algorithm or program written in a high-level circuit representation into a quantum resource specific instruction set. Its primary purpose is to optimize and adapt quantum code for execution on specific quantum hardware or simulator architectures, addressing differences in gate sets, connectivity, and other hardware-specific constraints. Transpilers play a crucial role in making quantum algorithms portable across different quantum processors and improving their performance on available hardware.
Quantum Volume	Quantum volume is a metric introduced by IBM to assess the overall capability and performance of a quantum computer. It is designed to provide a single, comprehensive figure that reflects the potential of a quantum processor, considering factors such as qubit quality, connectivity, and error rates. The Quantum volume takes into account various aspects of a quantum device: 1. **Number of qubits:** It considers the total number of qubits on the quantum processor. However, simply having more qubits doesn't necessarily imply better performance. 2. **Connectivity:** The quantum volume considers the connectivity between qubits. A higher degree of connectivity allows for more complex quantum circuits and, consequently, more powerful quantum computations. 3. **Error rates:** Quantum computers are prone to errors due to factors like noise and decoherence. Quantum Volume factors in the error rates of qubits and gates, aiming for a balance between qubit quantity and error mitigation.
Long Running Quantum Simulators (LRQS)	LRQS is a mode of quantum simulation where one single simulation resource can be shared by multiple jobs. Its lifetime is longer than the jobs it serves.
Quantum Configuration Parameters (QCF)	QCF refers to the user-provided parameters essential for assisting the RMS in accurately reserving the appropriate set of quantum resources.
Quantum Simulation Environment (QSE)	This refers to a set of HPC nodes and associated software modules and environment variables designated to run quantum simulators. This is determined at reservation time.
Quantum Simulation Environment Category Partition (QSECP)	This refers to a partition of the QSE with its associated software modules and environment variables designated to run a specific category of quantum simulators, e.g., tensor network simulators, state vector simulators. This is determined at reservation time.

Quantum Simulation Software Stack Partition (QSSSP)	This refers to a partition of the QSECP with its associated software modules and environment variables designated to run a specific quantum simulation software stack, e.g., TNQVM, QTENSOR, etc. This is determined at reservation time.
Quantum Simulation Instance Partition (QSIP)	Refers to a partition of the QSSSP with its associated software modules and environment variables running a specific instance of a quantum simulator. This is determined at runtime.

5.2 HIGH-LEVEL SYSTEM DIAGRAM


Resource Management System (RMS)
HPC Resource Management System (HPC-RMS)
QC Resource Management System (QC-RMS)
Nodes assigned to HPC Application
Nodes assigned to Quantum Simulator(s)
HPC Nodes
HPC Nodes
Application MPI Process
Application Logic
Circuit Composition Framework
QFw Backend
N:1
MPI
Quantum Task Manager Tensor Network (QTM-TN)
Quantum Task Manager State Vector (QTM-SV)
Quantum Task Manager Hardware (QTM-HW)
non-MPI
N:1
N:1
Quantum Platform Manager (QPM)
Quantum Runtime Controller (QRC)
Transpiler
Compilation
Optimization
Execution
Quantum Simulators
Quantum Hardware
...
QRM, QRC and Quantum Simulators run on HPC Nodes
QRM, QRTC and Transpiler for Quantum HW run on HPC nodes
HPC
QC

5.3 QUANTUM OPERATIONAL PHASES

5.3.1 Resource Reservation

The HPC/Quantum Framework (QFw) is designed for use by hybrid HPC/QC application and is intended to abstract away quantum resource details. QFw is designed to integrate with the application, allowing the application to interface with the framework in a seamless manner.

A modified Resource Management System (RMS) which will be created as part of this work will integrate with the QFw. The RMS is responsible for reserving both HPC and QC resources. While reserving HPC resources aligns with established practices supported by frameworks like SLURM [9], the reservation process for QC resources lacks such standardized support. To address this, a set of criteria for reserving QC resources will be detailed in the requirements section of this document. Assuming these criteria are established, the "QC Resource Management System (QC-RMS)" component in the system diagram above will employ them to reserve either Quantum Hardware or a dedicated set of nodes for running QC simulators.

5.3.1.1 Hardware Reservation

- For HW resources or Long Running Quantum Simulators (LRQS) each resource will need to run a "Quantum Platform Manager (QPM)" process, depicted in the system diagram above.
- The QPM will be responsible for:
    - communication with both compute nodes and the quantum resource
    - Providing a standardized API for submitting Quantum Tasks
    - Managing and configuring the underlying quantum resource.
        - Different QC resources can implement different resource sharing models; QPU-Based sharing model; Qubit-Based Sharing model or Shares-based sharing model.
        - The QPM is aware of its resource's reservation model and can respond to the QC-RMS reservation requests appropriately.
- The QPM will register with the QC-RMS. This will allow the QC-RMS to determine which QC resources are available for reservation.
- The QC-RMS will then proceed to reserve the available HW or LRQS.
- Upon reservation success, QPM will start a "Quantum Runtime Controller (QRC)" for each reserved portion of the resource.
    - e.g., for the qubit-based model a quantum platform can be subdivided into multiple portions. A QRC process will run for each of these portions.
- QFw will query the RMS to get information about the allocation; both the Quantum HW allocation and the HPC node allocation.
    - QFw will start a "Quantum Task Manager - Hardware (QTM-HW)" MPI process to track the Quantum HW reserved.
    - The QFw will pair the QPM with the QTM-HW; more details on this step below. This pairing allows execution of Quantum Tasks on the QC resource.
    - The QFw will start the Application on the nodes reserved for the HPC application.
- Once this infrastructure is in place, the application can start submitting QT.

5.3.1.2 Cloud Reservation

- Cloud quantum resource reservation theoretically align with local quantum hardware reservation but face challenges due to coscheduling difficulties and job execution latencies.
- Reserving both local HPC and cloud quantum resources requires synchronization, but abstracted cloud reservation policies may introduce significant delays.
- Quantum Task execution, even with both resources reserved, is subject to cloud provider queuing policies, with no assurance of completion within the HPC job's lifetime.
- Security concerns arising from connecting to an external system also pose a set of challenges which need to be resolved.
- Aside from security concerns, if a time slice is allocated on a cloud quantum resource with a dedicated execution time, then the QFw can handle it.
- However, given the listed challenges, this document will not explore solutions for the co-scheduling and execution latency issues inherit to cloud resources.

