A Simulation Framework for Workload Management in Quantum Cloud System

Waylon Luo
Kent State University
Kent, OH, USA

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

Tong Zhan
Meta Platform
Bellevue, WA, USA

Qiang Guan
Kent State University
Kent, OH, USA

ABSTRACT

The convergence of quantum computing and high-performance computing (HPC) is enabling hybrid cloud platforms that execute workflows spanning classical and quantum resources. These heterogeneous environments pose new challenges for workload management, including device variability, noise-prone quantum processors, and the need for coordinated 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 systems. We propose a taxonomy that categorizes approaches by scheduling strategy, allocation model, orchestration mechanism, and performance metric, and analyze how HPC practices can inform noise-aware, fidelity-preserving methods for quantum workloads. Key research gaps are identified in scalable orchestration, co-scheduling, and communication optimization, with future directions including AI-driven scheduling and error-mitigation integration. This work provides a foundation for advancing workload management in hybrid quantum–HPC cloud infrastructures.

1 INTRODUCTION

Advances in *quantum computing* and *high-performance computing (HPC)* are reshaping the computational landscape, enabling 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.

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.

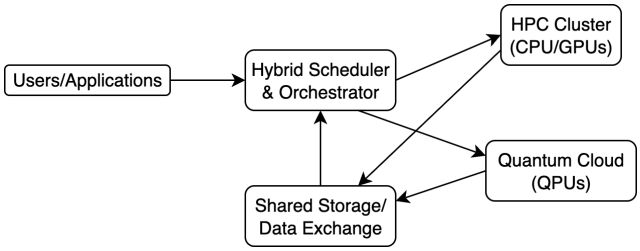

Figure 1: Conceptual architecture of hybrid quantum–HPC cloud workload management. A unified scheduler orchestrates job execution across classical HPC clusters and quantum processors, leveraging shared storage and communication channels.

While workload management in classical HPC clouds [8] and quantum computing platforms [3, 5] has been extensively 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. Our contributions are threefold: (1) a review of workload management strategies in classical, quantum, and hybrid environments, highlighting design principles and limitations; (2) a taxonomy categorizing approaches by scheduling strategy, allocation model, orchestration mechanism, and performance metric; and (3) the identification of future research directions, including AI-driven scheduling, fidelity-aware optimization, and scalable orchestration frameworks for heterogeneous environments.

2 BACKGROUND AND FOUNDATIONS

This section provides the necessary 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 Computing Overview

Quantum computing leverages the principles of quantum mechanics to perform computations beyond the capabilities of classical systems [7]. 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. These limitations constrain the execution of large, complex circuits, necessitating advanced error-mitigation techniques and circuit partitioning. Cloud-based access to quantum hardware through providers such as IBM Quantum [3], Amazon Braket [5], and Microsoft Azure Quantum [4] has accelerated research but also introduced additional challenges in scheduling, resource sharing, and managing performance variability.

2.2 High-Performance Computing Cloud Overview

High-performance computing (HPC) has long been the backbone of scientific computing, enabling large-scale simulations and data analytics. Modern HPC systems employ clusters of CPUs, GPUs, and other accelerators orchestrated by workload managers such as Slurm, PBS, and Kubernetes [8]. These schedulers rely on queue-based resource allocation, priority mechanisms, and parallel job execution to maximize throughput and utilization. The maturity of HPC workload management frameworks and their performance-oriented optimizations offer valuable insights for hybrid quantum–HPC scheduling, particularly in the areas of job prioritization, load balancing, and adaptive resource control.

2.3 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 offered by major cloud providers [4]. 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.

2.4 Workload Management Concepts

Workload management in hybrid quantum–HPC systems involves the coordinated execution of jobs across both classical and quantum resources. Scheduling determines the order and placement of jobs on available devices, often incorporating noise- and error-awareness to improve result quality. Resource allocation assigns computational and communication resources to meet job requirements efficiently, balancing competing demands across heterogeneous hardware. Orchestration ensures that hybrid workflows spanning multiple devices and environments execute correctly and efficiently, handling interdependencies between quantum and classical stages. Finally, performance metrics—such as runtime, throughput, fidelity, cost, and latency—are used to evaluate system effectiveness and guide optimization strategies. These concepts form the foundation of the taxonomy of workload management approaches discussed in the next section.

