2025 International Conference on Quantum Communications, Networking, and Computing (QCNC) | 979-8-3315-3159-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/QCNC64685.2025.00018

QuCloudSim: A Customizable Discrete Event Simulator for Quantum Cloud Computing Enviroment

Ruilin Zhou, Yuhang Gan, Lucinda Shen, Yi Liu, Mignot Messel and Chen Qian
University of California, Santa Cruz

***Abstract*—Current quantum computing service providers operate in a cloud-based model, hosting expensive quantum computers in data centers for remote access. However, in the Noisy Intermediate-Scale Quantum (NISQ) era, quantum hardware is limited in terms of qubit count and connectivity. To address scalability challenges, future quantum cloud infrastructures will likely adopt a modular architecture where Quantum Processing Units (QPUs) are interconnected via quantum networks using various hardware technologies. In such systems, challenges like resource management, scheduling, and placement become critical, yet there is currently a gap in simulation tools capable of modeling quantum computers connected via quantum networks. To address this, we propose QuCloudSim, a quantum cloud simulator that adopts a modular design to model various hardware components within a quantum cloud environment. It employs discrete event simulation to replicate real-world scenarios with concurrent user requests, and our evaluation of different scheduling and placement methods demonstrates that QuCloudSim effectively models modular quantum cloud systems while providing valuable insights for developing scalable quantum cloud infrastructures.**

I. Introduction

Quantum computing has been considered a powerful candidate to advance the current classical computing paradigm, and it has shown potential to solve complex problems and provide speedups in areas such as factorization [1], quadratic speedup in database search [2], and simulations for physical sciences [3]. Current quantum computing techniques are situated in the Noisy Intermediate-Scale Quantum (NISQ) era [4], characterized by a limited number of qubits, restricted connectivity, and high levels of noise. These challenges significantly hinder the practical use of quantum computing, as it is generally estimated that millions of qubits would be required to execute useful quantum algorithms given current error rates [5].

Increasing the qubit count on a single quantum processor has proven difficult due to hardware constraints such as crosstalk errors, qubit addressability issues [6], and fabrication complexities [7]. Furthermore, these issues tend to exacerbate as quantum hardware scales [8]. An alternative approach to scaling quantum hardware, instead of placing all qubits on a single Quantum Processing Unit (QPU), is to adopt a modular architecture where multiple QPUs are interconnected via quantum networks and work collaboratively. In this modular design, each QPU contains a manageable number of qubits, and quantum interconnects facilitate communication between them, effectively increasing the total computational capacity while mitigating individual hardware limitations.

Another key observation is that current quantum service providers are delivering their services via cloud-based platforms, allowing users to access quantum computers remotely. IBM offers access to their quantum systems through the cloud, while AWS has partnered with companies like IonQ and QuEra to provide access to quantum hardware. It's therefore natural to envision that the future quantum cloud will evolve similarly to today's classical cloud platforms, where individual quantum processors (QPUs) are interconnected via quantum networks. This interconnected quantum cloud would facilitate distributed quantum computing, harnessing the collective power of multiple QPUs to tackle complex problems more efficiently and effectively. Recent advances in distributed quantum computing, quantum networks, and quantum interconnect technologies indicate that this vision may soon become a reality.

However, in such systems, resource management, placement, and scheduling present significant challenges—similar to those in classical cloud computing but amplified by the unique characteristics of quantum computing. Unlike classical systems, quantum computing operates under fundamentally different models and abstractions. Additionally, quantum hardware is scarce and prohibitively expensive, making direct experimentation impractical for most researchers and developers.

While significant progress has been made in quantum simulation—such as quantum network and quantum computing simulations—current quantum service providers offer limited access to real quantum computers. To overcome these challenges, it is essential to develop advanced simulation tools capable of modeling future quantum cloud environments. These tools will be instrumental in designing novel protocols and optimizing resource management at the system level.

By simulating quantum cloud systems, researchers can explore and address issues related to resource allocation, placement, and scheduling without the high costs and accessibility barriers of physical quantum computers. This approach enables rigorous testing and refinement of strategies in a controlled environment, paving the way for more efficient and scalable quantum cloud computing solutions.

Much progress has been made in quantum simulation, especially in quantum network hardware experiments and quantum computing simulations. However, neither of these can fully address the challenges associated with quantum cloud computing. Quantum network simulations focus on the communication

979-8-3315-3159-1/25/$31.00 ©2025 IEEE
DOI 10.1109/QCNC64685.2025.00018

aspects between quantum devices, while quantum computing simulations concentrate on computations within a single quantum processor. A quantum cloud simulator, on the other hand, requires an integrated approach that combines both quantum networking and quantum computing. More specifically, we need to simulate behavior of quantum processors, and the quantum networks that connect them—all while considering the resource constraints and scheduling complexities inherent in such systems. Only by integrating both sides can we explore and resolve the unique challenges of resource management, placement, and scheduling in quantum cloud computing.

