A Simulation Framework for Workload Management in Hybrid Quantum-HPC Cloud System

Waylon Luo
Department of Computer Science
Kent State University
Kent, OH, USA
wluo1@kent.edu

Cheng-Chang Lu
Qradle Inc.
Kent, OH, USA

Tong Zhan
Meta Platform
Bellevue, WA, USA

Qiang Guan
Department of Computer Science
Kent State University
Kent, OH, USA
qguan@kent.edu

Abstract

The convergence of quantum computing and high-performance computing (HPC) is giving rise to hybrid cloud platforms that coordinate workflows across classical and quantum resources. These heterogeneous environments introduce new challenges for workload management, including device variability, noise-prone quantum processors, and the need for synchronized orchestration across distributed systems. This survey reviews state-of-the-art techniques for scheduling, resource allocation, and workflow management in classical HPC, quantum clouds, and emerging hybrid infrastructures. A taxonomy is proposed to categorize workload management approaches by scheduling strategy, allocation model, orchestration mechanism, and performance metric. We present *HybridCloudSim*, a novel simulation framework designed to model hybrid quantum–HPC cloud environments and optimize resource usage under realistic constraints. HybridCloudSim is a conceptual foundation and a practical tool for advancing workload management in hybrid quantum–HPC systems.

CCS Concepts

• **Computing methodologies → Model development and analysis**; *Distributed computing methodologies.*

Keywords

Hybrid cloud computing, workload management, quantum job scheduling, simulation framework

ACM Reference Format:
Waylon Luo, Cheng-Chang Lu, Tong Zhan, and Qiang Guan. 2025. A Simulation Framework for Workload Management in Hybrid Quantum-HPC Cloud System. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3731599.3767548

1 Introduction

Advances in *quantum computing* and *high-performance computing (HPC)* are reshaping the computational landscape, leading to breakthroughs in materials science, cryptography, optimization, and machine learning. Quantum processors (QPUs) promise exponential speedups for specific problem classes but are constrained by limited qubit counts, short coherence times, and high error rates, restricting their applicability. In contrast, HPC systems deliver unmatched performance for large-scale classical workloads but cannot efficiently solve problems that are inherently quantum in nature.

To exploit the complementary strengths of both paradigms, **hybrid quantum–HPC cloud platforms** are emerging, where QPUs are integrated as specialized accelerators within HPC workflows. This integration raises new challenges in **workload management**, including heterogeneous resource allocation, noise-aware scheduling, co-scheduling across device types, and minimizing communication overhead. Figure 1 depicts a representative hybrid architecture in which a unified scheduler coordinates execution across HPC clusters and QPUs, supported by shared storage and high-speed interconnects.

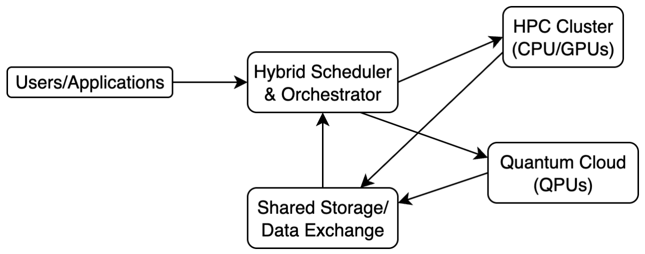
Users/Applications
Hybrid Scheduler & Orchestrator
HPC Cluster (CPU/GPUs)
Quantum Cloud (QPUs)
Shared Storage/ Data Exchange

Figure 1: Conceptual architecture of hybrid quantum–HPC cloud workload management. A unified scheduler orchestrates job execution across classical HPC clusters and quantum processors, leveraging shared storage and communication channels.

The motivation for building such systems lies in maximizing the utilization of available resources while ensuring efficient execution of workloads. Hybrid platforms inherently involve devices with vastly different capabilities, costs, and operational constraints. Without intelligent workload distribution, QPUs may remain idle due to scheduling delays, or HPC nodes may be underutilized while waiting for quantum tasks to complete. Effective workload management is therefore essential to improve throughput, reduce execution time, and optimize the combined use of classical and quantum resources.

While workload management in classical HPC clouds [37] and quantum computing platforms [13, 29] has been studied, hybrid environments remain relatively underexplored. This work addresses this gap by reviewing current scheduling, resource allocation, and orchestration strategies across classical, quantum, and hybrid systems. We present a taxonomy of workload management approaches, analyze how lessons from HPC can inform quantum–classical integration, and identify open research challenges.

This paper is divided into two parts. The first part provides a review of workload management strategies in classical, quantum, and hybrid environments, highlighting key design principles and limitations; and a taxonomy that categorizes existing approaches based on scheduling strategies, allocation models, orchestration mechanisms, and performance metrics. The second part introduces *HybridCloudSim*, a novel simulation framework for hybrid quantum–HPC cloud environments designed to optimize resource usage and manage workloads. We then demonstrate a use case of *HybridCloudSim* and analyze the resulting simulation output.

The remainder of this paper is organized as follows. Section 2 presents background on quantum computing, high-performance computing, and hybrid quantum–HPC systems, along with core workload management concepts. Section 3 defines a taxonomy for classifying workload management approaches. Section 4 reviews established workload management techniques in classical HPC clouds and examines their applicability to hybrid environments. Section 5 describes the architecture of the proposed HybridCloudSim framework. Section 6 showcases a representative use case illustrating the framework's capabilities. Finally, Section 7 summarizes the paper and outlines future research directions.

2 Background and Foundations

In this section, we discuss the foundations for understanding workload management in hybrid quantum–HPC cloud environments. We first review the basics of quantum computing, high-performance computing (HPC), and their integration into hybrid systems. Finally, we introduce key workload management concepts relevant to this survey.

2.1 Quantum and High-Performance Computing Overview

Quantum computing leverages the principles of quantum mechanics to perform computations beyond the capabilities of classical systems [36]. Feynman proposed in his 1982 keynotes that classical computers are fundamentally limited in simulating physical systems, and quantum computers as a more powerful platform for modeling quantum physics [12]. Current quantum devices, often referred to as Noisy Intermediate-Scale Quantum (NISQ) processors, are characterized by limited qubit counts, short coherence times, and high susceptibility to errors. Cloud-based access to quantum hardware through providers such as IBM Quantum [29], Amazon Braket [13], and Microsoft Azure Quantum [26] has accelerated research but also introduced additional challenges in scheduling, resource sharing, and managing performance variability.