5.3.1.3 Simulator Reservation

- For SW simulator bring up, the criteria provided will be used to determine how many HPC nodes to reserve and the category of simulation required by the application, e.g, tensor network simulation or state vector simulation.
  - It's possible that an application might want different simulation categories simultaneously.
- QFw will query the RMS for the HPC node allocation and all Quantum Configuration Parameters (QCP).
- QFw will start the application on the nodes designated for the HPC/QC Hybrid application.
  - The exact breakdown of responsibility in the QFw will be explored in the High-Level Design.
- QFw will use the QCP to partition the simulation designated HPC nodes among the different simulation categories.
- QFw will designate a head node per simulation category partition and bring up a QPM process on each head node.
  - A head node is one of the nodes in allocation designated for running QFw services.
- The QPM in the simulation case is responsible for further partitioning the designated HPC nodes into Quantum Simulation Partitions.
- Each partition can run exactly one quantum simulator instance of the category assigned to it. This can be done dynamically.
- A head node is designated per partition, and it runs a Quantum Runtime Controller (QRC) process.
- The QRC process is responsible for transpiling a quantum circuit and executing it on its designated simulator.
- QFw will start the correct category of "Quantum Task Manager" MPI process to track the Quantum Simulator.
- The QFw will pair the QPM with the QTM.
- Once this infrastructure is in place, the application can start submitting QT.

5.3.2 Quantum Resource Pairing

- Different categories of Quantum Task Managers (QTM) run as MPI processes.
- Each category has one QTMs running as depicted in the diagram above.
- Each QTM would be responsible for a group of similar QC resources.
    - For example, if we have both TNQVM [10] and QTensor [11] simulators, we would have one QTM for each. The QTM can manage multiple QC resource of the same type.
- The QTM's primary role is to provide a standard MPI interface for an application to submit Quantum Tasks.
    - Since they run as an MPI process, the standard MPI communication patterns can be used to submit one or many Quantum Tasks for execution and then wait for their completion.
- On the backend each QTM can handle one or more quantum resources of the same category.
- This assignment of QPM to QTM is done by the QFw on QFw initialization.
    - For example, if the QFw (through querying the QC-RMS) was instructed to start a QTNVM and a QTensor simulators, it would
        - partition the Quantum Simulator HPC node allocation into two.
        - Designate a head node for each of the partitions
        - Start a QPM on each of the head nodes
        - Start one MPI QTM-TN process
        - Provide the QTM-TN information to the QPMs
        - QPMs would register with the QTM-TN.
        - QTM-TN can subsequently assign tasks as appropriate to the QPMs
        - QTM-TN manages the task execution on the separate simulators depending on the submitted Quantum Task meta data

5.3.3 Quantum Task Execution

- Once the resource allocation and initialization of the QFw is complete, the infrastructure is in place for the application to exercise the QC resources through standard MPI message patterns.
- On the back end running QPMs have registered with their assigned QTM before QFw initialization can be completed.
- The Application may use any circuit composition framework, such as Qiskit or PennyLane to create a quantum circuit.
- The QFw will provide plugins for supported circuit composition framework to translate the framework specific circuit object into the agreed upon standardized format (e.g., QASM 2.0, QIR).
- The QFw will provide an API for the application to build a quantum task using the generated circuit.
- The Application can then query the QFw for available quantum resources, in a specific category.
- The QFw will return one QTM MPI handle to the application indicating the quantum resource(s) which matches the Application's query criteria.
- The application can then send the QT to the provided QTM using MPI_Isend() or similar APIs.

- The application can then wait on the completion of these QTs using standard MPI patterns, e.g., MPI_Waitall().
- The QTM can manage the usage of the underlying resources via some selection criteria. In its simplest form, it can round robin over all the resources it knows about.
- When a QTM receives a QT, it invokes the public QPM APIs to execute that QT.
- The QT is queued on the QPM Task Queue and scheduled for execution
- Once the QT completes a Quantum Result needs to be returned in a canonical format to the QTM.
- The QFw will provide a utility library to build the Quantum Result
- The QFw will provide plugins for supported circuit composition framework to translate the canonical quantum result format back into the circuit composition framework specific format.
    - The point is to minimize the amount of changes an application needs to make in order to integrate with the QFw