In this work we make the first attempt to design such simulation tools. As we know our work is the first work to simulate a quantum cloud that bridges both quantum network and quantum computing. The implementation of our simulation is available on Github[1] . Our contributions are as follows:

- We propose QuCloudSim, a modular discrete-event simulator specifically designed for quantum cloud computing. QuCloudSim integrates both quantum computing and quantum networking simulations to model the complex interactions in a quantum cloud environment. We simulate various components in the quantum cloud, such as QPUs, computing qubits, communication qubits, and quantum links. We especially take the unique probabilistic nature of quantum networks into consideration.
- We develop and implement various resource management strategies within QuCloudSim, including different placement and scheduling methods for quantum tasks. This allows for comprehensive testing and analysis of these strategies in a simulated quantum cloud setting. To show the availability of our simulation tool. We conduct extensive experiments using QuCloudSim to evaluate the performance of different placement and scheduling algorithms.

II. Background

In this section, we briefly introduce the main background of our work.

A. Quantum Cloud Computing

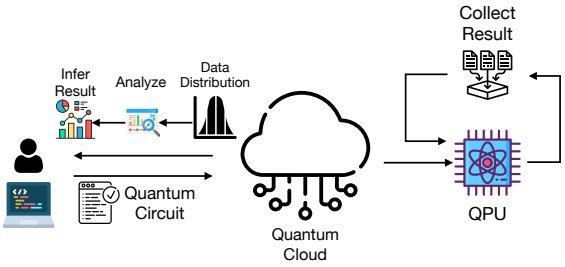

Fig. 1: Workflow of a Quantum Cloud Infrastructure

Current cloud service providers host their quantum computers in data centers, and users access these quantum computers through cloud interfaces. An example of the workflow is shown in Fig. 1. Users first encode their real-world problems into quantum circuits using quantum programming languages and frameworks such as Qiskit, Cirq. Then, users submit their quantum circuits to the cloud service provider via web portals, APIs, or SDKs provided by the quantum cloud platform. The cloud provider then schedules these quantum jobs for execution on the available quantum hardware or on users' predefined quantum hardware selections.

Once scheduled, the quantum cloud executes the circuits multiple times, a process referred to as "shots." Each shot represents a single execution of the quantum circuit, and multiple shots are necessary due to the probabilistic nature of quantum measurements. By running the circuit many times, users can gather a statistical distribution of the outcomes, which is essential for interpreting the results accurately. After execution, the final results—typically in the form of measurement data—are sent back to the users. Users then analyze the measurement distributions to infer solutions to their original problems, such as calculating probabilities, optimizing functions, or simulating quantum systems.

III. Design of QuCloudSim

A. Design Overview

The design of QuCloudSim's simulation framework is driven by the following principles:

- **Realistic Modeling**: Realistic modeling is a critical aspect of QuCloudSim, encompassing multiple facets to accurately simulate a quantum cloud environment. Firstly, we model the behavior of Quantum Processing Units (QPUs) by representing qubits with realistic parameters such as decoherence times, error rates, and gate fidelities, capturing the nuances of physical quantum hardware. Secondly, the execution of quantum circuits is simulated by incorporating gate operations, scheduling, and error propagation, reflecting the practical challenges in quantum computation. Thirdly, the probabilistic nature of quantum networks is modeled by simulating quantum channels with realistic loss and noise characteristics, entanglement generation rates, and quantum communication protocols like teleportation and entanglement swapping. Lastly, we adopt Discrete Event Simulation (DES) to efficiently model asynchronous events—such as gate executions, qubit decoherence, entanglement generation attempts, and classical communication—allowing for scalable and accurate simulation of complex quantum cloud operations. By integrating these elements, QuCloudSim provides a comprehensive and realistic simulation platform for exploring resource management, placement, and scheduling strategies in a distributed quantum computing environment.
- **Extensibility**: Extensibility is a fundamental design principle of QuCloudSim, enabling the simulator to adapt seamlessly to future advancements in quantum computing and networking. By employing a modular architecture, each component—such as QPU models, quantum network

[1]https://github.com/embersax/QuCloudSim.git

protocols, resource management strategies, and simulation engines—is designed to be easily extendable and replaceable. This modularity allows researchers and developers to incorporate new features, algorithms, and functionalities without altering the core framework or compromising existing components. As a result, QuCloudSim remains flexible and adaptable, fostering innovation and facilitating the exploration of emerging technologies and methodologies within the evolving landscape of quantum cloud computing.

- **Reusability and Flexibility**: Reusability and flexibility are integral to the design of QuCloudSim, enabling users to efficiently explore a wide range of resource management strategies. The simulator's modular architecture allows for the easy integration and interchange of components, such as scheduling algorithms and placement methods. This means that researchers can generate and test different scheduling policies or task placement strategies without altering the underlying framework. By reusing and customizing existing modules, QuCloudSim promotes efficient development and facilitates comparative analysis of various approaches. This flexibility not only accelerates experimentation but also ensures that the simulator can adapt to diverse research needs and evolving technologies within quantum cloud computing.