High-performance computing (HPC) has long been the backbone of scientific computing, supporting large-scale simulations and data analytics. Modern HPC systems integrate clusters of CPUs, GPUs, and other accelerators, coordinated by workload managers such as Slurm [37], portable batch system (PBS) [14], and Kubernetes [7]. These schedulers apply queue-based resource allocation, priority mechanisms, 34e parallel job execution to maximize throughput and utilization. The maturity of HPC workload management frameworks and their performance-oriented optimizations provides insights for hybrid quantum–HPC scheduling, particularly in job prioritization, load balancing, and adaptive resource control.

2.2 Hybrid Quantum–HPC Systems

Hybrid platforms integrate quantum processors as specialized accelerators within classical HPC infrastructures. Examples include the deployment of quantum resources alongside supercomputers at research facilities such as Oak Ridge National Laboratory and commercial hybrid platforms such as Hewlett Packard Enterprise (HPE) offered by major cloud providers [27]. These systems enable workflows in which quantum subroutines are offloaded to QPUs while classical tasks run on HPC nodes. However, they also introduce orchestration challenges, such as coordinating heterogeneous tasks across devices, minimizing data movement overhead, and mitigating the effects of quantum noise in distributed execution environments.

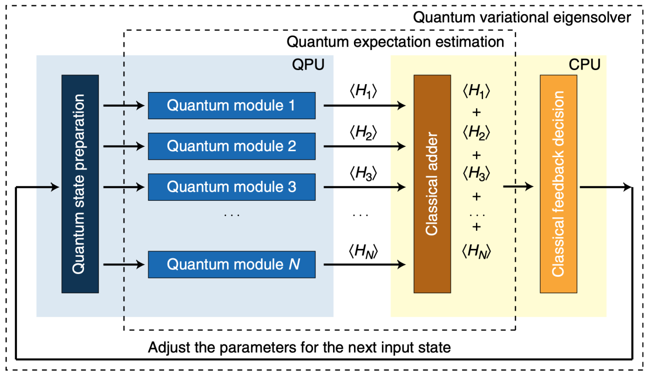
Quantum variational eigensolver
Quantum expectation estimation
QPU
CPU
Quantum state preparation
Quantum module 1
Quantum module 2
Quantum module 3
Quantum module N
$\langle H_1 \rangle$
$\langle H_2 \rangle$
$\langle H_3 \rangle$
$\langle H_N \rangle$
Classical adder
$\langle H_1 \rangle + \langle H_2 \rangle + \langle H_3 \rangle + \dots + \langle H_N \rangle$
Classical feedback decision
Adjust the parameters for the next input state

Figure 2: Architecture of a hybrid quantum–classical workflow using the variational quantum eigensolver (VQE). Quantum modules on QPUs compute expectation values for Hamiltonian terms, which are aggregated on CPUs by a classical optimization algorithm. The updated parameters are iteratively fed back to the QPUs, forming a quantum–classical feedback loop. Image Source: Peruzzo et al. (2014) [25]

Figure 2 illustrates a representative hybrid quantum–classical workflow based on the variational quantum eigensolver (VQE), a prominent example of near-term hybrid algorithms [25]. In this model, quantum processors (QPUs) prepare and execute quantum circuits corresponding to different terms in the Hamiltonian, producing expectation values for each term. These results are aggregated on classical processors (CPUs), where a classical optimization algorithm determines updated parameters for the quantum circuits. The updated parameters are then fed back to the QPUs, completing an iterative quantum–classical feedback loop. This architecture exemplifies the tightly coupled interaction between quantum and classical resources in hybrid cloud systems, where high-performance classical nodes handle optimization and orchestration, while QPUs execute quantum subroutines. Such workflows place stringent demands on workload management, requiring low-latency communication, effective co-scheduling of quantum and classical tasks, and noise-aware execution strategies for computational accuracy and efficiency.

2.3 Workload Management Concepts

We introduce workload management in hybrid quantum-HPC systems by coordinating job execution across classical [11] and quantum resources [22, 23, 28]. Scheduling determines the order and placement of jobs [2] on available devices, often incorporating noise and error-awareness in quantum clouds to enhance throughputs' quality. Resource allocation assigns computational and communication resources to meet job requirements while balancing competing demands across heterogeneous hardware [15]. Orchestration is responsible for making sure that hybrid workflows spanning multiple devices and environments execute correctly and efficiently, managing interdependencies between quantum and classical stages. Performance metrics such as runtime, throughput, fidelity, cost, and latency are used to evaluate system effectiveness and guide optimization. We developed these concepts as the foundation of hybrid cloud workload management taxonomy discussed in the next section.

2.4 Literature Reviews

Simulation plays a crucial role in studying workload management for quantum and hybrid cloud [4, 18]. With simulations, researchers can explore scheduling policies, resource allocation strategies, and system scalability without incurring the high costs or facing the limited availability of real quantum hardware. Digital twin approaches are popular, as they provide realistic modeling of hardware constraints, noise characteristics, and network communication.

One example is *QCloudSim*, a digital twin framework designed to model scalable quantum cloud infrastructures [20]. *QCloudSim* supports configurable quantum devices, noise-aware execution models, and flexible scheduling policies, allowing evaluation of workload management strategies under realistic conditions. Its modular architecture accommodates heterogeneous devices and can be extended to represent hybrid environments that integrate both quantum and classical resources.

Another recent contribution addresses adaptive job scheduling in quantum clouds using reinforcement learning [21]. This work formulates the scheduling problem as a Markov Decision Process and applies reinforcement learning to allocate quantum jobs efficiently while adapting to device noise and workload variability. The approach demonstrates improved performance over schedulers and highlights the potential of machine learning methods for hybrid workload management.

GymCloudSim [16] focuses on classical high-performance cluster scheduling and energy-driven cloud scaling, without addressing quantum-specific or multi-QPU challenges. QuNetSim [10] and NetSquid [8] provide discrete-event simulations for quantum networks, with NetSquid incorporating noise models; however, both focus exclusively on network communication rather than distributed computation.

CloudSim [5, 6] is a widely adopted simulation framework for modeling and analyzing cloud computing infrastructures, services, and resource management policies. The initial version [6] introduced a flexible architecture for simulating large-scale cloud environments, supporting customizable data center configurations, virtualized resources, and service provisioning scenarios. The extended version [5] evolved into a comprehensive toolkit, adding features for modeling network topologies, advanced resource allocation policies, and dynamic workload generation.