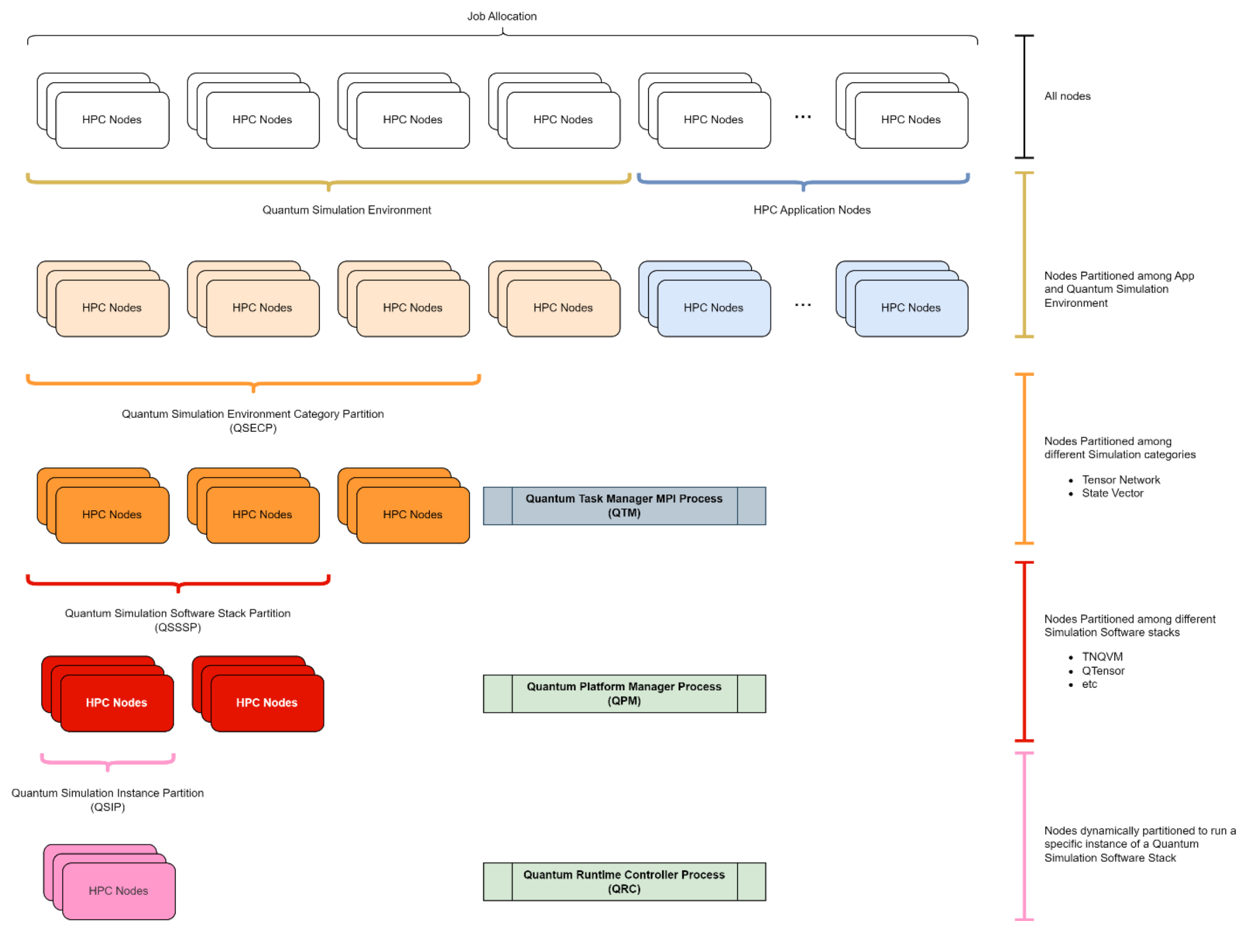
5.3.4 Dynamic Simulation Environment

In this section we introduce the concept of a simulation environment. Its purpose is to manage different classical quantum simulators instances running in parallel. A simulation environment can be partitioned per simulator category, if necessary, to ensure that resources can be used evenly. The partitioning of resources for use by the simulation environment are specified by the user for the job. The configuration rules are instantiated at runtime during the parallel job launch (i.e., application and simulation environment). The appropriate simulation environment is chosen at runtime based on user specified selection criteria and circuit metadata.

A quantum simulation environment needs to be flexible such that it can handle different quantum task work loads. As illustrated in the below diagram, once a job allocation is granted it will be partitioned as follows:

1. HPC Application, and
2. Quantum Simulation Environment (divided into 3 levels):

    - **Level-1:** *(Determined at reservation time)* One or more **Quantum Simulation Environment Category Partitions (QSECP):** These partitions can fall in the simulator type category, e.g., tensor network category, state vector category.
        - **Level-2:** *(Determined at reservation time)* One or more **Quantum Simulation Software Stack Partitions (QSSSP):** Each simulation stack will have its own partition.
            - **Level-3:** *(Determined at runtime)* One or more **Quantum Simulation Instance (QSI):** There could be multiple instances of the simulation stack running on each QSSSP.


Job Allocation
HPC Nodes
All nodes
Quantum Simulation Environment
HPC Application Nodes
Nodes Partitioned among App and Quantum Simulation Environment
Quantum Simulation Environment Category Partition (QSECP)
Nodes Partitioned among different Simulation categories
Tensor Network
State Vector
Quantum Task Manager MPI Process (QTM)
Quantum Simulation Software Stack Partition (QSSSP)
Nodes Partitioned among different Simulation Software stacks
TNQVM
QTensor
etc
Quantum Platform Manager Process (QPM)
Quantum Simulation Instance Partition (QSIP)
Nodes dynamically partitioned to run a specific instance of a Quantum Simulation Software Stack
Quantum Runtime Controller Process (QRC)

There are two primary **Quantum Task models** to consider:

1. **Ensemble circuit model:** this model represents applications which want to execute many independent circuits in parallel.
2. **Single large circuit model:** This model represents applications which want to execute one single large circuit

The **ensemble circuit model** would benefit from multiple simulators running in parallel, each executing a single independent circuit. The **single large circuit model** would benefit from one simulator running on multiple nodes which would parallelize the execution of the large circuit. A single application could potentially exercise both models at the same run. Therefore, having a dynamic simulation environment that adapts to the workload in the pipeline would be ideal. Each QPM managing a simulated environment has a set of nodes it knows about. It has a queue of Quantum Tasks. It can look ahead in the queue and using Quantum Task metadata can determine how the Simulation environment can be configured to most efficiently execute the queued tasks.

The following example can be used as an illustration for dynamic resource management.

- **INPUT:**
  - QPM manages a simulation environment consisting of:
    - 5 nodes.
    - 30 Quantum Tasks in its queue.
    - 4 qubits per task.
- **DECISION:**
  - Run 5 quantum simulators one on each node is best.
- **ACTION:**
  - start a quantum simulator and an associated QRC on each node.
  - Using round robin, it will assign six circuits per quantum simulator for execution.
  - Each circuit submitted by the QTM has a Circuit ID (CID). The QPM will use the CID to track each circuit execution.
- **OUTPUT:**
  - QPM reports back a completion event to the QTM using the CID.

5.4 HPC ECOSYSTEM INTEGRATION

The integration of a Quantum Computing Resource Management System (QC-RMS) with existing Resource Management Systems (RMS) like SLURM is crucial for establishing a cohesive and efficient allocation of both HPC and quantum resources. SLURM, as a widely used RMS, excels in managing classical computing tasks and orchestrating the allocation of HPC resources. The QC-RMS must seamlessly align with SLURM's capabilities to create a unified environment where both classical and quantum computing resources can be cohesively managed. This integration involves extending SLURM's functionality to incorporate quantum-specific resource allocation, ensuring a streamlined and consistent experience for users and administrators.