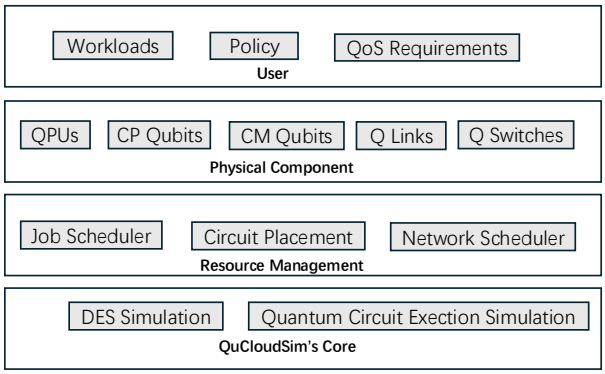

Fig. 2: QuCloudSim's Layers

We illustrate the layer structure of QuCloudSim in Figure 2, which comprises four main layers: the User layer, the Physical Component layer, the Resource Management layer, and QuCloudSim's Core layer.The User layer represents the interface where users interact with the simulator. This layer includes components such as workloads, which define the quantum computing tasks to be executed; policies that govern the operational parameters of the simulator; and Quality of Service (QoS) requirements that specify performance and reliability criteria for the quantum jobs. The Physical Component layer models the quantum hardware infrastructure. It encompasses Quantum Processing Units (QPUs) equipped with computing qubits for executing quantum circuits and communication qubits for enabling quantum networking. Additionally, this layer includes quantum links and quantum switches, which form the quantum network that interconnects the QPUs, allowing for entanglement distribution and quantum communication between different nodes. The Resource Management layer is responsible for the efficient allocation and scheduling of resources within the quantum cloud environment. It features components such as the job scheduler, which manages the queue of quantum tasks awaiting execution; circuit placement algorithms, which determine the optimal mapping of quantum circuits onto the available QPUs and qubits; and the network scheduler, which oversees the scheduling and management of quantum communication resources between QPUs. Finally, QuCloudSim's Core layer comprises the foundational simulation engines that drive the simulator. This includes the Discrete Event Simulation (DES) framework, which models the asynchronous and event-driven nature of quantum operations and communications, and the quantum circuit simulation module, which accurately simulates the execution of quantum circuits on the QPUs, including the effects of noise and errors. Together, these layers enable QuCloudSim to provide a comprehensive and realistic simulation of quantum cloud environments, facilitating the exploration and analysis of various resource management strategies and system architectures.

B. QuCloudSim's Modeling

An overview of QuCloudSim's quantum cloud modeling is depicted in Figure 3. In our representation, the future quantum cloud comprises a cluster of Quantum Processing Units (QPUs) interconnected by quantum links. Each QPU is equipped with both computing qubits for processing quantum circuits and communication qubits for facilitating quantum networking. Additionally, every QPU is connected to a central controller via classical links. The central controller is responsible for managing the quantum cloud by performing tasks such as circuit scheduling, circuit placement, and resource allocation. In Figure 3, we illustrate the interconnections between QPUs using a random graph. The number of outgoing links from each QPU is limited, reflecting the practical challenges associated with establishing quantum interconnects between QPUs.

1) QuCloudSim's Component: **Quantum Processing Unit**: In QuCloudSim, each Quantum Processing Unit (QPU) is formally modeled as a composite system comprising several key components. Firstly, it includes a set of computing qubits, denoted as $Q_{\text{comp}} = \{q_{\text{comp},1},\ q_{\text{comp},2},\ \ldots,\ q_{\text{comp},n}\}$, which are dedicated to executing quantum circuits. Secondly, it contains a set of communication qubits, represented as $Q_{\text{comm}} = \{q_{\text{comm},1},\ q_{\text{comm},2},\ \ldots,\ q_{\text{comm},m}\}$, used for quantum communication tasks such as entanglement generation and quantum teleportation between QPUs. The internal architecture of the QPU is defined by a topology graph $G = (V, E)$, where the vertices V are the union of computing and communication qubits, $V = Q_{\text{comp}} \cup Q_{\text{comm}}$, and the edges E represent the physical connections between qubits, indicating which pairs of qubits can directly perform two-qubit operations. Furthermore, each qubit $q_i \in V$ is associated with a single-qubit error

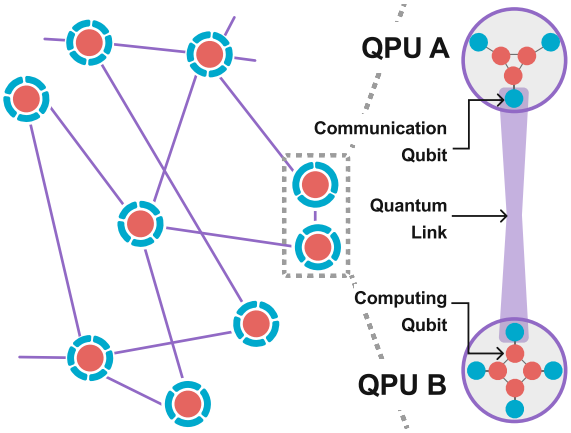

Fig. 3: Model of a Quantum Cloud Infrastructure

rate $\varepsilon_{1q}(q_i)$, representing the probability of error during single-qubit gate operations. Similarly, each connected pair of qubits $(q_i,\ q_j) \in E$ has a two-qubit error rate $\varepsilon_{2q}(q_i,\ q_j)$, indicating the error probability during two-qubit gate operations. This formal modeling captures the essential characteristics of a QPU, including its computational and communication capabilities, physical connectivity constraints, and realistic error profiles, thereby enabling accurate simulation of quantum cloud environments in QuCloudSim.