2.5 Simulation Frameworks for Quantum and Hybrid Clouds

Simulation plays a crucial role in the study of workload management for quantum and hybrid cloud environments. It enables researchers to explore scheduling policies, resource allocation strategies, and system scalability without incurring the high cost or limited availability of real quantum hardware. Digital twin approaches are particularly valuable, as they allow 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 [1]. QCloudSim provides configurable quantum devices, noise-aware execution models, and flexible scheduling policies, enabling the evaluation of workload management strategies under realistic conditions. Its modular architecture supports heterogeneous devices and can be extended to represent hybrid environments that combine quantum and classical resources.

Another recent contribution focuses on adaptive job scheduling in quantum clouds using reinforcement learning [2]. This work formulates the scheduling problem as a Markov Decision Process and leverages reinforcement learning to allocate quantum jobs efficiently while adapting to device noise and workload variability. The approach demonstrates improved performance over static and heuristic-based schedulers, highlighting the potential of machine learning methods for hybrid workload management.

These simulation-based frameworks provide a foundation for testing and validating scheduling algorithms in controlled environments, offering insights that can be transferred to real-world hybrid quantum–HPC platforms. They also complement existing HPC simulation tools by introducing models specific to quantum device behavior, error propagation, and quantum–classical interaction.

3 TAXONOMY OF WORKLOAD MANAGEMENT APPROACHES

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**. Figure 2 illustrates this taxonomy.

3.1 Scheduling Strategies

Scheduling determines how jobs are prioritized and mapped to computing resources. In hybrid environments, it must account for both classical workloads and quantum circuits while incorporating constraints such as QPU availability, calibration status, and error rates. Static scheduling approaches make decisions prior to execution, assigning resources according to predefined plans, a method common in batch HPC schedulers such as Slurm [8]. Dynamic scheduling adapts allocations at runtime by considering system state, job progress, and real-time noise data from QPUs. Heuristic-based 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 employs machine learning or reinforcement learning techniques to optimize decision-making under uncertainty.

3.2 Resource Allocation

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 associated with 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, in the case of quantum workloads, strategies to preserve fidelity across distributed execution.

3.3 Orchestration

Orchestration is the coordination of hybrid workflows that span both quantum and classical resources. This process must handle data dependencies, manage communication latency, and enable recovery from hardware or software failures. Commercial cloud platforms, such as Azure Quantum [4], already provide APIs for orchestrating hybrid workflows, but current research focuses on designing scalable orchestration layers capable of optimizing performance across distributed and heterogeneous infrastructures. Such orchestration frameworks must align task execution timelines, minimize idle periods for both QPUs and HPC nodes, and ensure seamless integration between classical and quantum computation stages.

3.4 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, offering insight into overall throughput. Fidelity preservation evaluates 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 introduced by inter-device data transfers, while cost and utilization metrics assess the efficiency of resource usage and the monetary implications in cloud-based settings. Together, these metrics provide a comprehensive basis for comparing different workload management strategies.

These four dimensions collectively define the design space for workload management in hybrid quantum–HPC clouds. The proposed taxonomy not only organizes existing approaches but also reveals potential research gaps and opportunities for innovation.

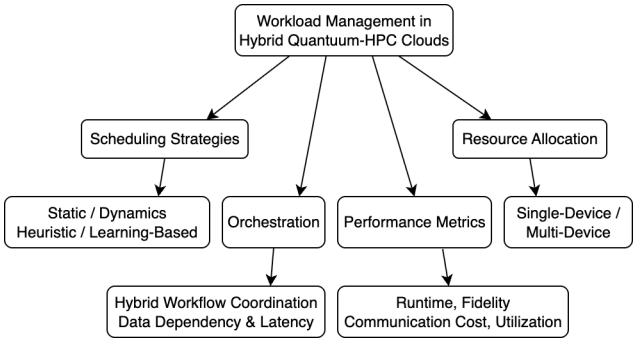

Figure 2: Proposed taxonomy for workload management in hybrid quantum–HPC clouds. Approaches are categorized along four key dimensions: scheduling, resource allocation, orchestration, and performance metrics.

4 WORKLOAD MANAGEMENT IN CLASSICAL HPC CLOUDS

High-performance computing (HPC) clouds have evolved over decades to deliver robust workload management capabilities, enabling the efficient scheduling and allocation of resources across large-scale clusters. These systems employ mature resource managers and job schedulers designed to optimize utilization while minimizing turnaround time. Understanding these classical approaches is essential, as many of their principles can be adapted to the unique 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 [8]. These schedulers handle the complete lifecycle of jobs, from submission to execution, by managing queues, prioritizing workloads, and allocating resources according to defined policies. Common strategies include First-Come-First-Serve (FCFS) scheduling, 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 the number of CPUs or GPUs, 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 by providing elastic resource management and fault-tolerant execution. These frameworks add the ability to dynamically scale resources, manage heterogeneous hardware, and maintain high availability—features that are 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 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

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

variable performance, limited availability, and periodic calibration requirements.