To achieve this integration, the QC-RMS should introduce a quantum-aware scheduler that comprehensively understands the intricacies of quantum computing tasks and their unique resource requirements. This scheduler will interface with SLURM, allowing users to submit hybrid quantum-classical jobs seamlessly. It becomes imperative to extend SLURM's job definition to include quantum-specific parameters, such as the type of quantum hardware or simulator required, quantum circuit details, and expected parallel circuits to run. The QC-RMS should also introduce a mechanism to reserve quantum resources alongside classical ones, ensuring that both are available concurrently for hybrid jobs. By harmonizing the functionalities of SLURM with quantum-specific requirements, the integrated system will empower users to leverage the full potential of hybrid computing environments, ultimately enhancing the overall efficiency of computational workflows.

6. DATA MODEL

Hybrid HPC/QC applications typically function by processing datasets through classical logic on the HPC platform. This processed data is utilized to generate Quantum Tasks (QT) destined for execution on a quantum platform. The outcomes obtained from these quantum tasks serve as inputs for generating additional QTs or for further processing on the classical side. While the initial datasets for processing may be substantial, leveraging HPC for handling large datasets is a well-established practice, and no specific requirements unique to this context arise. However, the transportation of QTs to the quantum platform poses a challenge, especially when the quantum platform is not directly connected to the HPC platform.

Practically, the Quantum Platform Manager (QPM) may consist of two distinct components. The first component operates on the HPC platform, acting as a proxy, while the second component runs on the classical segment of the quantum platform. This second component interfaces directly with the quantum platform's provided API. It is noteworthy that QTs and their results are generally small, typically a few kilobytes in size. Consequently, the network throughput required for transferring QTs and their results is not extensive and is not anticipated to be a bottleneck.

7. PLUGIN ARCHITECTURE

The QFw plugin architecture is intended to cover the following aspects.

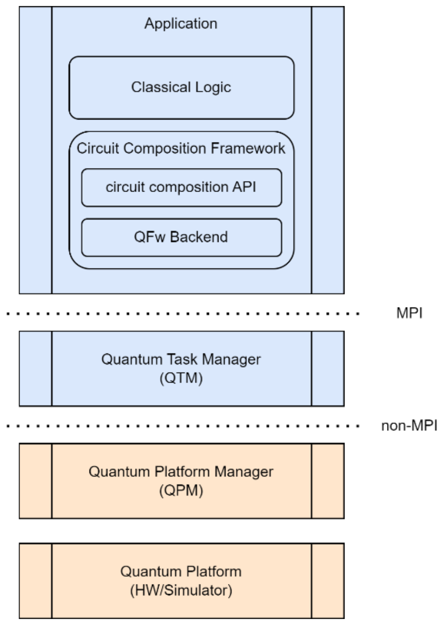
1. Application integration into the QFw
2. Quantum platforms integration into the QFw
3. External Platform Integration

7.1 APPLICATION INTEGRATION

The landscape of quantum simulation is rich with many frameworks which provide a Python interface to ease the composition and transpiling of quantum circuits. Each platform has its strengths. For example, PennyLane is designed to aid in natural language processing. Applications might prefer to use one over the other.

The aim of the QFw is to allow applications to use whatever circuit composition framework they need. The QFw will support the most used frameworks like Qiskit and Pennylane, by providing a backend plugin for these frameworks. These frameworks themselves have a plugin architecture that allows developers to specify the backend they would like their circuit to execute on. The QFw will provide a new backend for these frameworks, which should make QFw appear like yet another quantum platform. This allows applications to specify the QFw backend. The rest of the application's code will not change. When the application executes the circuit built by their preferred framework, the QFw backend will be invoked.

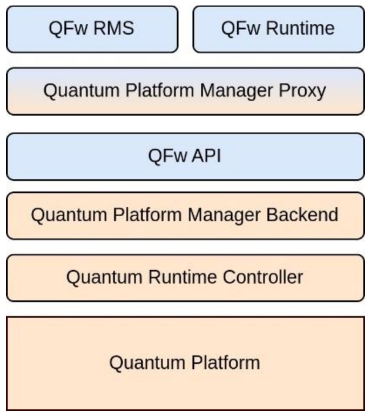
- A QFw Quantum Task will be built and send to the QTM.
- The QTM will post this QT on the QPM.
- The QPM will execute the QT on the quantum platform.
- Once the QT has completed the QPM will return the result back to the QTM.
- The QTM will return the quantum result back to the QFw Backend plugin of the circuit composition framework.
- The QFw backend plugin will do the necessary conversion into circuit composition framework specific format.



7.2 QUANTUM PLATFORM

The other aspect of the plugin architecture is the ability to add new quantum platforms for applications to use without the need to change the application code or the framework. The QFw shall achieve this goal by:

1. Standardizing an interface API which the quantum platform needs to implement in order to integrate into the QFw.
2. Defining a quantum task and quantum result format which is submitted and received respectively to and from the QPM.
3. Supplying a software framework which allows quantum platform providers to write plugins which adhere to the standardized API.


QFw RMS
QFw Runtime
Quantum Platform Manager Proxy
QFw API
Quantum Platform Manager Backend
Quantum Runtime Controller
Quantum Platform

7.2.1 Standardized API

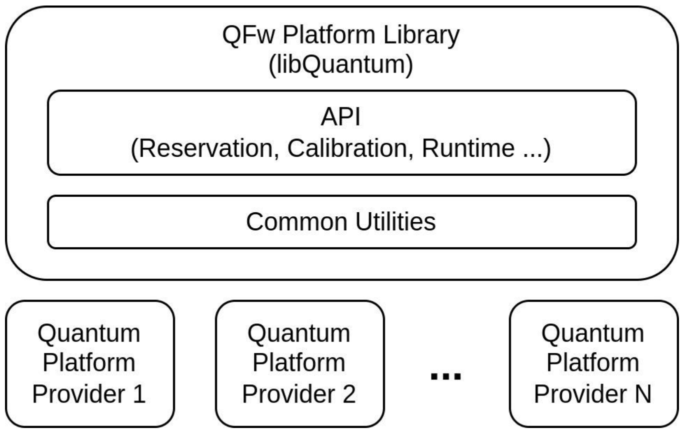
1. **Reservation API:** API used to reserve the underlying platform.
2. **A standard reservation description specification:** Specification adopted such that all platforms report their resources in a uniform manner. Example: Hardware location (hwloc).
3. **Resource query API:** API used to query details regarding the reservation model and availability of the platform.
4. **Quantum task submission API:** API used to submit quantum task for execution.
5. **Quantum task query API:** API used to query the quantum task execution status.
6. **Quantum task result API:** API used to form the quantum task results in a uniform format.
7. **Quantum platform configuration API:** API used to configure platform specific parameters.