Quantum Link In QuCloudSim, quantum links are modeled to accurately represent the probabilistic nature of quantum communication between QPUs. Each quantum link connects a pair of communication qubits from two different QPUs and is characterized by several parameters. Formally, a quantum link L between QPU A and QPU B is defined as

$$L_{A,B} = \left(q^A_{\text{comm}},\ q^B_{\text{comm}},\ \eta,\ \gamma\right),$$

where q^A_{comm} and q^B_{comm} are the communication qubits in QPU A and QPU B, respectively. The parameter η represents the transmission efficiency of the quantum channel, accounting for losses due to factors like attenuation and scattering. The parameter γ denotes the error rate associated with quantum state transmission over the link, reflecting decoherence and other noise effects during communication. Additionally, the quantum link incorporates the entanglement generation rate r_e, which is the probability per unit time of successfully establishing entanglement between the connected communication qubits. This formalization allows QuCloudSim to simulate quantum communication processes such as entanglement distribution and quantum teleportation with realistic success probabilities and error characteristics, essential for analyzing the performance of distributed quantum algorithms and resource management strategies in the quantum cloud.

Modeling of Time and Event: Precise modeling of time and events is critical in QuCloudSim for two main reasons. First, both quantum networks and quantum computing operations are highly sensitive to timing. Accurate time modeling is essential to understand how quantum states evolve and potentially collapse due to temporal effects such as decoherence. Second, from a system-level perspective, quantum cloud service providers are concerned with metrics like utilization and throughput, while users focus on metrics such as latency, job completion time, and the fidelity of their workloads. All these performance indicators are closely related to time. Therefore, modeling time and events precisely is a key consideration in our design.

In QuCloudSim, we employ a discrete-event simulation (DES) approach to model time and events. Time advances in discrete steps, and all generated events are stored in a priority queue sorted by their scheduled timestamps. The simulation kernel continuously processes the event at the top of the queue, advancing the simulation clock to that event's timestamp and executing its associated actions. This procedure repeats until the priority queue is empty or a predefined simulation end condition is met. We model various events, such as quantum circuit executions, job generation, entanglement distribution, and distributed quantum circuit operations scheduled at specific times. This fine-grained modeling of time and events allows us to accurately represent the complex dynamics of a quantum cloud environment.

A critical consideration in our simulator is the trade-off between modeling granularity and computational resources. Modeling quantum circuit execution at an extremely fine granularity—for instance, at the level of individual gate operations with millisecond precision—would require tracking and simulating every minute detail of the system's state, leading to significant computational overhead. While this level of detail enhances accuracy, it may not be practical for large-scale simulations due to resource constraints. Therefore, QuCloudSim provides flexibility in adjusting the granularity of the simulation. Users can choose an appropriate level of detail that balances accuracy with computational efficiency, enabling the simulation of complex quantum cloud scenarios within reasonable time frames and resource usage. This approach ensures that the simulator remains both practical and precise, catering to a wide range of research needs.

2) Simulation on Distributed Quantum Computing: One important task of our simulation is to simulate the execution of a quantum circuit, especially a distributed quantum circuit, in the cloud setting. In QuCloudSim, we model DQC jobs using Directed Acyclic Graphs (DAGs) to capture the inherent parallelism and dependencies within quantum circuits.

A DQC job in QuCloudSim is represented as a DAG, where each node corresponds to a quantum gate operation, and the directed edges represent dependencies between these operations. This representation effectively captures the structure of quantum circuits, enabling the simulation of complex quantum workflows. The DAG allows us to identify parallelizable operations and enforce the correct execution order based on gate dependencies. Within a DQC job, gates are categorized into two types: local gates and remote gates. Local gates are gates that can be performed within a single quantum node without communicating with other nodes. On the other hand,

Remote Gates are gates that involve qubits that are distributed across different quantum nodes. Implementing remote gates requires entanglement generation between communication qubits on different nodes, and then quantum teleportation or cat-entangler and cat dis-entangler are used to perform remote gate operations. By distinguishing between local and remote gates, QuCloudSim can accurately model the communication overhead and gate dependency inherent in distributed quantum computations.

