iQuantum: A Case for Modeling and Simulation of Quantum Computing Environments

Hoa T. Nguyen[1], Muhammad Usman[2,3], and Rajkumar Buyya[1]

[1]Cloud Computing and Distributed Systems (CLOUDS) Laboratory,
School of Computing and Information Systems, The University of Melbourne, Parkville, 3052, Victoria, Australia
[2]School of Physics, The University of Melbourne, Parkville, 3052, Victoria, Australia
[3]Data61, CSIRO, Clayton, 3168, Victoria, Australia

Abstract—Today's quantum computers are primarily accessible through the cloud and are expected to be deployed in edge networks in the near future. With the rapid advancement and proliferation of quantum computing research worldwide, there has been a considerable increase in demand for using cloud-based quantum computation resources. This demand has highlighted the need for designing efficient and adaptable resource management strategies and service models for quantum computing. However, the limited quantity, quality, and accessibility of quantum resources pose significant challenges to practical research in quantum software and systems. To address these challenges, we propose iQuantum, a first-of-its-kind simulation toolkit that can model quantum computing environments for prototyping and evaluating system design and scheduling algorithms. This paper presents the quantum computing system model, architectural design, proof-of-concept implementation, potential use cases, and future development of iQuantum. Our proposed iQuantum simulator is anticipated to boost research in quantum software and systems, particularly in the creation and evaluation of policies and algorithms for resource management, job scheduling, and hybrid quantum-classical task orchestration in quantum computing environments integrating edge and cloud resources.

Index Terms—quantum computing, quantum cloud modeling, simulation, hybrid quantum computing, quantum job scheduling

I. INTRODUCTION

Quantum computing is an emerging field with tremendous potential to solve computationally intractable problems that can take years to solve with classical supercomputers [1]. Its applications are expected to revolutionize many fields, including drug discovery [2], finance [3], optimization [4], and machine learning [5]. However, quantum computing technologies are still under development, and there are numerous challenges in realizing its potential to bring advantages to industry and the practical world [6]. Furthermore, one of the primary challenges of quantum computing is its difficulty in operation, requiring extreme environmental conditions [7], specialized expertise, and massive investments in hardware.

Fortunately, the emergence and development of the cloud-based quantum computing (or quantum cloud) model provide a viable solution to make quantum computing accessible to the general public [8]. Quantum cloud is the intersection of quantum computing and cloud computing, allowing access to quantum computation resources over the cloud without significant upfront investment in quantum hardware [9]. Many giant cloud providers, such as Microsoft Azure Quantum, IBM Quantum, and Amazon Braket, have started offering quantum computing services. In light of this quantum service paradigm shift, we can anticipate significant progress in the field of quantum software engineering in the near future.

As the demand for quantum computing services has risen rapidly, efficient cloud systems and resource management strategies are becoming increasingly necessary for optimal quantum resource utilization [10]. While physical quantum hardware can be accessed through the cloud, its high cost and limited availability pose challenges for large-scale evaluation of resource management strategies and can lead to insufficient experimental validation. Therefore, developing a simulation framework to model hybrid quantum computing environments is crucial to support research in cloud-based quantum software and systems. However, existing quantum simulators primarily focus on simulating the physical operation of the quantum system (quantum operation) rather than modeling quantum systems and services in quantum computing environments. As a result, research in system design and resource management for quantum cloud computing remains challenging. Without quantum computing simulation and modeling tools, researchers will find evaluating their system design or job scheduling algorithms difficult or even unattainable in a realistic quantum computing environment. Besides, without the standard simulator, it can be complicated to reproduce experimental results or compare the performance of different algorithms or applications consistently and meaningfully.

To address these challenges, we propose *iQuantum*, a simulation framework for modeling quantum computing environments and facilitating resource management and system design. We propose the quantum system model and its resource management problems, which foster the development of a quantum modeling simulator. iQuantum will enable researchers to evaluate and validate their algorithms and applications in a simulated environment without the need for spending expensive costs to use real quantum hardware. Besides, it can empower research and experiment in quantum software engineering, making it easier to compare results and reproduce experiments for more rigorous and impactful studies aligned with the latest advances in quantum computing. This study aims to address two research questions:

- **RQ1.** *Can we model a quantum system in a cloud-based environment similar to modeling a classical paradigm?*

RQ1 aims to investigate the feasibility of modeling today's cloud-based quantum computing environments. This is the key question we have thoroughly studied before embarking on the prototyping and implementation of the simulator. After realizing the possibility of addressing this question, we continue to identify *what are the key elements of hybrid quantum system* and *how to model them*. The detailed results of our studies on these questions are presented in Section III and IV.

- **RQ2.** *Can we use the proposed simulator to model any use cases in practice*?
  RQ2 aims to identify potential use cases for the proposed simulator to consider its usefulness in quantum computing research. Our investigation to address this question is outlined in Section IV.

By addressing these research questions, we aim to provide viable insights into the potential of designing and developing the proposed simulator for quantum computing environments. The main contributions and novelty of our work are as follows:

- We derived the conceptual model for quantum computing environments using the various metrics of today's quantum computers and quantum job execution on the cloud. This model provides a theoretical basis for simulating and analyzing quantum computing systems, which has not been previously formulated in the literature.
- We proposed the architecture design and proof-of-concept implementation of core components in iQuantum based on CloudSim [11], which is a well-known simulator for classical cloud system modeling.
- We proposed various potential applications of iQuantum for modeling cloud-based quantum computing environments, which support research in system design and resource management, including quantum job scheduling, qubit mapping, and hybrid quantum-classical task orchestration. These applications demonstrate the practical usefulness of our proposal for researchers and practitioners in the field.

The rest of the paper is organized as follows: In Section II, we introduce the related background of our study, including the concept of cloud-based quantum computing, the QCaaS model, quantum applications, and performance metrics used to benchmark present quantum computers. In Section III, we propose the system model of a quantum computing environment. Then, we suggest the system design and core implementation of iQuantum to validate the proposed quantum system model in Section IV. In Section V, we illustrate the use of iQuantum simulator in creating a model and simulation of a quantum node and quantum tasks. We discuss potential use cases and extensions of iQuantum, including job scheduling, qubit mapping, and modeling hybrid quantum-classical cloud systems in Section VI. We review other related work in the literature in Section VII and conclude our study with future work in Section VIII.