Despite these advances, existing simulation frameworks [3, 9, 17, 19, 24, 32, 33] address various layers of the quantum stack, but none provide a comprehensive platform for modeling hybrid quantum–HPC clouds. Current tools lack the ability to simulate tightly coupled quantum–classical workflows, co-schedule across heterogeneous resources, and apply noise-aware execution strategies within a unified environment. In this work, we present **HybridCloudSim**, a hybrid quantum–HPC cloud simulation framework that fills this gap by supporting realistic modeling, scheduling, and performance evaluation of integrated quantum–classical workloads.

3 Taxonomy of Workload Management Approaches

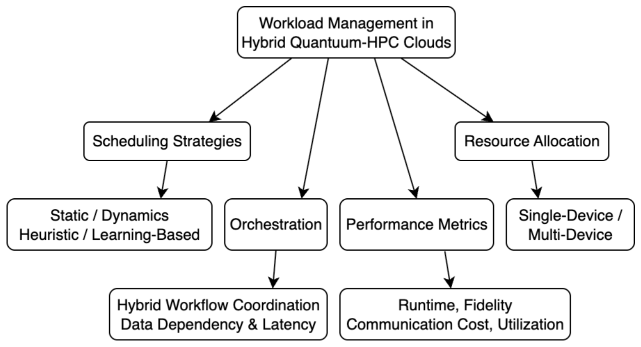
Workload Management in Hybrid Quantuum-HPC Clouds
Scheduling Strategies
Resource Allocation
Static / Dynamics Heuristic / Learning-Based
Orchestration
Performance Metrics
Single-Device / Multi-Device
Hybrid Workflow Coordination Data Dependency & Latency
Runtime, Fidelity Communication Cost, Utilization

Figure 3: Proposed taxonomy for workload management in hybrid quantum–HPC clouds. Approaches are categorized along four key dimensions: scheduling, resource allocation, orchestration, and performance metrics.

Workload management in hybrid quantum–HPC clouds is inherently more complex than in traditional computing environments due to the heterogeneity of resources, the presence of quantum noise, and the distributed orchestration required across diverse devices. To systematically analyze existing research, we propose a taxonomy that categorizes workload management approaches along four key dimensions: **scheduling strategies**, **resource allocation**, **orchestration**, and **performance metrics** as illustrated in Fig. 3.

3.1 Scheduling and Resource Allocation Strategies

Scheduling determines how jobs are prioritized and mapped to computing resources. In hybrid environments, it must address both classical workloads and quantum circuits while considering constraints such as QPU availability, calibration status, and error rates. Static scheduling assigns resources according to predefined plans

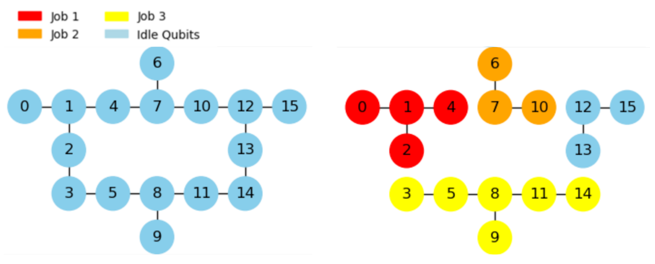

Figure 4: Example of parallel programming on a QPU using the retired *IBM Guadalupe* device. The QPU's qubits are divided into disjoint subsets, each assigned to a separate job. The left diagram shows the idle state, and the right diagram shows the qubit allocation during concurrent execution.

prior to execution, as in batch HPC schedulers such as Slurm [37]. Dynamic scheduling adapts allocations at runtime based on system state, job progress, and real-time noise data from QPUs. Heuristic scheduling applies rule-based methods—such as shortest-job-first or fidelity-aware heuristics—to achieve efficient, though not necessarily optimal, decisions. More recently, learning-based scheduling leverages machine learning or reinforcement learning to improve decision-making under uncertainty.

Resource allocation governs the distribution of computational resources—such as CPUs, GPUs, QPUs—and communication bandwidth to executing jobs. In hybrid platforms, allocation policies must balance HPC throughput with quantum fidelity, often co-allocating classical nodes for pre- and post-processing of quantum tasks. Single-device allocation assigns entire jobs to individual QPUs or HPC nodes without decomposition, simplifying execution but limiting scalability. Multi-device allocation partitions large jobs into subcomponents executed across multiple devices, requiring inter-device communication, synchronization, and, for quantum workloads, strategies to preserve fidelity in distributed execution.

Parallel programming in QPUs allows multiple quantum jobs to execute concurrently on disjoint subsets of qubits within the same device. As illustrated in Fig. 4, qubits are partitioned into independent groups, each allocated to a different job. This approach improves device utilization and reduces job waiting times by enabling concurrent execution without interference between jobs. Effective parallel scheduling must account for the QPU's topology, making certain that qubit assignments for different jobs do not overlap and that logical connectivity requirements are satisfied.

Orchestration is the coordination of hybrid workflows spanning quantum and classical resources. This process must handle data dependencies, manage communication latency, and support recovery from hardware or software failures. Commercial cloud platforms, such as Azure Quantum, provide APIs for orchestrating hybrid workflows, while current research focuses on scalable orchestration layers that optimize performance across distributed and heterogeneous infrastructures. Such frameworks must align task execution timelines, minimize idle periods for both QPUs and HPC nodes, and maintain seamless integration between classical and quantum computation stages.

3.2 Performance Metrics

Evaluating workload management strategies in hybrid environments requires metrics that capture both classical and quantum performance dimensions. Runtime efficiency measures total execution time and makespan, providing insight into overall throughput. Fidelity preservation assesses the accuracy of quantum computations in the presence of noise, a critical factor for quantum-enabled workloads. Communication cost accounts for the time and bandwidth overhead of inter-device data transfers, while cost and utilization metrics measure resource efficiency and the monetary impact in cloud-based settings. Together, these metrics form a comprehensive basis for comparing workload management strategies.

These four dimensions collectively define the design space for workload management in hybrid quantum–HPC clouds. The proposed taxonomy organizes existing approaches and highlights potential research gaps and opportunities for innovation.