4.3 Lessons for Hybrid Quantum–HPC Workload Management

Several principles from classical HPC scheduling can inform the design of hybrid workload management systems. 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 enable 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.

4.4 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 can primarily focus on maximizing throughput and maintaining fairness. In contrast, quantum clouds consist of specialized noisy quantum processors (QPUs) that are constrained by limited qubit counts, hardware connectivity, and calibration-dependent noise characteristics. Here, scheduling policies must also account for 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 in these systems often involve alternating quantum and classical computation stages, where quantum subroutines are embedded within larger HPC pipelines. This integration requires coordination between fundamentally different execution models, making noise mitigation, inter-device communication, and synchronization critical for 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

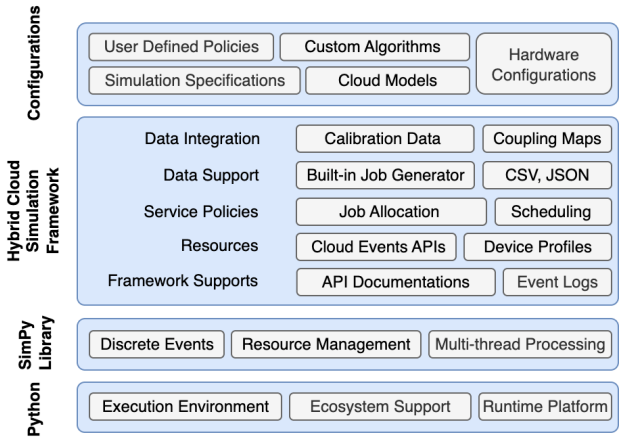

Figure 3: Layered architecture of HybridCloudSim, extending QCloudSim with hybrid quantum–classical simulation capabilities. The three layers comprise: SimPy Engine, HybridCloudSim Core, and Configurations.

The architecture of HybridCloudSim builds upon the layered design of QCloudSim (Fig. 3), extending its capabilities to model 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: As in QCloudSim, the foundational layer relies on the SimPy [6] 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 allows synchronous or asynchronous task execution between QPUs and CPUs, enabling accurate modeling of hybrid workloads that involve iterative quantum–classical feedback loops.

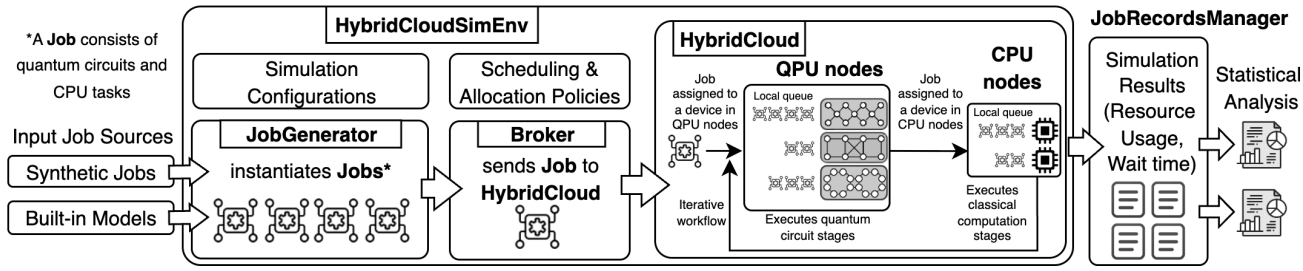

Figure 4: HybridCloudSim ecosystem and data flow. Jobs—hybrid workloads comprising quantum circuits and classical CPU tasks—are instantiated from synthetic datasets or built-in models by the `JobGenerator` under user-defined simulation configurations and policies in `HybridCloudSimEnv`. The `Broker` 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.

HybridCloudSim Core Layer: The HybridCloudSim core layer extends the `QCloudSim` functionality to support heterogeneous device orchestration. It introduces new device abstractions, such as `CPUDevice` (for high-performance classical processors) in addition to `QDevice` (for quantum processors), and integrates them into a unified `HybridCloudSimEnv` environment. This environment models hybrid workflows where quantum and classical tasks can be executed in parallel or in tightly coupled iterative sequences. The framework includes:

- A hybrid-aware `Broker` (e.g., `ParallelBroker`) for dispatching jobs to both QPUs and CPUs.
- A `HybridScheduler` that supports policies considering device type, workload dependency, and execution order.
- APIs for defining composite `HybridJobs` that consist of quantum and classical stages.