II. BACKGROUND

A. Hybrid Quantum Computing and QCaaS Models

Cloud-based Quantum Computing (quantum cloud) is a paradigm in which quantum computation is performed on a quantum backend, such as a quantum computer or quantum simulator, over the cloud. These backends can be publicly accessible over the Internet (public quantum cloud) or exclusively accessible within a specific organization using a privileged network (private quantum cloud). QCC provides a means for users to execute quantum tasks without needing to manage their own quantum hardware [9]. Quantum cloud is currently the only viable way to access today's quantum computers, due to the stringent requirements [7] for operating a quantum system to avoid environmental interactions that can affect quantum execution. In an ideal scenario, the quantum cloud can be fully accommodated by a quantum internet [12], where all quantum data communications are processed through the quantum-based network without dependence on classical computers. However, the quantum internet is still in its early stage of development, with numerous challenges [13]. As a result, today's cloud-based quantum computers still rely on the classical internet and classical computers, which can be categorized as a hybrid quantum-classical cloud.
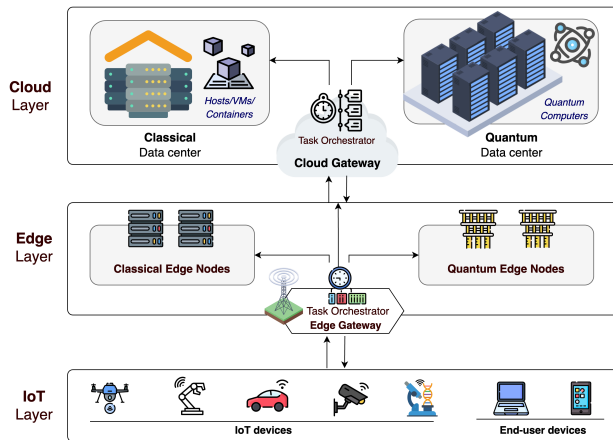

Fig. 1. A view of a Hybrid Quantum Computing Model

In the classical realm, edge computing has arisen to extend cloud computing capabilities to the edge network [14] and enable in-situ processing to reduce the network latency for time-sensitive applications [15]. As quantum computing matures, we anticipate witnessing the future hybrid quantum paradigm, in which the quantum computation node can also be placed in the edge layer, which is closer to the data source and Internet-of-Things (IoT) devices (see Figure 1). The realization of this complete paradigm depends on the advances in quantum hardware development and quantum error correction techniques. As recent years witnessed rapid growth and breakthrough in quantum hardware, we can begin to prepare the system prototype, software, and system algorithm for the near future of hybrid cloud-edge quantum computing [16].

In the current quantum cloud paradigm, Quantum Computing as a Service (QCaaS) [17] becomes a primary service model that provides access to quantum computation resources for executing quantum jobs. Cloud vendors offer this service model with a pay-per-use pricing model, where users only need to pay for the actual resources they consume on a per-time unit or per-task basis. Given those near-term quantum computers are subject to noise [18], quantum executions must be run multiple times (or shots) and corrected with error mitigation techniques to obtain better results. In addition to this, most variational quantum algorithms, such as Variational Quantum Eigensolver (VQE), must be executed multiple times (shots). Thus, the QCaaS model also considers the number of shots to determine the total cost of execution, and this characteristic should be included in the quantum service model.

B. Quantum Programs and their Execution in the Cloud

A quantum program comprises a sequence of primitive instructions that can be executed on a quantum computer [19]. There are two types of quantum programs: gate-based quantum circuits and annealing-based quantum programs [20]. This study considers the gate-based model, which most quantum hardware vendors, such as IBM Quantum, Google, IonQ, and Rigetti, have widely employed. Based on this model, a quantum circuit is built by performing quantum operations and measurements on qubits using different quantum gates [21].

As a quantum program cannot be deployed permanently on a quantum computer for multiple executions [22], the classical cloud backend is still required in this hybrid system to receive all user requests. Then, a job orchestrator classifies these tasks and forwards them to either classical or quantum computation nodes for execution. We refer to a quantum task that is sent to a cloud-based quantum computer for execution as a *"qulet"*, analogous to a classical task or *"cloudlet"* [11]. While isolated environments such as virtual machines (VMs) or containers can be created within classical servers, there are currently no equivalent techniques for creating quantum VMs or containers. Therefore, qulets must be sent to execute directly on quantum computers each time they are compiled from classical computers.

C. Quantum Computer Performance and Benchmarking

According to Wack *et al.* [23], three metrics are suggested for evaluating the performance of quantum computers: the number of qubits, quantum volume (QV), and circuit layer operation per second (CLOPS). The number of qubits determines the scale of a quantum system, *i.e.*, the amount of quantum information that can be processed. In the current era of noisy intermediate-scale quantum (NISQ) devices [18], the number of qubits is relatively limited, ranging from tens to several hundreds of qubits. For example, a recent IBM Quantum Osprey system supports 433 qubits[1], QuEra's quantum computer has 256 qubits[2], and IonQ has an 11-fully connected-qubit

quantum computer[3]. Apart from the number of qubits, QV and CLOPS are widely used by quantum hardware vendors, such as IBM Quantum and Rigetti, to benchmark the performance of gate-based quantum systems. Quantum volume (QV) is a metric to measure the quality of qubits of a quantum system. Cross *et al.* [24] defined the formula of quantum volume as $V_Q = 2^{\min(d,m)}$ where d and m are the depth and width of the largest square circuit ($d = m$) that can be precisely executed. Besides, CLOPS is the most recent proposed metric to measure the speed of a quantum system by determining how many QV circuits can be successfully executed per second [23]. CLOPS is empirically measured by $(M \times K \times S \times D)/time\_taken$ where $M = 100, K = 10, S = 100, D = \log_2 QV$ which stands for the number of templates, number of parameter updates, number of shots, and number of QV layers, respectively. Similar to FLOPS (floating point operations per second) or MIPS (million instructions per second) in classical computing, CLOPS is a potential metric that can be used to estimate the running time of a quantum job in a specific system.

III. MODELING QUANTUM COMPUTING ENVIRONMENTS

As cloud-based quantum computing is still an emerging paradigm, there is a lack of standards, models, and methodologies that can proficiently effectively optimize resource management while also dealing with intricate infrastructure and application challenges. In this section, we propose the first system model for quantum computing environments based on the characteristics of the state-of-the-art quantum computing paradigm. This system model facilitates the design and development of iQuantum and research in distributed quantum systems.