4 Workload Management in Classical HPC Clouds

High-performance computing (HPC) clouds have evolved over decades to deliver robust workload management, supporting efficient scheduling and resource allocation across large-scale clusters. These systems employ mature resource managers and job schedulers to optimize utilization while minimizing turnaround time. Understanding these classical approaches is essential, as many of their principles can be adapted to the specific requirements of hybrid quantum–HPC workload management.

4.1 Traditional HPC Scheduling Frameworks

HPC clusters typically rely on workload managers such as Slurm, PBS Pro, and LSF to manage thousands of compute nodes [37]. These schedulers handle the complete job lifecycle, from submission to execution, by managing queues, prioritizing workloads, and allocating resources according to defined policies. Common strategies include First-Come-First-Serve (FCFS), backfilling to insert smaller jobs without delaying larger ones, and priority-based scheduling to favor specific users or applications. Resource requests in these systems are explicit, specifying requirements such as CPU or GPU count, memory size, and expected runtime, enabling efficient matching of jobs to available resources.

In modern cloud-based HPC environments, container orchestration frameworks such as Kubernetes extend these capabilities with elastic resource management and fault-tolerant execution. These frameworks can dynamically scale resources, manage heterogeneous hardware, and maintain high availability—features increasingly relevant to hybrid quantum–HPC workloads.

4.2 HPC Resource Allocation Models

Traditional HPC scheduling is built around maximizing throughput and minimizing makespan. Batch scheduling organizes jobs into queues and executes them according to predetermined policies that balance fairness and efficiency. Backfilling strategies opportunistically advance smaller jobs in the queue when they can be executed without delaying higher-priority tasks, improving overall system utilization. Priority scheduling, meanwhile, grants preferential access to resources for certain jobs or users, often based

Table 1: Comparison between classical HPC, quantum cloud, and hybrid quantum–HPC workload management.

Aspect	Classical HPC Clouds	Quantum Clouds	Hybrid Quantum–HPC Clouds
Resource Type	Homogeneous nodes (CPUs, GPUs)	Noisy quantum processors (QPUs) with limited qubit counts	Heterogeneous mix of HPC nodes and noisy QPUs
Scheduling Goal	Throughput and fairness	Maximize fidelity, qubit utilization, and minimize queue wait time	Throughput, fidelity, and noise-aware optimization
Job Granularity	Parallel jobs with predictable performance	Quantum circuits with variable depth, shots, and hardware noise	Quantum subroutines embedded within HPC workflows, with variable noise and execution time
Challenges	Scalability, energy efficiency	Noise mitigation, calibration drift, limited connectivity	Noise mitigation, inter-device coordination, communication latency between QPU and HPC nodes

on institutional policy or service-level agreements. While these models perform well for homogeneous HPC resources, they require adaptation for hybrid systems where quantum devices exhibit variable performance, limited availability, and periodic calibration requirements.

The design of hybrid workload management systems can be adopted from several principles from classical HPC scheduling. Queue management techniques, such as backfilling, can reduce idle time for both HPC nodes and quantum processors. Scalability in mature HPC schedulers demonstrates how to manage thousands of concurrent jobs and resources effectively, a capability that will be crucial for future large-scale hybrid systems. Policy-driven scheduling frameworks, which allow administrators to implement flexible and workload-specific rules, can also be extended to accommodate hybrid workloads that combine classical and quantum tasks. However, hybrid environments introduce additional constraints—including noise awareness, calibration scheduling, and inter-device communication latency—that go beyond the capabilities of current HPC workload managers.

4.3 Comparison with Hybrid Requirements

The transition from classical HPC workloads to hybrid quantum–HPC workflows introduces fundamentally different scheduling and resource allocation challenges. Classical HPC clouds operate on largely homogeneous compute nodes, such as CPUs and GPUs, where performance is predictable and resource allocation policies focus on maximizing throughput and maintaining fairness. In contrast, quantum clouds comprise specialized, noisy quantum processors (QPUs) constrained by limited qubit counts, hardware connectivity, and calibration-dependent noise characteristics. Scheduling in this context must also consider fidelity optimization, qubit utilization efficiency, and device-specific noise profiles in addition to minimizing wait times.

Hybrid quantum–HPC clouds integrate both HPC and quantum resources, creating highly heterogeneous computing environments. Workflows often alternate between quantum and classical computation stages, with quantum subroutines embedded within larger HPC pipelines. This integration requires coordination between fundamentally different execution models, making noise mitigation, inter-device communication, and synchronization critical to overall performance.

Table 1 summarizes the distinct characteristics and challenges of classical HPC clouds, quantum clouds, and hybrid quantum–HPC clouds across four key dimensions: resource type, scheduling goals, job granularity, and operational challenges.

5 Framework Architecture

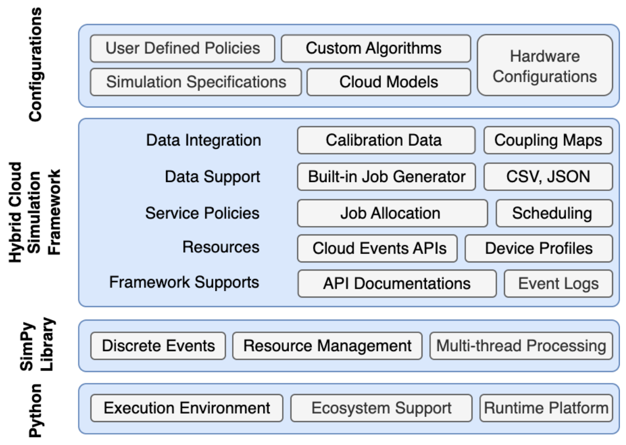

Figure 5: Layered architecture of HybridCloudSim: Python, SimPy Engine, Hybrid Cloud Simulation Core, and Configurations.

The architecture of the framework follows a layered design (Fig. 5), which supports the modeling of hybrid quantum–classical workflows that leverage both quantum processing units (QPUs) and high-performance classical processors (CPUs). The layers and their respective roles are described below.

SimPy Engine Layer: The foundational layer uses the SimPy [30] discrete-event simulation engine, providing APIs for time progression, event scheduling, concurrency, and shared resource management. This layer drives the simulation timeline and coordinates parallel execution across heterogeneous device types. SimPy's event-driven model supports synchronous or asynchronous task execution between QPUs and CPUs, allowing accurate modeling of hybrid workloads with iterative quantum–classical feedback loops.