Modeling Dependencies and Execution Order The DAG structure inherently defines the dependencies between quantum gate operations, ensuring that a gate cannot commence execution until all its predecessor gates in the DAG have been completed. Currently, QuCloudSim does not account for the commutation rules of quantum gates, but this feature is planned for future updates. This dependency management ensures that the simulation adheres to the logical flow of the quantum algorithm being modeled. QuCloudSim leverages a discrete-event simulation (DES) framework to schedule and execute gate operations based on these dependencies, maintaining the integrity of the quantum computation process. Specifically, when a gate completes execution—represented by a node in the DAG—it is registered as an event in the DES, marking a specific time step for subsequent operations. This mechanism ensures that the simulation accurately reflects the temporal dynamics and execution order of the quantum circuit.

Modeling Remote operations and Local Operations: Remote gate operations in a Distributed Quantum Computing (DQC) job involve two major steps: entanglement generation between communication qubits and the execution of remote operations, which can be performed either through cat-entanglers or quantum teleportation. As introduced in earlier sections, entanglement generation between two Quantum Processing Units (QPUs) consists of probabilistic events, where both the generation rate and success rate are influenced by our previous modeling of quantum links. To maintain generality, remote gate operations in the DQC setting are also modeled as probabilistic events. When a remote gate operation is scheduled, QuCloudSim first attempts to generate the necessary entanglement between the communicating qubits. Given the probabilistic nature of entanglement generation, there is a certain probability that the attempt will succeed or fail. Furthermore, the probability of a successful attempt depends on the number of communication qubits being scheduled. If entanglement generation is successful, the simulation proceeds to execute the remote gate, incurring the associated time delay and resource usage. If entanglement generation fails, QuCloudSim can implement retry mechanisms based on predefined strategies.

Putting everything together, a local gate operation in QuCloudSim is modeled as a deterministic event with a fixed execution time, reflecting the predictable nature of operations confined within a single quantum node. In contrast, remote operations in QuCloudSim are modeled as probabilistic events. The duration and success rates of these remote operations are calculated based on parameters defined in the quantum network model, such as entanglement generation rates, fidelity of entangled states, and the efficiency of quantum communication protocols.

We illustrate this with an example of a Quantum Approximate Optimization Algorithm (QAOA) circuit snapshot and its corresponding Directed Acyclic Graph (DAG) in Fig. 4. In this example, the circuit is distributed across QPU A and QPU B. As shown in the figure, entanglement generation between the two QPUs is probabilistic, with a success rate p = 0.6 and a time delay dt = 0.05 ns for each attempt. Local operations, on the other hand, are deterministic, occurring within a single QPU with a delay of dt = 500 ns. This structure allows us to simulate both the computational and communication aspects of distributed quantum circuits, demonstrating the balance between local and remote operations.

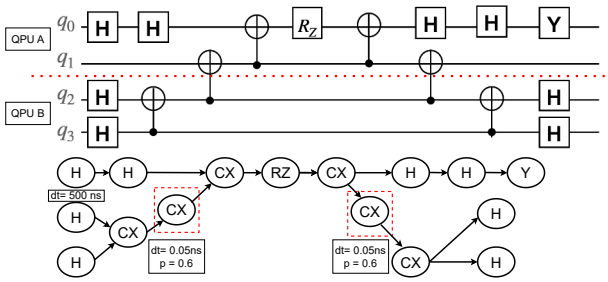

Fig. 4: Example DAG

C. Simulation Workflow

The simulation workflow of QuCloudSim orchestrates the end-to-end process of handling quantum computing jobs within a cloud environment. This workflow encompasses job submission, scheduling, circuit mapping, modeling circuit execution, and job completion, ensuring that quantum tasks are efficiently and accurately simulated. The detailed steps of the simulation workflow are outlined below:

1) **Job Submission**:
   - **User Interaction**: Users initiate the simulation by submitting quantum computing jobs to the cloud controller. Each job comprises one or more quantum circuits intended for execution on the quantum cloud infrastructure.
   - **Job Specifications**: Along with the quantum circuits, users may provide additional parameters such as the number of shots (i.e., the number of times the circuit is executed to gather statistical results) and Quality of Service (QoS) requirements, which may include constraints on latency, fidelity, or resource utilization.
   - **Job Queuing**: Submitted jobs are placed into a job queue managed by the resource management layer, awaiting scheduling and allocation.
2) **Job Scheduling**:
   - **Scheduling Algorithm Selection**: The job scheduler selects an appropriate scheduling algorithm based

on predefined policies or user-specified preferences. Scheduling algorithms may vary in their strategies, such as First-Come-First-Served (FCFS), Priority-Based Scheduling, or more sophisticated heuristic or optimization-based methods.
- **Priority Assignment**: Each job is assigned a priority level, taking into account factors such as job size, urgency (as dictated by QoS requirements), and current cloud resource utilization.
- **Scheduling Decision**: The scheduler determines the order in which jobs will be processed, ensuring that high-priority jobs are allocated resources promptly while maintaining overall system efficiency.