Figure 2 illustrates the overall quantum cloud service model, including the main components of the quantum cloud computing paradigm. The core components of the quantum cloud environment can be modeled as follows:

1) *Quantum Computation nodes* ($\mathcal{QC}$) are physical quantum computers located in a specific quantum data center and take the key responsibilities for executing quantum tasks. A quantum node can have single or multiple Quantum Processing Units (QPU) or quantum chips, which are the fundamental components for performing quantum operations. Each QPU contains a set of interconnected qubits with other control mechanisms to manipulate and measure them [25]. A QPU ($\mathcal{Q}$) can be modeled by a set of properties $\mathcal{Q}_i = \{q^w, q^v, q^s, q^g, q^t, q^e\}$, where:
   - q^w is the number of qubits, which indicates the scale of the quantum system.
   - q^v is the quantum volume of the system, which measures the quality of qubits and the system's capability to execute a quantum circuit faithfully.
   - q^s refers to the quantum computation speed of the system, which is measured by using CLOPS metric. The higher CLOPS, the faster the system

---

[1]https://spectrum.ieee.org/ibm-quantum-computer-osprey
[2]https://www.quera.com/aquila
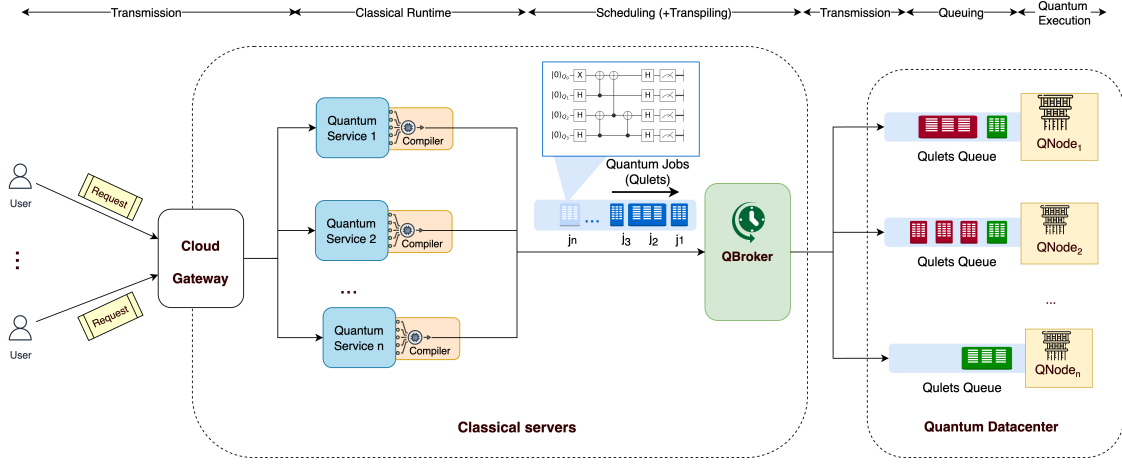[3]https://ionq.com/quantum-systems/harmony

Fig. 2. Overview of the System Model in Hybrid Quantum Computing Environments

can perform quantum computations and the more complex problems it can tackle.

- q^g is the list of the quantum gate sets supported by the quantum system.
- q^t is the qubit topology, which includes a list of all connections between pairs of qubits within the quantum chip. The qubit topology describes how the qubits are arranged and interconnected and is an important factor of the quantum chip's capabilities.
- q^e indicates the quantum system's error rates (such as readout and CNOT gate error). This parameter can help to demonstrate a complete picture of NISQ device performance. Indeed, NISQ devices are characterized by their limited number of qubits and high error rates, which can particularly affect the reliability of quantum computations.

Besides, unlike classical computing, quantum computing does not currently employ virtualization or containerization techniques. Instead, quantum computers directly execute quantum tasks without delegating to any virtual machines (VMs) or containers that might be allocated inside. As a result, there is no analogous conceptual model for VMs or containers in the quantum computing domain at this time.

2) *Quantum Data center* ($\mathcal{D}_{\mathcal{QC}}$) is a centralized hub that manages a set of quantum systems which are accessible through a classical cloud or edge computing system. The representation of a quantum datacenter is as follows: $\mathcal{D}_{\mathcal{QC}} = \mathcal{QC}_1, \mathcal{QC}_2, ..., \mathcal{QC}_n$, where $\mathcal{QC}_i$ represent a quantum computation node in the data center. These quantum nodes can be either homogeneous or heterogeneous based on physical quantum nodes' characteristics and underlying technology. Most quantum cloud providers also provide access to their quantum simulators. However, as they are classical-based resources that facilitate the testing phase in the NISQ era but not the production phases for the long term, we do not consider these simulators in the quantum computing model.

3) *Quantum Tasks (or qulet γ)* represent units of quantum computation that can be executed on a quantum computation node. The term *qulet* is analogous to *cloudlet* [11], which refers to a cloud-based task in classical computing. A qulet is defined by a set of attributes $\gamma_i = \{\gamma^a, \gamma^g, \gamma^w, \gamma^d, \gamma^s, \gamma^e, \gamma^t\}$, where:
- γ^a is the submission time of qulet
- γ^g is quantum gate set in the circuit, $j^g = G_s \cup G_m$ where G_s is a list of single-qubit gates and G_m is a list of multiple-qubit gates.
- γ^w is the number of qubits (circuit width).
- γ^d is the number of circuit layers (circuit depth).
- γ^s is the number of shots (*i.e.,* execution repetition).
- γ^t is the qubit topology in the circuit.

If the error rates are considered, each quantum task can be accommodated with the Quality-of-Service (QoS) metrics, which indicates the acceptable error threshold for its execution in the NISQ device.

4) *Quantum Computation Broker* (QBroker) is an intermediary entity between the cloud/edge servers and the quantum data center. Its primary responsibility is to schedule qulets to the most appropriate quantum computation nodes based on the qulets' properties and resource availability. This component's design and implementation of scheduling policies can be customized according to specific needs. By effectively scheduling qulets with the most suitable quantum computation nodes, the broker can minimize wait times and maximize resource utilization, thereby improving the overall performance and efficiency of quantum computing.