7.2.2 Quantum Task Format

1. The QFw shall use the standardized circuit format to describe the QT to execute.
2. The QFw shall define a set of QT metadata information to pass along with the QT.
3. The QFw shall define a message format to include all the data pertaining to a QT.

7.2.3 Software Framework

To streamline the standardization process, the Quantum Framework (QFw) will introduce a software library structured akin to the functionality of libfabric [1]. This library serves as a central component for seamless integration with various quantum platforms.


QFw Platform Library
(libQuantum)
API
(Reservation, Calibration, Runtime ...)
Common Utilities
Quantum Platform Provider 1
Quantum Platform Provider 2
...
Quantum Platform Provider N

The proposed structure of the library encompasses:

1. **User Facing API:**
    - ***Purpose:*** Providing a common interface for applications and middleware to interact with the underlying quantum platform.

    - ***Functionality:*** Enabling a unified set of calls that applications and middleware can utilize to communicate with the quantum platform.
2. **Common Utilities:**
    - ***Purpose:*** Offering a collection of standardized utilities to assist platform providers in harnessing shared functionalities.
    - ***Examples:*** Provision of request queues, completion queues, and other essential components that enhance overall efficiency.
3. **Provider Plugin:**
    - ***Purpose:*** Allows each platform provider to develop a plugin responsible for implementing either the entire or a subset of the user-facing API.
    - ***Functionality:*** Allowing platform providers to tailor their plugins to suit specific requirements, ensuring flexibility and adaptability.

For the success of this approach, a consensus on the set of APIs must be achieved. This necessitates collaboration among industry platform providers. Notably, this collaborative effort involves establishing a working group, comprising industry leaders such as IBM, to deliberate and reach a consensus on the essential APIs. This inclusive process ensures that the finalized set of APIs is agreeable to all stakeholders, facilitating smooth integration and fostering a standardized approach across diverse quantum platforms.

7.3 EXTERNAL PLATFORM INTEGRATION

It is inevitable that the QFw will need to integrate a quantum platform which exists on a different network than the HPC cluster, either internal to the organization or managed by an external organization. It is not feasible for the QFw to present a standard way to connect with all potential external platforms. The challenge is not only technical but due to policies of both the local and external organizations.

ORNL is working on methods to allow external organizations to programmatically access local ORNL resources, such as frontier. These methods will be used to transfer data in and out of the HPC resources. The best approach to handle external platform integration is for the QFw to leverage work already done at ORNL in this area.

The QFw can provide utility functions which allow QPMs to open ports and manage the security with the external quantum platform. It is likely these operations will be unique for each external quantum platform, especially if it's managed by a different organization. It might be possible to standardize access from the ORNL side, but since the work to allow external access into ORNL OLCF machines is still ongoing, no approach can be suggested at the time of this writing. The best approach is to have an amendment to this document, when a clear design can be envisioned.

8. JUSTIFICATION

Existing frameworks like XACC [8], Qiskit [6] and PennyLane [7] provide a plugin architecture, which allows different quantum platforms to be added. The proposed quantum framework distinguishes itself from these existing frameworks by focusing on the seamless integration of quantum computing within HPC environments. While Qiskit and PennyLane are excellent tools for quantum algorithm development and simulation, they primarily cater to standalone quantum computing applications. In contrast, our framework is specifically tailored to foster a symbiotic relationship between quantum processors and classical HPC systems.

Key differentiators are:

- **Holistic Integration with HPC:** The QFw is designed to be deeply integrated into HPC ecosystems. It provides a cohesive approach for organizations looking to harness the complementary strengths of classical and quantum computing.
- **Hybrid Quantum-Classical Computing Model:** The framework advocates for a practical and efficient hybrid quantum-classical computing model. This model seamlessly incorporates quantum algorithms into existing HPC workflows.
- **Application Flexibility:** QFw doesn't lock the application into using a specific circuit composition framework. Applications can use any they deem fit; e.g., Qiskit, Pennylane, etc. QFw offers a backend to these frameworks.
- **Resource Abstraction and Standardization:** One of the framework's key features is the provision of a layer of abstraction to the application. This abstraction enables standardized access to quantum platforms, whether they are hardware or software simulators. The framework empowers applications to express their resource requirements intelligently.
- **Expandability and Plugin Architecture:** The QFw is designed to be highly adaptable, allowing for the seamless integration of new quantum simulators and hardware. The development of plugins tailored to the framework's requirements facilitates the incorporation of emerging technologies without requiring changes to the existing QFw architecture.
- **Intelligent Resource Allocation:** The framework goes beyond providing a mere interface and actively assists in resource management. It intelligently selects the most suitable quantum resource based on the configuration provided by the application, optimizing the execution of quantum tasks within the broader HPC context.
- **Dynamic Simulation Environment:** The framework provides a method to schedule quantum task simulation in the most efficient way possible on a set of HPC nodes.

In summary, QFw positions itself as a comprehensive solution for organizations seeking to integrate quantum computing capabilities into their existing HPC infrastructure. The emphasis on hybrid quantum-classical computing, resource abstraction, expandability, intelligent and dynamic resource allocation distinguishes it as a strategic choice for harnessing the power of both classical and quantum paradigms within a unified framework.

9. REFERENCES

[1] P. Grun, S. Hefty, S. Sur, D. Goodall, R.D. Russell, H. Pritchard, and J. Squyres, *A Brief Introduction to the OpenFabrics Interfaces – A New Network API for Maximixing High Performance Application Efficiency*, Proc. of Annual Symposium on High-Performance Interconnects (HotI), 2015, IEEE.