The core layer also manages device-specific constraints such as qubit topology, gate fidelities, and CPU computational throughput, enabling realistic modeling of end-to-end hybrid algorithms such as VQE or QAOA. It preserves QCloudSim's extensibility, allowing users to incorporate custom scheduling algorithms, hybrid partitioning strategies, and workload generators.

Configurations Layer: The configuration layer provides the user-facing interface for customizing simulation scenarios. It allows users to define hybrid workflows by specifying:

- Hardware configurations (e.g., number and type of QPUs and CPUs).
- Scheduling and allocation policies for both device classes.
- Inter-device communication models and synchronization rules.
- Input workloads, including synthetic job datasets and benchmark-driven hybrid tasks.

This layer makes HybridCloudSim adaptable to a wide range of research use cases, 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 `QDevice` class represents quantum processing units (QPUs) with configurable hardware properties such as qubit count, coupling maps, gate fidelities, and coherence times. These parameters may be defined synthetically or based on real hardware calibration data to achieve realistic performance modeling. The `CPUDevice` class models classical high-performance processors, described by attributes such as the number of cores, processing speed, and throughput. Classical devices are used for workloads such as numerical optimization, classical simulation, or data preprocessing and postprocessing.

Hybrid workloads are encapsulated in the `HybridJob` abstraction, which allows a job to consist of multiple execution stages of different types. Quantum stages are executed on QPUs, while classical stages are assigned to CPUs. Stage dependencies can be explicitly defined, enabling accurate simulation of hybrid algorithms such as the Variational Quantum Eigensolver (VQE), the Quantum Approximate Optimization Algorithm (QAOA), and quantum–classical neural networks.

Job dispatching is managed by a hybrid-aware broker, `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

the `HybridScheduler`, which determines 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. The simulation is configured through user-defined parameters, including the list of available devices, workload input files, broker and scheduler selection, and communication models. This modular design illustrated in Figure 4 allows HybridCloudSim to be adapted to a variety of research scenarios in hybrid quantum–classical cloud computing.

6 USE CASE DEMONSTRATION

To evaluate the performance of the proposed hybrid quantum-classical simulation framework, we conducted an experiment using a heterogeneous system configuration comprising two quantum processing units (QPUs) and two classical processing units (CPUs). Each QPU is modeled with a capacity of 128 qubits and 30,000 CLOPS, while each CPU is equipped with 100 cores and a memory bandwidth of 200 units. The scheduling policy allows for parallel execution, enabling multiple jobs to be processed concurrently across the 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: $[5, 15]$
- Circuit depth: $[5, 20]$
- Number of shots: $[5, 15] \times 100$
- Iterations per job: $[3, 7]$

6.1 Simulation Configuration

The simulation was executed using the `QCloud` package with the configuration shown in Listing 1. Two IBM QPU models (`IBM_Kawasaki` and `IBM_Kyiv`) and two generic CPU models were instantiated and registered in the hybrid simulation environment `HybridCloudSimEnv`, which employs a parallel broker to allow multiple jobs to be processed concurrently. Jobs were loaded from a synthetic CSV file and fed into the system using the `dispatcher` job feed method.

```
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("AMD_EPYC_9654-1", env=None)
AMD_EPYC_9654-2 = CPU("AMD_EPYC_9654-2", 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/10-job.csv',
    job_generation_model=None,
    printlog=PRINTLOG)

sim_env.run()
```

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

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 20 MB/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,
  'qpu_start': [1.0201, 6.6306, 11.4015],
  'devc_name': ['QPU-1', 'CPU-2', 'QPU-1', 'CPU-2', 'QPU-2',
      'CPU-2'],
  'qpu_arrive': [1.0201, 6.6306, 11.4015],
  'qpu_units': [6, 6, 6],
  'qpu_finish': [4.2385, 9.849, 14.5126],
  'cpu_start': [4.2385, 9.849, 14.5126],
  'cpu_arrive': [4.2385, 9.849, 14.5126],
  'cpu_units': [8, 4, 7],
  'cpu_mem_bw': [20, 20, 20],
  '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 5 shows the execution timeline for the first five 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:

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

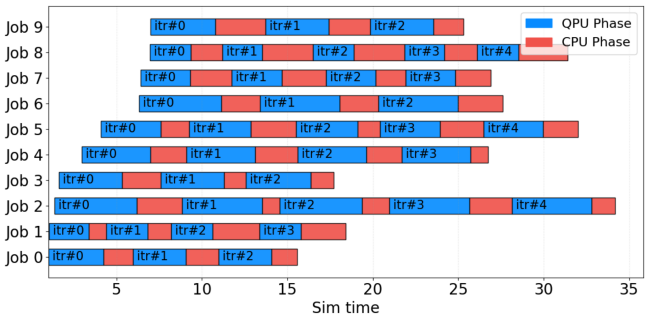

Figure 5: Gantt chart illustrating the hybrid QPU/CPU execution timeline for the first five jobs in the simulation. Each job is represented by a horizontal row, where blue segments correspond to QPU execution phases and red segments correspond to CPU execution phases. Iterations are labeled for clarity, 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 6. QPU utilization (blue) exhibits frequent peaks due to high concurrency and device contention, while CPU utilization (red) and memory bandwidth utilization (purple) reflect the post-QPU processing stages and memory-intensive workloads.

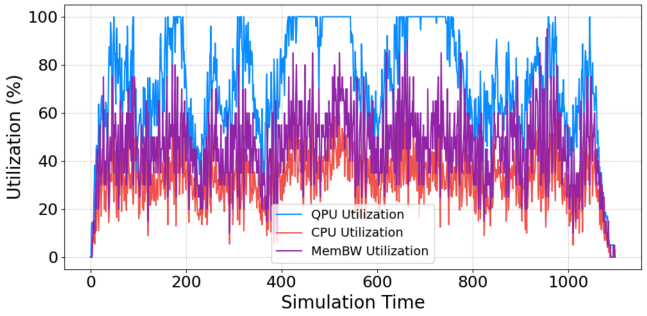

Figure 6: Time-series plot showing utilization trends for QPU, CPU, and memory bandwidth throughout the simulation. QPU utilization (blue) exhibits higher peaks due to parallelism and device contention, while CPU utilization (red) and memory bandwidth utilization (purple) reflect post-QPU processing 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 $\text{res} \in \{\text{QPU}, \text{CPU}, \text{MemBW}\}$, and let R_{cap} denote its maximum capacity. The utilization percentage is given by:

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

The computed values are clamped to the range $[0, 100]$ to ensure valid percentages and are plotted as a bar chart for visual comparison. This metric captures the proportion of total resource capacity actively used during the simulation and provides a direct measure of hardware load across different components.

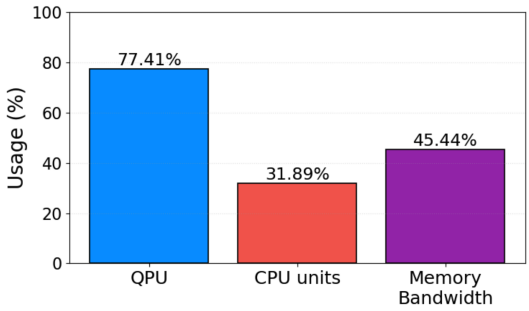

Figure 7: Overall average utilization of QPU, CPU units, and memory bandwidth across the simulation. QPU utilization reaches approximately 76.85%, CPU utilization is around 36.26%, and memory bandwidth utilization is approximately 45.32%. These values reflect the cumulative resource demands imposed by all jobs over the simulation period.

Figure 7 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

This work addresses the challenges of workload management and resource utilization in hybrid quantum–classical cloud environments. 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. By providing a configurable and extensible platform, the framework enables 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. The results highlight the potential of HybridCloudSim to serve as a valuable tool for evaluating and improving workload orchestration strategies in emerging hybrid cloud systems. Future extensions may incorporate advanced communication modeling, larger-scale heterogeneous device configurations, and integration with real-world execution traces to further enhance the realism and applicability of the framework.

ACKNOWLEDGMENTS

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

REFERENCES

[1] 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.

[2] Waylon Luo, Jiapeng Zhao, Tong Zhan, and Qiang Guan. 2025. Adaptive Job Scheduling in Quantum Clouds Using Reinforcement Learning. *arXiv preprint arXiv:2506.10889* (2025).

[3] IBM Quantum. 2023. IBM Quantum Cloud Services and Roadmap. In *IBM Quantum Summit*. Accessed: 2025-07-29.

[4] Microsoft Azure Quantum. 2022. Hybrid Quantum–HPC Workflows in Azure Quantum. In *Microsoft Quantum Computing Conference*. Accessed: 2025-07-29.

[5] Amazon Web Services. 2021. AWS Braket: A Fully Managed Quantum Computing Service. *AWS Quantum Computing* (2021). Accessed: 2025-07-29.

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

[7] 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. https://doi.org/10.1145/800135.804414

[8] 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. https://doi.org/10.1007/10968987_3