IV. IQUANTUM DESIGN AND IMPLEMENTATION

This section describes the design and proof-of-concept implementation of core components in our iQuantum simulator.

A. Architectural Design of iQuantum

We propose the layered design of iQuantum based on CloudSim [11], which is the most widely used simulation

framework for system modeling, designing, and evaluating scheduling algorithms for cloud computing environments. The extensions to CloudSim for creating the iQuantum simulator focus on adding quantum features to simulate the hybrid quantum computing environment (as the system model defined in Section III). The layered architecture design of iQuantum is illustrated in Figure 3.
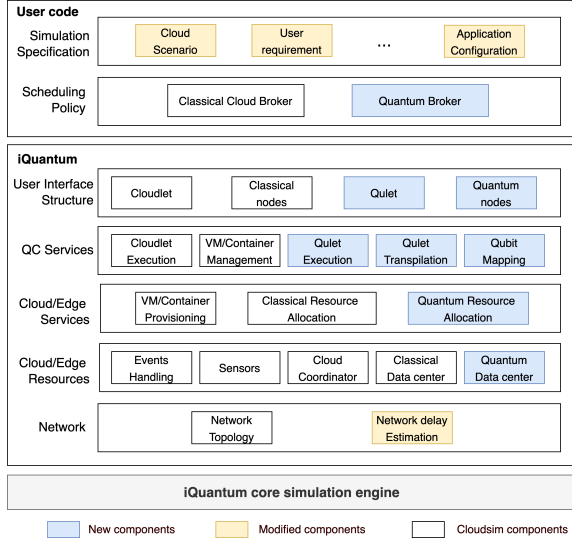

Fig. 3. Layered Design of iQuantum

The User Code layer allows users to define the specific scenarios, requirements, application configurations, and scheduling policies for classical and quantum jobs. Given the unique characteristics of quantum jobs (qulets), which differ significantly from classical jobs (cloudlets), new algorithms with specific constraints are needed to place qulets optimally on suitable quantum nodes. These algorithms should be designed and implemented in the quantum cloud broker.

The iQuantum simulation layers provide dedicated interfaces for modeling and simulating all core components, as defined in section III, including quantum data centers, quantum nodes, qulets, cloud services, and other resource allocation policies. The specific processes of quantum computing, such as qubit mapping and circuit transpiling, can affect the qulet execution procedure, and strategies to address these challenges can be defined in corresponding classes of iQuantum. Since quantum resources cannot be easily divided into smaller isolated parts, such as virtual machines or containers in classical computing, different resource allocation strategies must be designed to distribute resource usage among multiple users. Furthermore, quantum job execution is expected to incur more network delay, as intermediate transmissions from the classical cloud to the quantum data center are required. Therefore, network delay estimation will be modified to include all encountered delays.

B. Implementation of iQuantum Core Components

We implement the core components of iQuantum in Java based on the discrete-event simulation (SimEvent) technique

of CloudSim [11]. Figure 4 shows the overview of the main class diagram of our initial implementation with respect to the proposed quantum computing environment model (Section III). These classes are discussed below:
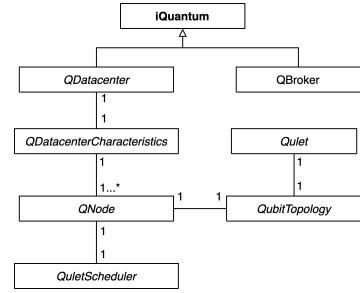

Fig. 4. Overview of iQuantum Main Class Diagram

1) `QDatacenter` class models the core infrastructure of a quantum data center, which consists of a collection of quantum nodes (`QNode`) used for the qulet execution. All configurations of the quantum data center, including the list of quantum nodes (`qNodeList`), are defined using the `QDatacenterCharacteristics` class.
2) `QDatacenterCharacteristic` class models the configurations and information of a quantum data center.
3) `QBroker` class models a broker that handles interactions between other components and the core simulator.
4) `QNode` class models a physical gate-based quantum computation instance. It contains single or multiple QPU, where each QPU has important metrics such as the number of qubits, quantum volume, CLOPS, gate set, qubit topology, and a scheduling policy.
5) `QubitTopology` class models the connectivity among all qubits of a quantum node or a quantum circuit in qulets. These topologies can be used to design a qubit mapping strategy for qulets to be mapped into a quantum node (discussed in Section VI-B).
6) `Qulet` class models a quantum task (or quantum circuit) to be sent to the `QBroker` for scheduling and execution in the appropriate `QNode`, following the scheduling policy defined in the `QuletScheduler` class.
7) `QuletScheduler` is an abstract class implemented by a `QNode` instance to model the scheduling policy for determining the share of quantum computation resources among multiple qulets. In the initial implementation, we design the `QuletSchedulerSpaceShared` provisioning policy, which allows only one qulet to run in a quantum node at a time, and it must finish execution before scheduling another qulet to the same node. This policy reflects the current situation of cloud-based quantum computing services. However, other policies, such as the Time Shared policy, can be implemented in futher development to allow multiple qulets to share the same quantum node and execute simultaneously.

V. A Sample Simulation using iQuantum

This section presents an explanatory example outlining the main steps of executing a simulation scenario in iQuantum. Figure 5 illustrates the simple scenario with two quantum tasks (qulets) for execution on one quantum node (single QPU with 7 qubits). We model the quantum node using the same metrics and qubit topology to IBMQ Oslo node[4].
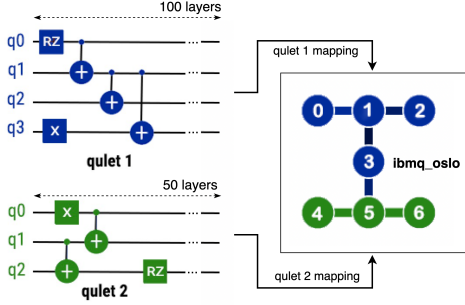

Fig. 5. A sample scenario for the simulation (*Left*: a part of quantum circuit in each qulet, *Right*: the qubit topology of the 7-qubit ibmq_oslo node)

Step 1. Initialize the core simulation instance using `iquantum.init()` function.