[2] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard Version 4.1*, November 2023. URL: https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf

[3] T. Nguyen, A. McCaskey, *Enabling Retargetable Optimizing Compilers for Quantum Accelerators via a Multi-level Intermediate Representation*, arXiv:2109.00506, 2021.

[4] QIR Alliance, *Quantum Intermediate Representation (QIR): The core of quantum development*, URL: https://www.qir-alliance.org. [Last visited: 27-aug-2024]

[5] A.W. Cross, L. Bishop, J. Smolin, and J. Gambetta, *Open Quantum Assembly Language*, arXiv:1707.03429, 2017.

[6] A. Javadi-Abhari, M. Treinish, K. Krsulich, C.J. Wood, J. Lishman, J. Gacon, S. Martiel, P.D. Nation, L.S. Bishop, A.W. Cross, B.R. Johnson, J.M. Gambetta, *Quantum computing with Qiskit*, arXiv:2405.08810, 2024.

[7] V. Bergholm et al., PennyLane: Automatic differentiation of hybrid quantum-classical computations, arXiv:1811.04968, 2018.

[8] A.J. McCaskey, D.I. Lyakh, E.F. Dumitrescu, S.S. Powers, and T.S. Humble, *XACC: A system-level software infrastructure for heterogeneous quantum-classical computing*, Quantum Science and Technology, Vol. 5 No. 2, Feb. 2020.

[9] SchedMD, SLURM: Hetergogenous job support, 2024. https://slurm.schedmd.com/heterogeneous_jobs.html [Last viewed: 27-aug-2024]

[10] T. Nguyen, D.I. Lyakh, E.F. Dumitrescu, D. Clark J. Larkin, and A.J. McCaskey, *Tensor network quantum virtual machine for simulating quantum circuits at exascale*, ACM Trans. on Quantum Computing, Vol. 4, pp.1-21, 2021.

[11] D. Lykov, R. Schutski, A. Galda, V. Vinokur, Y. Alexeev, *Tensor Network Quantum Simulator with Step-dependent Parallelization*, arXiv:2012.02430, 2022.

10. REQUIREMENTS

10.1 REQUIREMENT METADATA

10.1.1 Requirement Description

Requirement Priority (RP)	• Demo (D): Must be implemented for the feasibility of the demo • Mandatory (M): Must be implemented for the project to be declared complete • Optional (O): Need to be implemented to make the project stand out • Nice to have (N): Only implement if time permits In cases where a requirement is assigned multiple priorities, it may undergo partial implementation at the primary priority, while other components may be addressed at the secondary priority.
Requirement Description (RD)	Description of the requirement.
Requirement ID (RID)	Unique requirement ID to identify a specific requirement. Can be used to associate an HLD or Test in a test plan with the intended requirement Each Requirement ID is divided as follows <PROJECT>_<CATEGORY>_<NUMBER> • <PROJECT> is the project ID. This document will use QCI (Quantum Computing Integration) • <CATEGORY> is the requirement category. The categories are outlined in a later section • <NUMBER> unique requirement number

10.1.2 Project Acronym

Acronym	**Meaning**
QCI	Quantum Computing Integration

10.1.3 Requirement Category

Acronym	**Meaning**	**Description**
QCP	Quantum Configuration Parameters	Requirements for configuration parameters required for quantum resource reservation.
RMB	Resource Management Backend	Requirement for the the quantum reservation management system.
QT	Quantum Task	Requirements for defining a quantum task.
QTM	Quantum Task Manager	Requirements for managing quantum tasks.
QPM	Quantum Platform Manager	Requirements for managing a quantum resource.
QRC	Quantum Runtime Controller	Requirements for running operations on the quantum

Acronym	Meaning	Description
		resource.
PAA	Plugin Architecture for Applications	Requirements for formalizing the interface between QFw and the Application
PAP	Plugin Architecture for Quantum Platform	Requirements for formalizing the interface between QFw and the quantum platform.

10.2 RESOURCE MANAGEMENT FRONTEND

RID	RP	RD
QCI_QCP_001	D	The user shall have the option to specify Quantum Configuration Parameters (QCP) in YAML format
QCI_QCP_002	D	The user shall have the option to specify their preference for reserving either quantum hardware resources or quantum software simulation
QCI_QCP_003	O	The user shall have the option to specify the quantum hardware system from a list of available hardware
QCI_QCP_004	O	The user shall have the option to specify the location of the quantum hardware; cloud based or on-premises.
QCI_QCP_005	D	The user shall have the option to specify the quantum software simulation category. E.g., tensor network simulator, state vector simulator
QCI_QCP_006	O	The user shall have the option to specify multiple quantum software simulation categories it will require
QCI_QCP_007	O	The user shall have the option to specify the quantum software simulation software stack from a list of supported software stacks
QCI_QCP_008	D	The user shall have the option to specify the maximum number of qubits it will require
QCI_QCP_009	D	The user shall have the option to specify the maximum quantum gate depth it will require
QCI_QCP_010	N	The user shall have the option to specify the acceptable error rates for qubits and gates
QCI_QCP_011		The user shall have the option to specify a noise profile to apply to the simulation
QCI_QCP_012	N	The user shall have the option to specify the qubit connectivity Qubit connectivity can be expressed as a selection criteria by