Core Layer: The core layer supports heterogeneous device orchestration through dedicated abstractions, such as `CPUDevice`

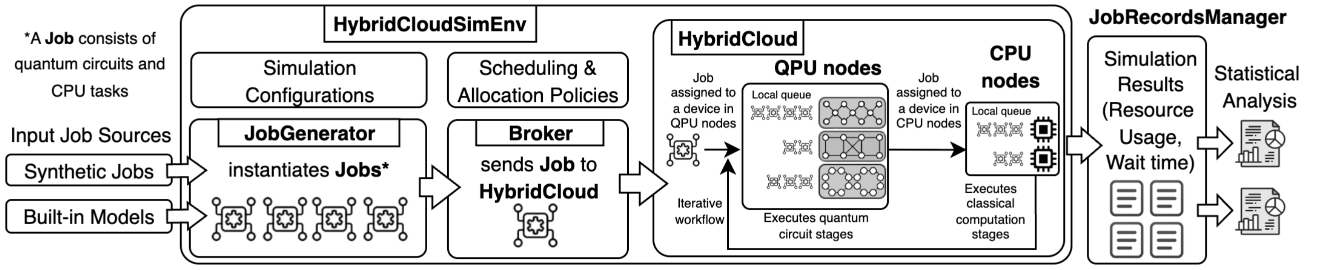
*A Job consists of quantum circuits and CPU tasks
Input Job Sources
Synthetic Jobs
Built-in Models
HybridCloudSimEnv
Simulation Configurations
Scheduling & Allocation Policies
JobGenerator
instantiates Jobs*
Broker
sends Job to HybridCloud
HybridCloud
Job assigned to a device in QPU nodes
QPU nodes
Local queue
Job assigned to a device in CPU nodes
CPU nodes
Local queue
Iterative workflow
Executes quantum circuit stages
Executes classical computation stages
JobRecordsManager
Simulation Results (Resource Usage, Wait time)
Statistical Analysis

Figure 6: HybridCloudSim ecosystem and data flow. Jobs—hybrid workloads comprising quantum circuits and classical CPU tasks—are instantiated from synthetic datasets or built-in models by the `JobGenerator` under user-defined simulation configurations and policies in `HybridCloudSimEnv`. The `Broker` dispatches each job to the `HybridCloud` execution layer, where quantum stages run on QPU nodes and classical stages run on CPU nodes; an iterative quantum–classical workflow coordinates dependencies between them. Execution metadata are collected by the `JobRecordsManager` to produce simulation results (resource usage and wait time) for downstream statistical analysis.

(for high-performance classical processors) and `QPUDevice` (for quantum processors), integrated within a unified simulation environment. This environment models hybrid workflows in which quantum and classical tasks can execute in parallel or in tightly coupled iterative sequences. Key components include a hybrid-aware `Broker` (e.g., `ParallelBroker`) for dispatching jobs to both QPUs and CPUs, a `Scheduler` that applies policies based on device type, workload dependency, and execution order and APIs for defining composite `Jobs` containing quantum and classical stages.

The core layer also models device-specific constraints such as qubit topology, gate fidelities, and CPU computational throughput. Its modular design allows integration of custom scheduling algorithms, hybrid partitioning strategies, and workload generators.

Configurations Layer: The configuration layer provides a user-facing interface for defining simulation scenarios. It enables specification of hardware configurations (e.g., number and type of QPUs and CPUs), scheduling and allocation policies for each device class, and input workloads, including synthetic job datasets and benchmark-driven hybrid tasks.

This layer allows the framework to adapt to a wide range of research objectives, from benchmarking hybrid algorithms to optimizing job orchestration across large-scale hybrid cloud infrastructures.

5.1 Components

HybridCloudSim is composed of several core components that collectively enable the simulation of hybrid quantum–classical cloud computing environments. Each component models a distinct part of the system, from device representation to workload orchestration and scheduling.

At the center of the framework is the `HybridCloudSimEnv` class, which manages the overall simulation environment. This environment registers available devices, receives and schedules incoming jobs, and coordinates the execution of both quantum and classical tasks. It also manages event scheduling and synchronization between stages, enabling the simulation of hybrid workflows that require iterative interaction between quantum and classical computation, as well as those that run tasks in parallel.

Device modeling is handled through two primary abstractions. The `QPUDevice` class represents quantum processing units (QPUs) with configurable hardware and operational properties, including device name, qubit topology (nodes and positions), number of qubits, coupling map, and color map for visualization. It supports maintenance scheduling through parameters such as maintenance interval, duration, and enable switch, and maintains a job queue, container, and priority resource within a SimPy environment. For hardware-specific modeling, the `IBM_QuantumDevice` subclass adds IBM-specific metrics, including circuit layer operations per second (CLOPS) [35], quantum volume, median T_1 and T_2 coherence times, processor type, and calibration file path, as well as extracted readout, single-qubit gate, and two-qubit gate error rates.

The `CPUDevice` class models classical high-performance processors, characterized by attributes such as device name, CPU capacity (in processing units), and memory bandwidth capacity (in GB/s). Within a SimPy environment, it manages a job queue, CPU unit container, memory bandwidth container, and priority resource. Classical devices are used for workloads such as numerical optimization, classical simulation, and data preprocessing or postprocessing in hybrid workflows.

Hybrid workloads are represented by the `Job` abstraction, which can consist of multiple execution stages spanning different computational phases. Quantum stages are executed on QPUs, while classical stages are assigned to CPUs. Each job is characterized by properties such as a unique identifier, circuit name, number of qubits, circuit depth, number of shots, gate set, expected execution time, priority level, noise model, arrival time, and number of iterations. These attributes allow precise modeling of workload requirements, execution complexity, and performance constraints.

Job dispatching is managed by `ParallelBroker`, which maintains a queue of pending jobs and assigns their stages to available devices. The broker ensures that execution order respects inter-stage dependencies and that workloads are balanced across heterogeneous devices. Scheduling decisions are made by policies, which determine the sequence and allocation of job stages based on factors such as device availability, estimated execution times, and inter-device communication delays. The scheduler can also incorporate advanced optimization strategies, including reinforcement learning and fidelity-aware scheduling.

Supporting utilities are provided for logging, result tracking, and visualization, allowing users to analyze performance and refine workload management strategies. This modular design illustrated in Fig. 6, allows HybridCloudSim to be adapted to a variety of research scenarios in hybrid quantum–classical cloud computing.