Step 2. Create `QNode` instances: The ibmq_oslo quantum node contains a single QPU with all the metrics as follows: quantum volume 32, CLOPS 2600, 5 basis gates (CX, ID, RZ, SX, X), and the qubit topology can be represented as $q_{oslo}^t = \{(0,1);(1,2);(1,3);(3,5);(4,5);(5,6)\}$. We models this `QNode` in iQuantum as follows (Code 1):

```
List<int[]> edges_oslo = new ArrayList<>();
edges_oslo.add(new int[]{0, 1});
edges_oslo.add(new int[]{1, 2});
... [truncated]
edges_oslo.add(new int[]{5, 6});
QubitTopology osloTpl = new QubitTopology(7,
    ↪ edges_oslo);
ArrayList<String> gateSet = new ArrayList<>(
    ↪ Arrays.asList("CX","ID","RZ","SX","X"));
QNode qNodeOslo = new QNode(7,32,2600, gateSet,
    ↪ osloTpl, new QuletSchedulerSpaceShared());
```

Code 1. Sample code for modeling a QNode (ibmq_oslo)

Step 3: Create a `QDatacenter` and add `QNode` instances to that `QDatacenter`. `qNodeList` and other information of the quantum datacenter are encapsulated in a `QDatacenterCharacteristics` object.

```
qNodeList = new ArrayList<QNode>();
qNodeList.add(qNodeOslo);
QDatacenterCharacteristics characteristics = new
    ↪ QDatacenterCharacteristics(qNodeList,
    ↪ timeZone, costPerSec);
QDatacenter qDatacenter = new QDatacenter("
    ↪ QDatacenter", characteristics);
```

Code 2. Sample code for modeling a single-node QDatacenter

Step 4: Create a `QBroker` instance and define a list of qulets to be executed (`quletList`). The qubit connectivity

in the quantum circuits of qulet 1 and qulet 2 are $\gamma_1^t = \{(0,1);(1,2);(1,3)\}$ and $\gamma_2^t = \{(0,1);(1,2)\}$, respectively. Both qulets employed 3 basic gates (CX, RZ, X), which is fully supported by the quantum node. Qulet 1 comprises 100 circuit layers and will be executed 4000 shots while these metrics for qulet 2 are 50 layers and 1000 shots. The `QBroker` and these qulets can be defined as follows:

```
QBroker qBroker = new QBroker("QBroker");
List<int[]> ql1Edges = new ArrayList<>();
ql1Edges.add(new int[]{0, 1});
ql1Edges.add(new int[]{1, 2});
... [truncated]
QubitTopology ql2Topology = new QubitTopology(3,
    ↪ ql2Edges);
ArrayList<String> qlGates = new ArrayList<>(
    ↪ Arrays.asList("CX", "RZ", "X"));
Qulet qulet1 = new Qulet(0, 5, 100, 4000, qlGates
    ↪ , ql1Topology);
Qulet qulet2 = new Qulet(1, 3, 50, 1000, qlGates,
    ↪ ql2Topology);
```

Code 3. Sample code for modeling a QBroker and a Qulet

Step 4: Design and implement the customized qulet scheduling policy. We implemented a simple `QuletSchedulerSpaceShare` with First Come First Serve (FCFS) scheduling policy by extending the `QuletScheduler` class. In the initial implementation, we estimated the approximate completion time (t^q) of a qulet inside a quantum node by implementing the following equation: $t^q = \frac{\gamma^d}{q^s} \times \gamma^s$ where γ^d is the number of circuit layers in the qulet, q^s is the CLOPS of the quantum node, and γ^s is the number of shots that qulet need to be executed. However, we acknowledge that other factors, such as transmission time, classical runtime, and queuing time, may also play a role in determining the scheduling policy. We plan to consider these factors and constraints in our future work on job scheduling (discussed in Section VI-A).

Step 5: Submit `quletList` to `qBroker` and start the simulation. Once all the simulation tasks are complete, stop the simulation and print out the final result. The simulator shows all events (with timestamps) happening during the simulation.

```
0.0: QBroker: Cloud Resource List received with 1
    ↪  resource(s)
0.01: QBroker : Started scheduling all Qulets to
    ↪ QDatacenter
0.01: QBroker: Sending Qulet 0 to QNode #0
0.01: QBroker: Sending Qulet 1 to QNode #0
153.86: QBroker: Qulet 0 result received
173.09: QBroker: Qulet 1 result received
173.09: QBroker: All Qulets executed. Finishing
```

Sample events in the simulation

As shown in the simulation results, two qulets are submitted to the QNode at timestamp t = 0.01s (minimum interval between 2 events). The execution time of qulet 1 and qulet 2 in the `QNode` are 153.85s and 19.23s, respectively. According to the Space Share scheduling policy, qulet 2 is executed after qulet 1 finish its execution. Therefore, the total execution time of all qulets is 153.85 + 19.23 = 173.08s. More complex scenario with advanced scheduling policies and qubit mapping

techniques (discussed in Section VI-B) can be implemented to allow concurrent execution of multiple qulets, optimize the total execution time and the resource utilization.

VI. POTENTIAL USE CASES AND EXTENSIONS

This section outlines potential applications and further extensions of iQuantum in modeling different quantum computing research problems, including quantum job scheduling, qubit mapping, and hybrid quantum-classical system.

A. Model and Design Quantum Job Scheduling Algorithms

Job scheduling (or task placement) is a critical aspect of distributed system research, including quantum computing, as it helps optimize resource utilization and minimize total job completion time. In a cloud-based quantum system, the responsibility of quantum job scheduling lies with the QBroker, which is tasked with finding the most suitable quantum system to execute a given qulet. To facilitate quantum job scheduling design, users can leverage iQuantum to generate cloud workloads and design and evaluate adaptable job scheduling algorithms for quantum cloud environments. By utilizing iQuantum, researchers can simulate various quantum computing scenarios and evaluate the effectiveness of different job scheduling strategies, helping to improve the performance and efficiency of the quantum computing system.
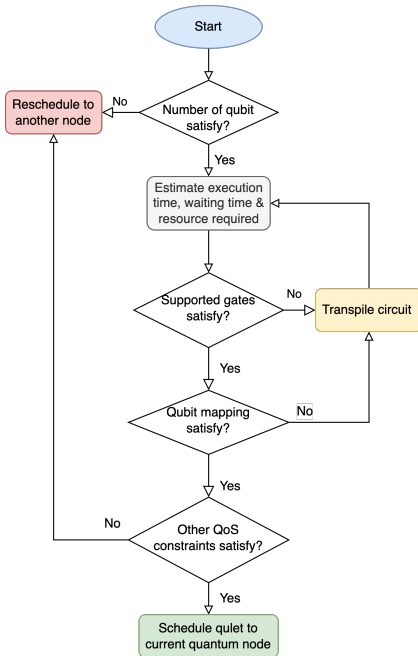