RID	RP	RD
		specifying the desired pattern or arrangement of qubit connections that aligns with the requirements of your quantum algorithm or application. Here are ways to express qubit connectivity as selection criteria: 1. Graph Representation: Describe the desired qubit connectivity using a graph representation where nodes represent qubits and edges represent allowable connections. Specify the connectivity pattern that best suits your quantum circuit. 2. Adjacency Matrix: Provide an adjacency matrix that defines the allowed connections between qubits. The matrix elements indicate whether qubits are connected (1) or not (0). This allows you to explicitly specify the connectivity constraints. 3. Topological Constraints: Express topological constraints on the qubit layout, specifying the relationships between qubits in terms of physical proximity or connectivity. For example, you may require qubits to be arranged in a linear chain or a specific geometric configuration. 4. Custom Constraints: Define custom constraints based on your application's needs. For instance, you might require a specific subset of qubits to be fully connected while allowing limited connectivity for other qubits. 5. Connectivity Patterns: Explicitly state the desired connectivity pattern, such as fully connected, nearest-neighbor connectivity, or any other specific arrangement that optimally supports your quantum algorithm. When reserving quantum resources, these criteria can be provided to the quantum reservation system, allowing the system to allocate a quantum processor with the specified qubit connectivity. This ensures that the selected quantum resource is well-suited for the connectivity requirements of your quantum application.
QCI_QCP_013	N	The user shall have the option to specify the maximum decoherence time of the qubits
QCI_QCP_014	N	The user shall have the option to specify the Quantum Volume of the quantum resource required
QCI_QCP_015	D	The user shall have the option to specify the maximum number of parallel quantum tasks it requires to execute
QCI_QCP_016	D	The user shall have the option to specify the number of nodes to dedicate to the quantum software simulation environment

RID	RP	RD
QCI_QCP_017	D	The user shall have the option to specify the details of how the quantum software simulation environment will be partitioned. This can be specified in the form of: • `<Category>: <number of nodes>` Example: • `Tensor Network: 5`

10.3 RESOURCE MANAGEMENT BACKEND

RID	RP	RD
QCI_RMB_018	D	The system shall require the specification of the quantum resource type: either quantum hardware resources or quantum software simulation
QCI_RMB_019	D	The system shall use the quantum resource type to determine if it's being requested to reserve quantum resources
QCI_RMB_020	O	The system shall provide a mechanism to query existing quantum resources available for reservation
QCI_RMB_021	O	The system shall support heterogeneous quantum resources
QCI_RMB_022	O	The system shall provide a mechanism to indicate the status of available quantum resources
QCI_RMB_023	O	The system shall provide a mechanism to query the users who are currently using the quantum resources
QCI_RMB_024	O	The system shall support reserving quantum hardware if requested and if resources are available
QCI_RMB_025	O	The system shall return an error if quantum hardware is specified but no resources are available for reservation
QCI_RMB_026	D,N	In the absence of explicit quantum hardware specification, the system shall use the following criteria when attempting to find a matching quantum hardware resource • Maximum number of specified qubits

RID	RP	RD
		• Maximum quantum circuit depth • Maximum number of parallel quantum tasks • Qubit Connectivity • Maximum acceptable error rate (noise profile) • Maximum qubit decoherence time • Minimum Quantum Volume
QCI_RMB_027	D	The system shall provide a plugin architecture to allow the integration of new types of quantum hardware
QCI_RMB_028	M	The system shall provide a mechanism to query available quantum simulation categories
QCI_RMB_029	M	The system shall support two quantum simulation categories • Tensor Network • State Vector
QCI_RMB_030	M	The system shall provide a plugin architecture to allow the integration of Quantum Simulation Software Stacks into one of the above categories
QCI_RMB_031	M	The system shall provide a mechanism to query available quantum simulation software stacks
QCI_RMB_032	D	The system shall allocate enough HPC nodes to accommodate the HPC application and the Quantum Simulation Environment (QSE)
QCI_RMB_033	M	The system shall prioritize user-specified simulation category when it selects the category to use for the job allocation
QCI_RMB_034	M	The system shall prioritize user-specified simulation software stack(s) when it selects the software stack(s) to use for the quantum simulation
QCI_RMB_035	D	The system shall leverage user-specified parameters, including the maximum number of parallel quantum tasks, required qubits, and quantum circuit depth, to ascertain the necessary number of HPC nodes for the quantum simulation environment. This determination is contingent upon the supported quantum simulation software stacks. Each supported software stack will undergo profiling to establish the maximum number of qubits and circuit depth that can be simulated per HPC node using the stack.

RID	RP	RD
		Subsequently, the derived software stack profile will be combined with the application's specified maximum number of parallel quantum tasks to calculate the requisite number of HPC nodes.
QCI_RMB_036	D	In the absence of user-specified simulation category or software stack, the system shall select one based on the QCP provided by the user
QCI_RMB_037	D	The system shall partition the job allocation into a partition for the HPC application and a partition for the QSE
QCI_RMB_038	M	The system shall partition the QSE among the application specified quantum simulation categories; these partition shall be referred to as Quantum Simulation Environment Category Partitions (QSECP)
QCI_RMB_039	D	The system shall designate a head node per QSECP
QCI_RMB_040	D	The system shall partition each QSECP further per specified software stacks. Each partition shall be assigned to a specific software stack; these partitions shall be referred to as Quantum Simulation Software Stack Partition (QSSSP). There shall be at least one QSSSP
QCI_RMB_041	D	The system shall designate a head node per QSSSP
QCI_RMB_042	D	The system shall spawn a QPM process per QSSSP head node
QCI_RMB_043	D	The system shall spawn an QTM MPI process per QSSSP
QCI_RMB_044	D	The system shall provide a registration mechanism to allow QPM modules to register with the QC-RMS In the hardware quantum resource or LRQS case QC-RMS uses the registration to know which resources are available
QCI_RMB_045	D	The system shall require each registered QPM to send an alive ping every 60 seconds

10.4 QUANTUM TASKS

RID	RP	RD
QCI_QT_046	D	The system shall allow the usage of any quantum circuit composition framework, such as Qiskit or PennyLane

RID	RP	RD
QCI_QT_047	D	The system shall require quantum circuits to be submitted in the standardized circuit format
QCI_QT_048	D	The system shall provide a way for the application to identify the MPI ranks of all running QTMs
QCI_QT_049	M	The system shall provide a way for the application to identify the category of running QTMs
QCI_QT_050	M	The system shall provide a way for the application to identify the quantum simulation stack managed by the QTM
QCI_QT_051	D	The system shall provide an API to build a Quantum Task (QT). A QT is constituted of: • Quantum circuit in the standardized circuit format, e.g., QASM 2.0, QIR • Quantum circuit metadata, e.g., ○ preferred software simulation stack ○ number of qubits ○ gate depth ○ noise profile
QCI_QT_052	D	The system shall support QT submission to QTMs through standard MPI communication patterns