3) **Circuit Mapping**:
   - **Mapping Strategy Selection**: Based on the scheduling decision and the specific requirements of each job, the mapping algorithm determines how to assign logical qubits in the quantum circuit to physical qubits across different Quantum Processing Units (QPUs).
   - **QPU Evaluation**: The mapping algorithm evaluates the current status of available QPUs, including their computational load, available qubits, connectivity, and error rates, to identify suitable QPUs for circuit execution.
   - **Physical Qubit Assignment**: Logical qubits are assigned to physical qubits on the selected QPUs, considering the QPU's topology and error profiles to optimize performance and minimize error rates.
4) **Modeling Circuit Execution**:
   - **Quantum Circuit Simulation**: The mapped quantum circuits are simulated based on QuCloudSim's execution framework. This involves simulating gate operations, managing gate dependencies, and accounting for error propagation due to decoherence and gate imperfections.
   - **Discrete Event Scheduling**: Utilizing the Discrete Event Simulation (DES) framework, events corresponding to the execution of quantum gates and communication operations are scheduled. Each event is timestamped based on the estimated execution time and any required communication delays.
   - **Event Queue Management**: All scheduled events are inserted into a priority queue, sorted by their scheduled timestamps. The simulation kernel processes events sequentially, advancing the simulation clock to the time of the next event and executing its associated actions.
   - **Dependency Enforcement**: The execution of each event respects the dependencies defined in the Directed Acyclic Graph (DAG). An event representing a gate operation is only executed after all its predecessor events have been completed, ensuring the correct order of operations.
   - **Communication Strategy Implementation**: Depending on the user's specified methods for allocating communication resources, the simulator manages the distribution of quantum information between QPUs. This may involve protocols such as quantum teleportation or the use of cat-entanglers and cat disentanglers.
5) **Job Execution and Completion**:
   - **Gate Execution**: As events are processed, quantum gates are executed on the allocated QPUs. Local gate operations proceed deterministically with fixed execution times, while remote operations proceed based on the successful allocation of communication resources.
   - **Result Aggregation**: Upon completion of all gate operations within a job's DAG, the results (e.g., measurement outcomes) are aggregated. For jobs with multiple shots, the simulator repeats the execution process as necessary to gather statistical data.
   - **Job Completion Notification**: The simulator updates the job's status to completed, records relevant performance metrics (such as total execution time, resource utilization, and fidelity), and makes the results available to the user.

D. QuCloudSim's Application

QuCloudSim has a wide range of applications in advancing quantum cloud computing research and development. Specifically, it enables users to:

1) **Test Different Methods of Distributing Quantum Circuits**: Researchers can simulate various strategies for partitioning quantum circuits across multiple QPUs and evaluate the execution time and resource requirements for each approach. By analyzing different distribution methods, users can understand the trade-offs between communication overhead and computational efficiency.
2) **Explore and Compare Scheduling Methods**: QuCloudSim allows users to implement and compare different scheduling algorithms to determine their impact on job throughput, latency, and overall system performance. This helps in identifying the most efficient scheduling strategies for quantum cloud environments.
3) **Model New Hardware Components**: The simulator's extensible architecture permits the modeling of new hardware components. By adding new modules or updating existing ones, users can simulate novel quantum devices or network configurations within QuCloudSim. This flexibility enables the testing of new hardware designs and their integration into the quantum cloud infrastructure.

This makes QuCloudSim an invaluable tool for testing hypotheses, optimizing system designs, and accelerating the development of scalable and efficient quantum cloud infrastructures.

IV. Evaluation

To demonstrate the capabilities and usage of our simulator, we perform the following evaluations: **Testing different methods for placing a circuit on multiple QPUs** and **Analyzing**

resource allocation strategies for a distributed circuit. The first evaluation examines how the number of remote operations varies with different circuit placement methods, while the second studies how communication resource allocation strategies affect the completion time of a distributed quantum circuit.

For the quantum cloud setup, we configure a total of 20 QPUs, each equipped with 20 computational qubits and 5 communication qubits. We do not assume any specific network topology connecting the QPUs; instead, we adopt a random network topology with varying numbers of links for each QPU. The success rate for a single entanglement generation attempt is set to 0.5, which is the theoretical limit of current experiments. All tested circuits are gathered from [9].

A. Evaluation on Circuit Placement

We present the results of circuit placement in Table I. In our evaluation, we compare the following methods: Simulated Annealing (SA), Random Placement, Genetic Algorithm (GA), and Breadth-First Search (BFS). SA and GA are classical metaheuristic algorithms for optimization problems. Random Placement simply finds a valid placement for a circuit by randomly assigning qubits to QPUs. BFS uses a breadth-first search strategy to find a possible set of QPUs that can accommodate the circuit.

From the evaluations, we have the following observations: **BFS performs the best among the four methods.** This is because BFS ensures that frequently interacting qubits are placed on the same or adjacent QPUs when selecting QPUs, which minimizes remote operations. **GA outperforms SA in most circuits** since it is more effective at finding global minima. **Random Placement performs the worst** because it does not consider qubit interactions and merely finds a possible placement without optimization. **The performance gap between different methods varies across different circuits.** This variation highlights the heterogeneity of quantum circuits and their distinct traffic patterns. From this, we can infer that when distributing a circuit across multiple QPUs, dedicated methods tailored to specific types of circuits are necessary to improve performance.