Fig. 6. Sample qulet scheduling logic with multiple constraints

A placement configuration of job $\gamma_i \in \Gamma$ can be define as $\sigma_i = \{\gamma_i, qc_k\}$ where $q_k \in \mathcal{QC}$ and $1 \leq k \leq |\mathcal{QC}|$ is the quantum computation node index. Several constraints must be considered when designing quantum job scheduling for specific situations. For example, requirements to place qulet γ_i to a target quantum computer q_T as follows (see Figure 6):

1) The total number of qubits of q_T must be greater or equal to the number of the required qubits to execute qulet γ_i: $q_T^w \geq \gamma_i^w$. This requirement must be satisfied when selecting a quantum computation node to execute a qulet. If the backend does not meet this requirement, the qulet must either be rescheduled to another quantum computation node or divided into multiple smaller circuits using techniques such as circuit cutting [26] to fit the capacity of the targeted quantum system.

2) To execute qulet γ_i on quantum computation node q_T, all gates used in γ_i must be supported by q_T, i.e., $\gamma_i^g \in q_T^g$. If an unsupported gate is detected, it must be decomposed into multiple native gates of the target machine (using circuit transpilation technique [27]). However, this approach may increase the cost of execution since the processing time may be longer.

3) The qubit connectivity in the circuit of γ_i must be a subset of the qubit topology of the target quantum system q_T, i.e., $\gamma_i^t \subset q_T^t$. If the connectivity does not meet this requirement, the input circuit must be transpiled and swapped to match the qubit order of the target quantum system. This process is known as qubit mapping and is described in more detail in Section VI-B.

4) Other quality of service (QoS) constraints must also be considered. For example, the estimated execution time t_i of qulet γ_i must be less than or equal to the expected completion time declared by users, i.e., $t_{\gamma_i} \leq \gamma_i^e$. Another constraint is to ensure that the quantum volume of the target quantum system q_T is large enough to execute the qulet faithfully and accurately.

To estimate the total execution time t_{γ_i} of a quantum job γ_i, multiple factors need to be assessed (as shown in Figure 2): $t_{\gamma_i} = t_{\gamma_i}^n + t_{\gamma_i}^c + t_{\gamma_i}^s + t_{\gamma_i}^w + t_{\gamma_i}^q$, where t^n is transmission times (or network delay), t^c is classical runtime to compile the quantum circuit, t^s is scheduling time (including transpilation time if necessary), t^w is the queuing time at the targeted quantum system, and t^q is the actual quantum execution time inside the quantum system.

The key objective of quantum job scheduling is to minimize the total execution time of all incoming jobs while optimizing the resource utilization of the whole system. As quantum computing is an emerging paradigm, job scheduling and resource allocation remain active research areas. Several studies have proposed strategies to tackle this issue. For example, Ravi *et al.* [28] proposed a statistical-based strategy by analyzing quantum job data over two years. Ngoenriang *et al.* [29] and Kaewpuang *et al.* [30] proposed stochastic resource allocation for distributed quantum computing under multiple uncertainties of circuit characteristics. Commercial platforms such as IBM Quantum currently use a fair-share algorithm to ensure fairness in using quantum resources for the general public[5]. As the quantum computing model becomes more well-defined and supported by simulators such as iQuantum, we anticipate a proliferation of new research endeavors focused on designing

---

[5]https://quantum-computing.ibm.com/lab/docs/iql/manage/systems/queue/

algorithms for job scheduling and resource allocation for quantum computing.

B. Model and Design Qubit Mapping Strategies

Qubit mapping (or quantum circuit mapping) is a mechanism that ensures the compatibility of quantum circuits with the target quantum computer [31]. As discussed earlier, this mechanism also plays an important role in quantum job scheduling. Each gate-based quantum computer can have a different qubit topology, which refers to the connectivity of all qubits due to different system designs and technologies used. For example, IBM Quantum Eagle chips have 127 partially connected qubits, while IonQ's quantum computer has 11 fully-connected qubits [32] (see Figure 7). A quantum circuit can only be executed if its qubit connectivity can be mapped to the qubit topology of the targeted quantum chip and if the target quantum system supports all single-qubit and multi-qubit gates in the circuit. Otherwise, circuit transpilation techniques can be used to transform the given circuit to match the qubit topology and gate set supported by the target quantum device [33]. Although qubit mapping becomes a challenging task when the number of qubits increases, it is an essential approach for optimizing the utilization of quantum cloud resources and enabling the concurrent processing of multiple quantum circuits in today's noisy quantum systems.
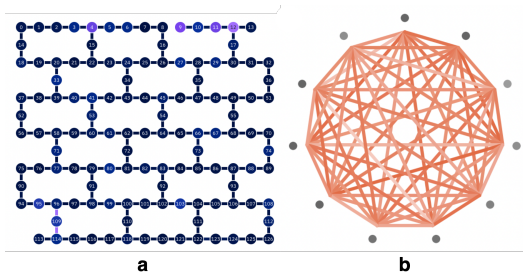

a   **b**

Fig. 7. Qubit Topology of (a) 127-qubit IBM Quantum Eagle (ibm_washington) and (b) 11-qubit IonQ's quantum computer [32].

We plan to expand the capabilities of iQuantum by enabling users to design and experiment with the circuit mapping technique. With this feature, users can explore various mapping strategies and evaluate their effectiveness in terms of reducing execution time and improving resource utilization on different quantum devices. For example, in our sample simulation scenario (see Figure 5), two qulets can be allocated simultaneously to *ibmq_oslo* to maximize the quantum resource utilization of this node and can reduce the total execution time. However, when the number of qubits increases and connectivity among them in both quantum systems and qulets becomes more complicated, the design of the qubit mapping algorithm for concurrent qulets becomes more challenging, as shown in some recent studies, such as [31], [34]. Hence, iQuantum can serve as the testbed environment for experimenting with new ideas in qubit mapping techniques.