6 Use Case

To evaluate the performance of the proposed hybrid quantum–HPC simulation framework, we conducted an experiment using a heterogeneous system configuration consisting of two quantum processing units (QPUs) and two classical processing units (CPUs). Each QPU was modeled with a capacity of 128 qubits and 30,000 CLOPS, while each CPU was configured with 100 cores and a memory bandwidth of 200 units. The scheduling policy supported parallel execution, allowing multiple jobs to run concurrently across QPU and CPU resources.

A total of 1,000 jobs were dispatched from a pre-generated CSV workload file. Job parameters were randomly sampled from the following ranges:

- Number of qubits required: $[5, 15]$
- Number of processors requried: $[4, 10]$
- Memory bandwidth: $[15, 25]$
- Circuit depth: $[5, 20]$
- Number of shots: $[5, 15] \times 100$
- Iterations per job: $[3, 7]$

In this experiment, all jobs are assumed to have equal priority.

6.1 Simulation Configuration

The simulation was executed using the `QCloud` package with the configuration shown in Listing 1.

```python
from QCloud import *

PRINTLOG = False
# Devices
ibm_kawasaki = IBM_Kawasaki(env=None, name="QPU-1",
                            printlog=PRINTLOG)
ibm_kyiv = IBM_Kyiv(env=None, name="QPU-2",
                            printlog=PRINTLOG)
AMD_EPYC_9654-1 = CPU("CPU-1", env=None)
AMD_EPYC_9654-2 = CPU("CPU-2", env=None)

# Hybrid environment
sim_env = HybridCloudSimEnv(
    qpu_devices=[ibm_kawasaki, ibm_kyiv],
    cpu_devices=[AMD_EPYC_9654-1, AMD_EPYC_9654-2],
    broker_class=ParallelBroker,
    job_feed_method='dispatcher',
    file_path='synth_job_batches/1000-jobs.csv',
    job_generation_model=None,
    printlog=PRINTLOG)

sim_env.run()
```

Listing 1: Python code for configuring and running the hybrid QPU/CPU simulation.

Two IBM QPU models (`IBM_Kawasaki` and `IBM_Kyiv`) and two CPU models were instantiated and registered in the hybrid simulation environment `HybridCloudSimEnv`. A parallel broker is used in this experiment to allow multiple jobs to be processed on QPUs and CPUs concurrently. Jobs were loaded from a synthetic CSV file and fed into the system using the `dispatcher` job feed method.

As an illustrative example, the detailed execution record for Job 0 is shown in Listing 2. This job arrives at simulation time $t = 1.02$ s and undergoes three quantum-classical execution iterations. The first QPU stage starts at $t = 1.0201$ s on QPU-1 using 6 qubits, and completes at $t = 4.2385$ s. It is followed by a CPU stage starting at $t = 4.2385$ s on CPU-2, using 8 CPU units and around 20 GB/s of memory bandwidth, finishing at $t = 6.6306$ s. The second QPU stage begins at $t = 6.6306$ s on QPU-1 (6 qubits), ending at $t = 9.849$ s, followed by a CPU stage on CPU-2 with 4 CPU units, ending at $t = 11.4015$ s. The final QPU stage runs on QPU-2 from $t = 11.4015$ s to $t = 14.5126$ s (6 qubits), and is followed by a CPU stage on CPU-2 using 7 CPU units until $t = 15.6912$ s. This alternating execution pattern between QPU and CPU stages reflects the hybrid workload processing model employed in the simulation.

```
{0: {'arrival': 1.02,
  'devc_name': ['QPU-1', 'CPU-2', 'QPU-1', 'CPU-2', 'QPU-2',
      'CPU-2'],
  'qpu_arrive': [1.0201, 6.6306, 11.4015],
  'qpu_start': [1.0201, 6.6306, 11.4015],
  'qpu_units': [6, 6, 6],
  'qpu_finish': [4.2385, 9.849, 14.5126],
  'cpu_arrive': [4.2385, 9.849, 14.5126],
  'cpu_start': [4.2385, 9.849, 14.5126],
  'cpu_units': [8, 4, 7],
  'cpu_mem_bw': [20, 25, 23],
  'cpu_finish': [6.6306, 11.4015, 15.6912]}}
```

Listing 2: Execution record of Job 0 in the simulation.

Such detailed job records are instrumental for visualizing execution patterns, analyzing bottlenecks, and optimizing resource usage and workload management strategies.

6.2 Data Visualization

Figure 7 shows the execution timeline for the first ten jobs, represented as a Gantt chart. Blue bars denote QPU execution phases, while red bars correspond to CPU execution phases, clearly illustrating the alternating processing pattern across hybrid resources.

To compute the time-series resource utilization, the simulation timeline is sampled at fixed intervals (Δt). For each timestamp t_k, the resources actively in use are accumulated across all jobs. For QPUs, the total qubits in use $Q_{\text{busy}}(t_k)$ are summed; for CPUs, the active CPU units $C_{\text{busy}}(t_k)$ and memory bandwidth units $B_{\text{busy}}(t_k)$ are aggregated separately. Each is normalized by the corresponding total device capacity (Q_{cap}, C_{cap}, B_{cap}) to obtain a utilization percentage:

$$U_{\text{res}}(t_k) = \frac{\text{busy\_units}_{\text{res}}(t_k)}{\text{capacity}_{\text{res}}} \times 100\%, \quad \text{res} \in \{\text{QPU}, \text{CPU}, \text{MemBW}\}.$$

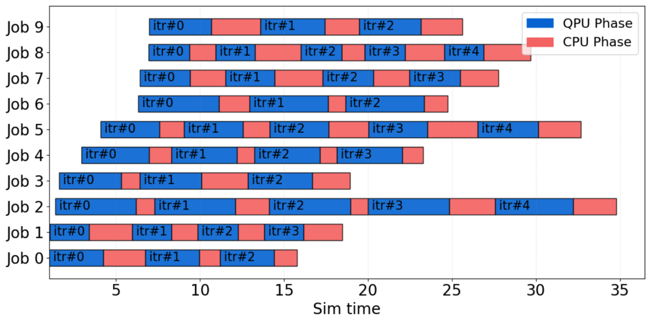

Figure 7: Gantt chart illustrating the hybrid QPU/CPU execution timeline for the first five jobs in the simulation. Each job is represented by a horizontal row, where blue segments correspond to QPU execution phases and red segments correspond to CPU execution phases. Iterations are labeled for clarity, showing the alternating processing stages across different jobs.