B. Evaluation on Communication Resources Allocation

Another evaluation we perform is on communication resource allocation. This involves determining, for a given distributed quantum circuit, how many entanglement generation attempts we prepare for each remote operation, given the limited number of communication qubits on each QPU. As shown in Fig. 5, we choose three strategies: average allocation, random allocation, and greedy allocation. From the results, we observe that the greedy method performs the worst since it wastes unnecessary communication resources. The average-based method performs best on certain circuits because it can capture the structure of the circuit. From this evaluation, we conclude that the strategy for allocating communication resources is critical when executing a distributed quantum circuit, as intelligent allocation strategies can significantly enhance performance by optimizing the use of limited communication qubits and reducing resource wastage.

TABLE I: Number of Remote Operations

Circuit	SA	Random	GA	BFS
ghz_n127	145	161	90	10
bv_n70	41	38	17	26
bv_n140	96	98	54	101
ising_n34	38	36	6	2
ising_n66	100	110	36	6
ising_n98	214	250	96	10
cat_n65	52	44	20	5
cat_n130	153	145	92	10
swap_test_n115	398	472	294	352
knn_n67	158	230	106	168
knn_n129	528	720	374	376
qugan_n71	334	482	278	180
qugan_n111	838	1080	718	404
cc_n64	45	44	44	46
adder_n64	269	450	142	33
adder_n118	748	1225	613	60
multiplier_n45	596	1452	493	611
multiplier_n75	2100	6809	2255	1993
qft_n63	2504	3202	2368	3012
qft_n160	12326	15514	14246	14814

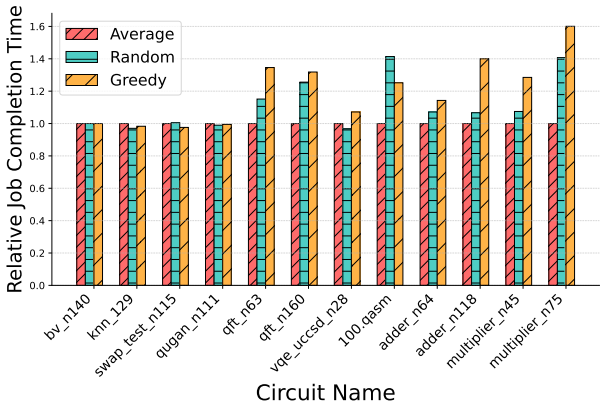

Fig. 5: Evaluation on Communication Resources Allocation Strategy on Different Circuits

However, designing and testing these strategies is complex due to the intricate nature of quantum networks and circuits, underscoring the necessity for advanced simulation tools. Our simulator addresses these challenges by providing a platform to model, test, and evaluate various resource allocation strategies in a controlled environment, allowing us to gain valuable insights into their performance impacts, guide the development of more efficient algorithms, and ultimately advance the field of distributed quantum computing.

V. Related Work

A. Quantum Network and Cloud Simulators

Several quantum network simulators address different aspects of quantum networking. **SeQUeNCe** [10] is a customizable, modular, discrete event simulator capable of modeling components from hardware to the control plane. It focuses on simulating single rare-earth ion memories and supports various quantum state encodings, including time-bin, polarization, and single-atom. **NetSquid** [11] is a modular simulator independent

of any specific network stack, offering kernel optimizations for scalability up to 1,000 nodes. It simulates devices like NV centers, atomic frequency combs, and electromagnetically induced transparency (EIT). **SimulaQron** [12] focuses on simulating the physical layer of quantum networks to facilitate the development and testing of quantum network applications and protocols, particularly at the application layer. **QuNetSim** [13] targets the simulation of higher-level quantum network protocols without emphasizing a specific physical model. **SQUANCH** [14] offers agent-based modeling with configurable error models but lacks a discrete event simulation kernel, limiting its ability to track time—crucial for studying generation rates and time-dependent noise affecting quantum state coherence. **QuISP** [15] emphasizes scalability by tracking error models of qubits rather than directly representing quantum states, allowing efficient simulations of large-scale quantum networks while providing insights into the effects of errors on system performance. **SimQN** [16] is a network-layer simulator that provides modular design with multiple physical backends, enabling efficient simulation and protocol evaluation while maintaining ease of configuration for quantum networks.**CFA** [17] presents a tensor network-based simulation method that optimizes control flow adaptations in quantum protocols, implemented as a module in ns-3 to achieve efficient large-scale quantum network simulations.

iQuantum [18], a holistic and lightweight discrete-event simulation toolkit, is tailored to model hybrid quantum computing environments for designing and evaluating quantum resource management policies. QSimPy [19], a discrete-event simulation framework, is designed to facilitate learning-centric approaches for quantum resource management problems in cloud environments. It provides a lightweight simulation environment based on SimPy [20], a well-known Python-based simulation engine for modeling quantum cloud resources dynamics and task operations. QSimPy's framework supports research in dynamic task allocation and optimization as well as the development and evaluation of reinforcement learning-based techniques for optimizing quantum cloud resource management.