C. Model Hybrid Quantum-Classical Computing Environments and Hybrid Task Orchestration Algorithms

Quantum computing is inevitably still in its infancy and cannot replace classical computing systems completely in the near future. Instead, quantum computing is expected to be a complementary technique for classical computing, where each technique will be to execute best-suited tasks to their respective advantages. Thus, the hybrid quantum-classical computing system is a potential approach to leverage the capabilities of both quantum and classical systems in the NISQ era [35]. This paradigm leverages the strengths of classical computing for tasks such as data pre-processing and post-processing while delegating more computationally-intensive tasks to quantum systems [21]. The outcomes of these quantum computations can then be combined with classical processing to obtain the final result. Figure 8 shows an example of hybrid quantum-classical execution logic.
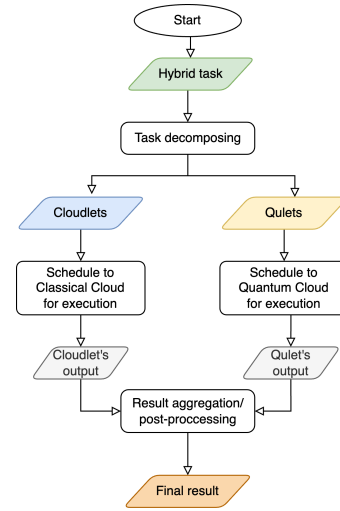

Fig. 8. An example of Hybrid Quantum-Classical Task Execution Logic

Each hybrid task comprises quantum and classical parts, which can be decomposed and distributed to different backends for execution. By building iQuantum on top of CloudSim [11], we can leverage the existing features of CloudSim to model classical cloud components, such as data centers, hosts, virtual machines, containers, and cloudlets (classical tasks). The remaining quantum components can be modeled using iQuantum features and seamlessly integrated with classical ones to form a hybrid system. We can design a hybrid task orchestration technique for scheduling both *cloudlets* and *qulets* to their respective backend to optimize resource utilization and total completion time for the whole system. As these cloudlets and qulets can be either independent or dependent on each other based on the application design, each task may need to wait for its dependent task to be complete before performing final aggregation or post-processing all outputs and generating the final result. iQuantum is expected to provide a simulation environment for modeling hybrid quantum-classical systems and testing new hybrid task orchestration for these systems.

VII. Related Work

Table I summarizes the overall comparison of our proposed simulator and existing works. As far as we know, no simulator can be used to model systems and tasks in hybrid quantum computing environments for designing scheduling and resource orchestration algorithms like our proposed simulator.

TABLE I
A FEATURE COMPARISON OVERVIEW OF RELATED WORKS WITH OURS

Simulators	Simulation Focus	Systems and Tasks Modeling		
		Quantum	*Cloud*	*Fog/Edge*
QuNetSim [36]	Quantum Network	E	×	×
NetSquid [37]	Quantum Network	E	×	×
QuEST [38]	Quantum Operation	E	×	×
QXTools [39]	Quantum Operation	E	×	×
PAS [40]	Quantum Operation	E	×	×
iQuantum *(Our proposal)*	System Modeling	S	✓	✓

✓: Supported; ×: Unsupported; E: Emulation; S: Simulation & Modeling (Cloud/edge features derived from CloudSim and iFogSim)

In the classical computing domain, CloudSim [11] is a popular simulator for modeling cloud computing environments. Similarly, iFogSim [41] is a well-known simulator for modeling edge and fog computing environments. However, these simulators do not support modeling quantum computing environments, as hybrid quantum computing involves the collaboration of quantum and classical resources. Therefore, this proposed iQuantum will help overcome the limitations of current modeling simulators. In fact, integrating iQuantum features in CloudSim and iFogSim will enable the modeling and simulation of quantum computing environments with seamless use of edge and cloud resources (i.e., both classical and quantum nodes) for real-time IoT applications. Regarding quantum computing environments, several studies have focused on quantum network protocol simulation. Diadamo *et al.* [36] proposed QuNetSim as a framework for simulating different quantum network protocols, such as quantum key distribution and quantum routing. As QuNetSim relies on other qubit simulators (such as SimulaQron [42], ProjectQ [43], and QuTiP [44]), its main objective is to develop quantum network protocol simulation rather than distributed quantum modeling. Similarly, NetSquid [37] is a discrete-event-based network simulator for modeling and simulating quantum network protocols. Jones *et al.* proposed QuEST [38] as another open-source software for simulating the behavior of quantum systems with high performance. QXTools [39] is another Julia-based framework for simulating distributed quantum circuits using the tensor networking approach. Also, Bian *et al.* [40] proposed PAS as a lightweight quantum simulator that works like other qubit simulators. For clarification, we categorized

these tools as *Emulation* frameworks as they support emulating the physical operation and functionality of actual quantum devices, while our framework focuses on modeling and simulation of quantum computing environments using the discrete-event simulation technique. Therefore, our new approach for the iQuantum simulator can empower researchers to conduct performance evaluation experiments of system design and resource management algorithms for quantum computing.

VIII. Conclusions and Future Work

In this paper, we proposed iQuantum as a first-of-its-kind simulator for modeling and simulation of quantum computing environments. We introduced the system model of key elements in quantum computing systems, including quantum data centers, quantum computation nodes, qulets, and quantum brokers. We also presented the proof-of-concept implementation and a simulation example to validate the proposed simulator. Furthermore, we discussed potential use cases and further development of iQuantum in terms of supporting research in algorithm development for quantum job scheduling, qubit mapping, and modeling hybrid quantum environments.

As quantum computing evolves, well-known service models and system algorithms are expected to become popular. We will improve our simulator to support these advances in quantum computing. For example, we will allow modeling multiple QPUs in each quantum node. Other characteristics of near-term quantum devices, such as noises, error correction, quantum network communications, and the pricing model, will also be addresse in the future development of iQuantum.

Software availability: The iQuantum software with the source code can be accessed from our website (clouds.cis.unimelb.edu.au/iquantum). Sample programs and tutorials illustrating the use of iQuantum are also available on the project website.

References

[1] S. S. Gill, A. Kumar, H. Singh, M. Singh, K. Kaur, M. Usman, and R. Buyya, "Quantum computing: A taxonomy, systematic review and future directions," *Software - Practice and Experience*, vol. 52, no. 1, pp. 66–114, 2022.