This yields three utilization curves—QPU, CPU, and memory bandwidth—representing the fraction of total available capacity consumed at each time step, enabling fine-grained analysis of workload dynamics throughout the simulation.

Resource usage over the simulation is depicted in Figure 8. QPU utilization (blue) exhibits frequent peaks due to high concurrency and device contention, while CPU utilization (red) and memory bandwidth utilization (green) reflect the post-QPU processing stages and memory-intensive workloads.

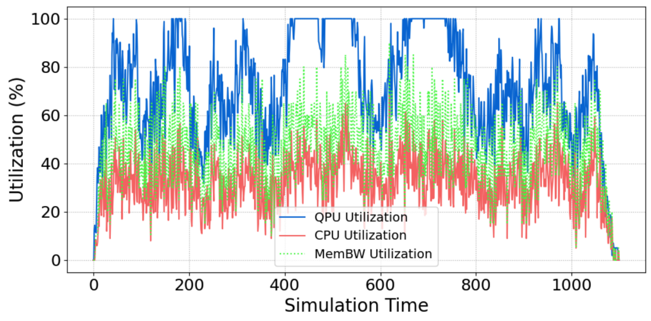

Figure 8: Time-series plot showing utilization trends for QPU, CPU, and memory bandwidth throughout the simulation. QPU utilization (blue) exhibits higher peaks due to parallelism and device contention, while CPU utilization (red) and memory bandwidth utilization (green) reflect post-QPU processing demands and memory-bound workloads, respectively.

The overall resource usage for each device type (QPU, CPU units, and memory bandwidth) is computed as the ratio of the total busy capacity to the total available capacity, expressed as a percentage. Let R_{busy} denote the measured busy units for a given resource type res $\in \{\text{QPU}, \text{CPU}, \text{MemBW}\}$, and let R_{cap} denote its maximum

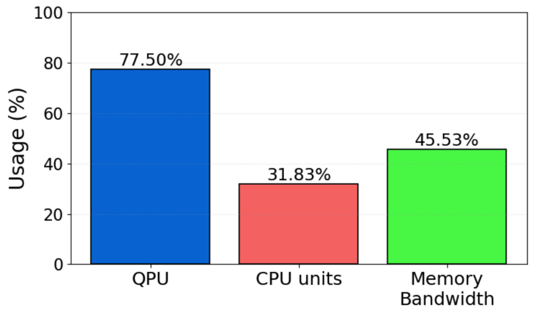

Figure 9: Overall average utilization of QPU, CPU units, and memory bandwidth across the simulation. QPU utilization reaches approximately 76.85%, CPU utilization is around 36.26%, and memory bandwidth utilization is approximately 45.32%. These values reflect the cumulative resource demands imposed by all jobs over the simulation period.

capacity. The utilization percentage is given by:

$$U_{\text{res}} = \frac{R_{\text{busy}}}{R_{\text{cap}}} \times 100\%.$$

This metric captures the proportion of total resource capacity in percentage actively used during the simulation and provides a direct measure of hardware load across different components.

Figure 9 summarizes the overall average utilization of QPU, CPU units, and memory bandwidth. The QPU reached an average utilization of 76.85%, while CPU units and memory bandwidth achieved 36.26% and 45.32% utilization, respectively. These values reflect the cumulative impact of hybrid workloads on resource demands across the simulation duration.

7 Conclusion

In this work, we introduced HybridCloudSim, a simulation framework designed to model and analyze the coordinated execution of workloads across quantum processing units and high-performance classical processors. The framework allows researchers to investigate diverse scheduling strategies, resource allocation policies, and execution models tailored to hybrid computing infrastructures. A representative use case was presented to demonstrate the framework's capability to optimize both resource usage and workload distribution. Future extensions may incorporate advanced communication modeling, larger-scale heterogeneous device configurations, and integration with real-world execution traces [1, 34, 38] to further enhance the realism and applicability of the framework.

Code and Data Availability

The source code and integrated data used for the simulations and experiments in this study are publicly accessible in the GitHub repository [31].

Acknowledgments

This work was partially sponsored by NSF 2230111, 2238734, 2311950.

References

[1] D. Arthur and P. Date. 2022. A Hybrid Quantum-Classical Neural Network Architecture for Binary Classification. (2022).

[2] Ibrahim Attiya, Mohamed Abd Elaziz, and Shengwu Xiong. 2020. Job scheduling in cloud computing using a modified harris hawks optimization and simulated annealing algorithm. *Computational intelligence and neuroscience* 2020, 1 (2020), 3504642.

[3] H. Bian, J. Huang, J. Tang, R. Dong, L. Wu, and X. Wang. 2023. PAS: A new powerful and simple quantum computing simulator. *Software: Practice and Experience* 53, 1 (2023), 142–159.

[4] John Brennan, Lee J O'Riordan, K. G. Hanley, Myles Doyle, Momme Allalen, David Brayford, Luigi Iapichino, and Niall Moran. 2022. QXTools: A Julia framework for distributed quantum circuit simulation. *J. Open Source Softw.* 7 (2022), 3711. https://api.semanticscholar.org/CorpusID:246886257

[5] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. 2011. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience* 41, 1 (2011), 23–50.

[6] Rodrigo N Calheiros, Rajiv Ranjan, César AF De Rose, and Rajkumar Buyya. 2009. Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services. *arXiv preprint arXiv:0903.2525* (2009).

[7] Carmen Carrión. 2022. Kubernetes scheduling: Taxonomy, ongoing issues and challenges. *Comput. Surveys* 55, 7 (2022), 1–37.

[8] Tim Coopmans, Robert Knegjens, Axel Dahlberg, David Maier, Loek Nijsten, Julio de Oliveira Filho, Martijn Papendrecht, Julian Rabbie, Filip Rozpędek, Matthew Skrzypczyk, Leon Wubben, Walter de Jong, Damian Podareanu, Ariana Torres-Knoop, David Elkouss, and Stephanie Wehner. 2020. NetSquid, a NETwork Simulator for QUantum Information using Discrete events. *Communications Physics* 4 (2020), 1–15. https://api.semanticscholar.org/CorpusID:235967111

[9] Ran Das, Shubham Shenoy, Li Ding, and Ambarish Santra. 2024. Multiprogramming on quantum computers using a fair and reliable partitioning method. *Computers, Materials & Continua* 79, 2 (2024), 1961–1974.