10.5 QUANTUM TASK MANAGER

RID	RP	RD
QCI_QTM_053	D	The QTM shall support MPI communication
QCI_QTM_054	D	The QTM shall support a standard QT format
QCI_QTM_055	D	The QTM shall support QT submission via MPI communication patterns
QCI_QTM_056	D	The QTM shall support reporting back QT completion via MPI communication patterns
QCI_QTM_057	M	The QTM shall timeout a QT if it has not completed after a configured time period
QCI_QTM_058	D	The QTM shall provide a mechanism for QPMs to register with it

RID	RP	RD
QCI_QTM_059	M	The QTM shall reject registrations from incompatible QPMs e.g., if the QTM is configured for a tensor network it will reject state vector QPM registration
QCI_QTM_060	M	The QTM shall support multiple compatible QPM registrations
QCI_QTM_061	M	The QTM shall enforce the following registration metadata • QPM software stack type • QPM supported API
QCI_QTM_062	D	The QTM shall support calling QPM APIs to submit QT
QCI_QTM_063	O	The QTM shall monitor the health of registered QPMs

10.6 QUANTUM PLATFORM MANAGER

RID	RP	RD
QCI_QPM_064	O	The QPM shall provide an API to query the reservation model
QCI_QPM_065	O	The QPM shall provide a resource reservation API
QCI_QPM_066	O	The QPM shall support QPU-Based Reservation Model • The QPU-Based Model is an approach to resource management in quantum computing where the entire quantum device is treated as a unified resource unit. In this model, the quantum processing unit (QPU) is considered as a single entity, and computational tasks are allocated to the entire QPU as a whole. This approach is similar to the current GPU model in traditional high-performance computing. • However, this approach has the potential downside of potential underuse of the QPU when jobs only target a subset of the QPU. It may not fully leverage the capabilities of the QPU, especially if the computational tasks do not require the entire quantum device. Therefore, while the QPU-Based Model simplifies resource allocation, it may not fully optimize the utilization of the quantum processing unit.

RID	RP	RD
QCI_QPM_067	N	The QPM shall support Qubit-Based Reservation Model • The Qubit-Based Model is an alternative approach to resource management in quantum computing. In this model, the qubits themselves are treated as resources, allowing a quantum processing unit (QPU) to be fully occupied simultaneously by a variety of jobs. Unlike the QPU-Based Model, where the entire QPU is treated as a unified resource unit, the Qubit-Based Model allocates computational tasks to individual qubits within the QPU. • However, in near-term devices, with qubits exhibiting different types and amounts of noise, and in the presence of limited device connectivity, this approach has shortcomings. There would be competition among jobs for the best or more suited resources within the QPU, and the allocation of tasks to specific qubits may introduce complexities in resource management. • Therefore, the Qubit-Based Model aims to provide more flexibility in resource allocation by treating individual qubits as resources, but it also introduces challenges related to qubit noise, connectivity, and competition among computational tasks for the available qubits within the QPU.
QCI_QPM_068	N	The QPM shall support Shares-Based Reservation Model • The Shares Model is a resource management approach in quantum computing that introduces a pseudo-unit of resource definition where each "share" represents a part of the device, a specific period of time, or the circuit size. Essentially, each share signifies a portion of the computational capacity of the resource. • In this model, computational tasks are allocated to shares of the quantum processing unit (QPU) based on the specific requirements of the task. For example, a task that requires a larger circuit size may be allocated more shares than a task that requires a smaller circuit size. This approach provides more flexibility in resource allocation than the QPU-Based Model, which treats the entire QPU as a unified resource unit. • The Shares Model can be treated as a sub-category of the QPU-Based Model, as it still considers the QPU as a single entity but introduces a more granular approach to resource allocation. However, this approach may introduce additional complexity in resource management,

RID	RP	RD
		as it requires a more detailed understanding of the computational requirements of each task and the available resources within the QPU.
QCI_QPM_069	D	The QPM shall provide an API to submit QTs
QCI_QPM_070	O	The QPM shall support dynamic configuration of QSSSPs during run time
QCI_QPM_071	O	The QPM shall implement a scheduling algorithm, which looks at the queued QTs and determine the best configuration of the QSSSP
QCI_QPM_072	O	The QPM shall partition the QSSSP into QSIPs at runtime, one for each instance of a quantum simulator
QCI_QPM_073	O	The QPM shall designate a head node per QSIP
QCI_QPM_074	D	The QPM shall spawn a QRC on each QSIP head node
QCI_QPM_075	D	The QPM shall schedule the QTs among the running QRCs

10.7 QUANTUM RUNTIME CONTROLLER

RID	RP	RD
QCI_QRC_076	D	The QRC shall expose an API to submit a QT for execution
QCI_QRC_077	D	The QRC shall report back when a QT has finished execution
QCI_QRC_078	D	The QRC shall be responsible for transpiling quantum tasks for the specific hardware or software simulator it manages
QCI_QRC_079	O	The QRC shall interface with the quantum platform specific API to perform quantum platform specific operations. e.g., running a QT, configuring the quantum platform, etc.
QCI_QRC_080	O	The QRC shall handle hardware or software specific errors which are unique to the underlying quantum platform and translate them into framework specific errors

10.8 PLUGIN ARCHITECTURE

RID	RP	RD
QCI_PAP_081	D	The system shall provide a standard reservation API implemented by the quantum platform
QCI_PAP_082	M	The system shall provide a standard reservation model query API implemented by the quantum platform
QCI_PAP_083	D	The system shall provide a standard quantum task submission API implemented by the quantum platform
QCI_PAP_084	M	The system shall provide a standard quantum task Query API implemented by the quantum platform
QCI_PAP_085	D,M	The system shall provide a standard set of API return codes to insure common behavior among all platforms
QCI_PAP_086	D,M	The system shall provide a set of APIs to build and parse a quantum task; thereby ensure quantum task standardization among all platforms
QCI_PAP_087	O	The system shall provide a standard set of APIs to build the canonical Quantum Result format
QCI_PAP_088	O	The system shall provide a software library to formalize the interface between the quantum platform provider and the system
QCI_PAA_089	O	The system shall provide a backend plugin for supported circuit composition frameworks to integrate with the QFw