VI. Conclusion

To conclude, we introduced QuCloudSim, a modular quantum cloud simulator designed to address the absence of tools capable of modeling quantum cloud environments where Quantum Processing Units (QPUs) are interconnected via quantum networks. QuCloudSim employs discrete event simulation to accurately replicate real-world scenarios with independent users submitting concurrent requests, allowing us to evaluate various scheduling and placement strategies under diverse workloads. Our results demonstrate that QuCloudSim effectively models the complexities of modular quantum cloud systems, providing valuable insights into resource management, scheduling, and placement—key challenges in developing scalable quantum cloud infrastructures. By bridging this gap, QuCloudSim not only aids researchers and developers in optimizing performance but also lays the groundwork for future advancements in scalable and efficient quantum computing services within cloud-based applications.

Acknowledgment

The authors were partially supported by NSF Grants 2322919, 2420632, 2426031, 2426940, 2114113, and DoE Grant DE-SC0022069.

References

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[2] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.

[3] R. P. Feynman, "Simulating physics with computers," in *Feynman and computation*. CRC Press, 2018, pp. 133–153.

[4] Y. Ding and F. T. Chong, "Quantum computer systems: Research for noisy intermediate-scale quantum computers," 2020.

[5] D. Monroe, "Building a practical quantum computer," *Communications of the ACM*, vol. 65, no. 7, pp. 15–17, 2022.

[6] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, "Trapped-ion quantum computing: progress and challenges," *Appl. Phys. Rev.*, 2019.

[7] M. Brink, J. M. Chow, J. Hertzberg, E. Magesan, and S. Rosenblatt, "Device challenges for near term superconducting quantum processors: frequency collisions," in *Proceedings of the IEEE International Electron Devices Meeting (IEDM)*, 2018.

[8] A. Wu, H. Zhang, G. Li, A. Shabani, Y. Xie, and Y. Ding, "Autocomm: A framework for enabling efficient communication in distributed quantum programs," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1027–1041.

[9] A. Li, S. Stein, S. Krishnamoorthy, and J. Ang, "Qasmbench: A low-level quantum benchmark suite for nisq evaluation and simulation," *ACM Transactions on Quantum Computing*, vol. 4, no. 2, pp. 1–26, 2023.

[10] X. Wu, A. Kolar, J. Chung, D. Jin, T. Zhong, R. Kettimuthu, and M. Suchara, "Sequence: a customizable discrete-event simulator of quantum networks," *Quantum Science and Technology*, vol. 6, no. 4, p. 045027, 2021.

[11] T. Coopmans, R. Knegjens, A. Dahlberg, D. Maier, L. Nijsten, J. de Oliveira Filho, M. Papendrecht, J. Rabbie, F. Rozpędek, M. Skrzypczyk *et al.*, "Netsquid, a network simulator for quantum information using discrete events," *Communications Physics*, vol. 4, no. 1, p. 164, 2021.

[12] A. Dahlberg and S. Wehner, "Simulaqron—a simulator for developing quantum internet software," *Quantum Science and Technology*, vol. 4, no. 1, p. 015001, 2018.

[13] S. DiAdamo, J. Nötzel, B. Zanger, and M. M. Beşe, "Qunetsim: A software framework for quantum networks," *IEEE Transactions on Quantum Engineering*, vol. 2, pp. 1–12, 2021.

[14] B. Bartlett, "A distributed simulation framework for quantum networks and channels," *arXiv preprint arXiv:1808.07047*, 2018.

[15] R. Satoh, M. Hajdušek, N. Benchasattabuse, S. Nagayama, K. Teramoto, T. Matsuo, S. A. Metwalli, P. Pathumsoot, T. Satoh, S. Suzuki *et al.*, "Quisp: a quantum internet simulation package," in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2022, pp. 353–364.

[16] L. Chen, K. Xue, J. Li, N. Yu, R. Li, Q. Sun, and J. Lu, "Simqn: A network-layer simulator for the quantum network investigation," *IEEE Network*, vol. 37, no. 5, pp. 182–189, 2023.

[17] H. Lin, R. Deng, C. Z. Yao, Z. Ji, and M. Ying, "Control flow adaption: An efficient simulation method for noisy quantum networks," *arXiv preprint arXiv:2412.08956*, 2024.

[18] H. T. Nguyen, M. Usman, and R. Buyya, "iquantum: A toolkit for modeling and simulation of quantum computing environments," *Software: Practice and Experience*, vol. 54, no. 6, pp. 1141–1171, 2024. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3331

[19] ——, "Qsimpy: A learning-centric simulation framework for quantum cloud resource management," 2024. [Online]. Available: https://arxiv.org/abs/2405.01021

[20] K. G. Müller, T. Vignaux, O. Lünsdorf, S. Scherfke, and K. Turner, "Simpy - discrete event simulation for python," 2024. [Online]. Available: https://simpy.readthedocs.io/en/latest/