[10] S. Diadamo, J. Notzel, B. Zanger, and M. M. Bese. 2021. QuNetSim: a software framework for quantum networks. *IEEE Transactions on Quantum Engineering* 2 (2021), 1–12.

[11] Y. Fan and Z. Lan. 2021. DRAS-CQSim: A reinforcement learning based framework for HPC cluster scheduling. *Software Impacts* 8 (may 2021), 100077.

[12] Richard P Feynman. 1986. Quantum mechanical computers. Vol. 16. 507–532.

[13] Constantin Gonzalez. 2021. Cloud based qc with amazon braket. *Digitale Welt* 5, 2 (2021), 14–17.

[14] Robert L Henderson. 1995. Job scheduling under the portable batch system. In *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 279–294.

[15] Leonid Hurwicz. 1973. The design of mechanisms for resource allocation. *The American Economic Review* 63, 2 (1973), 1–30.

[16] S. N. Agos Jawaddi and A. Ismail. 2024. Integrating OpenAI Gym and CloudSim Plus: A simulation environment for DRL agent training in energy-driven cloud scaling. *Simulation Modelling Practice and Theory* 130 (jan 2024), 102858.

[17] J.R. Johansson, P.D. Nation, and Franco Nori. 2013. QuTiP 2: A Python framework for the dynamics of open quantum systems. *Computer Physics Communications* 184, 4 (2013), 1234–1240. doi:10.1016/j.cpc.2012.11.019

[18] Tyson Jones, Anna Brown, Ian Bush, and Simon C Benjamin. 2019. QuEST and High Performance Simulation of Quantum Computers. *Sci Rep* 9, 1 (2019), 10736.

[19] Lei Liu and Xinglei Dou. 2021. Qucloud: A new qubit mapping mechanism for multi-programming quantum computing in cloud environment. In *2021 IEEE International symposium on high-performance computer architecture (HPCA)*. IEEE, 167–178.

[20] Waylon Luo, Betis Baheri, Travis Humble, Jiapeng Zhao, Tong Zhan, Rajan Maharjan, and Qiang Guan. 2025. A Digital Twin of Scalable Quantum Clouds. In *39th ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 165–175.

[21] Waylon Luo, Jiapeng Zhao, Tong Zhan, and Qiang Guan. 2025. Adaptive Job Scheduling in Quantum Clouds Using Reinforcement Learning. In *54th International Conference on Parallel Processing*.

[22] Hoa T. Nguyen, Muhammad Usman, and Rajkumar Buyya. 2023. iQuantum: A Case for Modeling and Simulation of Quantum Computing Environments. In *2023 IEEE International Conference on Quantum Software (QSW)*. 21–30. doi:10.1109/QSW59989.2023.00013

[23] Hoa T Nguyen, Muhammad Usman, and Rajkumar Buyya. 2024. QSimPy: A Learning-centric Simulation Framework for Quantum Cloud Resource Management. *arXiv preprint arXiv:2405.01021* (2024).

[24] Xiangyu Niu, Jianwei Li, Qing Wang, Jialin Yang, and Lei Liu. 2024. QuMC: Hardware-Aware Multi-Programming Compiler for Quantum Computing. *Computers, Materials & Continua* 79, 2 (2024), 1961–1974.

[25] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. 2014. A variational eigenvalue solver on a photonic quantum processor. *Nature communications* 5, 1 (2014), 4213.

[26] Kumar Prateek and Soumyadev Maity. 2023. Quantum programming on azure quantum—an open source tool for quantum developers. In *Quantum Computing: A Shift from Bits to Qubits*. Springer, 283–309.

[27] Konstantinos Rallis, Ioannis Liliopoulos, Evangelos Tsipas, Georgios D Varsamis, Nikolaos Melissourgos, Ioannis G Karafyllidis, Georgios Ch Sirakoulis, and Panagiotis Dimitrakis. 2025. Hardware-level Interfaces for Hybrid Quantum-Classical Computing Systems. *arXiv preprint arXiv:2503.18868* (2025).

[28] Subramaniam Ravi, Amandeep Singh, and Ravi Patel. 2023. Adaptive job scheduling framework for quantum cloud environments. *Journal of Quantum Computing* 78 (2023), 123–138.

[29] Heike Riel. 2022. Quantum computing technology and roadmap. In *ESSDERC 2022-IEEE 52nd European Solid-State Device Research Conference (ESSDERC)*. IEEE, 25–30.

[30] SimPy. 2024. Discrete event simulation for Python. https://simpy.readthedocs.io/en/latest/index.html.

[31] Quantum Cloud Simulation. 2025. Submission to SFWM Workshop at SC2025. https://github.com/QuantumCloudSimulation/HybridCloudSim.

[32] Damian Steiger, Thomas Häner, and Matthias Troyer. 2016. ProjectQ: An Open Source Software Framework for Quantum Computing. *Quantum* 2 (12 2016). doi:10.22331/q-2018-01-31-49

[33] Martin Suchara, Andrew Anderson, Harry Buhrman, and Andrew Schreier. 2024. QURE: Quantum Resource Estimator for Cloud Environments. *Computers, Materials & Continua* 79, 2 (2024), 1957–1974.

[34] J. Tilly, H. Chen, S. Cao, D. Picozzi, and K. Setia. 2021. The variational quantum eigensolver: a review of methods and best practices. *Physics Reports* (2021).

[35] Andrew Wack, Hanhee Paik, Ali Javadi-Abhari, Petar Jurcevic, Ismael Faro, Jay Gambetta, and Blake Johnson. 2021. Quality, Speed, and Scale: three key attributes to measure the performance of near-term quantum computers. (10 2021).

[36] Andrew C. Yao. 1979. Some Complexity Questions Related to Distributed Computing. *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing* (1979), 209–213. doi:10.1145/800135.804414

[37] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. Slurm: Simple Linux Utility for Resource Management. In *Workshop on Job Scheduling Strategies for Parallel Processing*. 44–60. doi:10.1007/10968987_3

[38] L. Zhou, S.-T. Wang, S. Choi, H. Pichler, and M.D. Lukin. 2020. Quantum Approximate Optimization Algorithm: Performance, Mechanism, and Implementation on Near-Term Devices. *Physical Review X* (2020).