[2] M. Zinner, F. Dahlhausen, P. Boehme, J. Ehlers, L. Bieske, and L. Fehring, "Quantum computing's potential for drug discovery: Early stage industry dynamics," *Drug Discovery Today*, vol. 26, pp. 1680–1688, July 2021.

[3] P. Griffin and R. Sampat, "Quantum Computing for Supply Chain Finance," in *Proceedings of the 2021 IEEE International Conference on Services Computing (SCC)*, (Chicago, IL, USA), pp. 456–459, Sept. 2021.

[4] N. Moll, P. Barkoutsos, L. S. Bishop, J. M. Chow, A. Cross, D. J. Egger, S. Filipp, A. Fuhrer, J. M. Gambetta, M. Ganzhorn, A. Kandala, A. Mezzacapo, P. Müller, W. Riess, G. Salis, J. Smolin, I. Tavernelli, and K. Temme, "Quantum optimization using variational algorithms on near-term quantum devices," *Quantum Science and Technology*, vol. 3, p. 030503, July 2018.

[5] M. T. West, S. M. Erfani, C. Leckie, M. Sevior, L. C. L. Hollenberg, and M. Usman, "Benchmarking Adversarially Robust Quantum Machine Learning at Scale," Nov. 2022. arXiv:2211.12681.

[6] Quantum Technology and Application Consortium – QUTAC, A. Bayerstadler, G. Becquin, J. Binder, T. Botter, H. Ehm, T. Ehmer, M. Erdmann, N. Gaus, P. Harbach, M. Hess, J. Klepsch, M. Leib, S. Luber, A. Luckow, M. Mansky, W. Mauerer, F. Neukart, C. Niedermeier, L. Palackal, R. Pfeiffer, C. Polenz, J. Sepulveda, T. Sievers, B. Standen, M. Streif, T. Strohm, C. Utschig-Utschig, D. Volz, H. Weiss, and F. Winter, "Industry quantum computing applications," *EPJ Quantum Technology*, vol. 8, p. 25, Dec. 2021.

[7] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver, "A quantum engineer's guide to superconducting qubits," *Applied Physics Reviews*, vol. 6, p. 021318, June 2019.

[8] S. J. Devitt, "Performing quantum computing experiments in the cloud," *Physical Review A*, vol. 94, no. 3, pp. 1–13, 2016.

[9] F. Leymann, J. Barzen, M. Falkenthal, D. Vietz, B. Weder, and K. Wild, "Quantum in the Cloud: Application Potentials and Research Opportunities," in *Proceedings of the 10th International Conference on Cloud Computing and Services Science*, pp. 9–24, SCITEPRESS - Science and Technology Publications, Mar. 2020.

[10] G. S. Ravi, K. N. Smith, P. Gokhale, and F. T. Chong, "Quantum Computing in the Cloud: Analyzing job and machine characteristics," in *Proceedings of the 2021 IEEE International Symposium on Workload Characterization (IISWC)*, (Storrs, CT, USA), pp. 39–50, Nov. 2021.

[11] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, pp. 23–50, Jan. 2011.

[12] A. Singh, K. Dev, H. Siljak, H. D. Joshi, and M. Magarini, "Quantum Internet—Applications, Functionalities, Enabling Technologies, Challenges, and Research Directions," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2218–2247, 2021.

[13] S. Wehner, D. Elkouss, and R. Hanson, "Quantum internet: A vision for the road ahead," *Science*, vol. 362, no. 6412, 2018.

[14] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, "Survey on Multi-Access Edge Computing for Internet of Things Realization," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2961–2991, 2018.

[15] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, pp. 637–646, Oct. 2016.

[16] L. Ma and L. Ding, "Hybrid quantum edge computing network," in *Quantum Communications and Quantum Imaging XX*, (San Diego, United States), p. 29, SPIE, Oct. 2022.

[17] R. S. K. V and N. P. Kavya, "A Review on Progress and Problems of Quantum Computing as a Service (QCaaS) in the Perspective of Cloud Computing," *Global Journal of Computer Science and Technology: B Cloud and Distributed*, vol. 15, no. 4, 2015.

[18] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018.

[19] B. Heim, M. Soeken, S. Marshall, C. Granade, M. Roetteler, A. Geller, M. Troyer, and K. Svore, "Quantum programming languages," *Nature Reviews Physics*, vol. 2, pp. 709–722, Nov. 2020.

[20] P. Hauke, H. G. Katzgraber, W. Lechner, H. Nishimori, and W. D. Oliver, "Perspectives of quantum annealing: methods and implementations," *Reports on Progress in Physics*, vol. 83, p. 054401, May 2020.

[21] B. Weder, J. Barzen, F. Leymann, and D. Vietz, "Quantum Software Development Lifecycle," in *Quantum Software Engineering* (M. A. Serrano, R. Pérez-Castillo, and M. Piattini, eds.), pp. 61–83, Cham: Springer International Publishing, 2022.

[22] J. Garcia-Alonso, J. Rojo, D. Valencia, E. Moguel, J. Berrocal, and J. M. Murillo, "Quantum Software as a Service Through a Quantum API Gateway," *IEEE Internet Computing*, vol. 26, no. 1, pp. 34–41, 2022.

[23] A. Wack, H. Paik, A. Javadi-Abhari, P. Jurcevic, I. Faro, J. M. Gambetta, and B. R. Johnson, "Quality, Speed, and Scale: three key attributes to measure the performance of near-term quantum computers," Oct. 2021. arXiv:2110.14108 [quant-ph].

[24] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, "Validating quantum computers using randomized model circuits," *Physical Review A*, vol. 100, p. 032328, Sept. 2019.

[25] K. A. Britt and T. S. Humble, "High-Performance Computing with Quantum Processing Units," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 13, pp. 1–13, July 2017.

[26] W. Tang, T. Tomesh, M. Suchara, J. Larson, and M. Martonosi, "CutQC: using small Quantum computers for large Quantum circuit evaluations," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, (Virtual USA), pp. 473–486, Apr. 2021.

[27] C. Aravanis, G. Korpas, and J. Marecek, "Transpiling Quantum Circuits using the Pentagon Equation," Sept. 2022. arXiv:2209.14356 [quant-ph].

[28] G. S. Ravi, K. N. Smith, P. Murali, and F. T. Chong, "Adaptive job and resource management for the growing quantum cloud," in *Proceedings of the 2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pp. 301–312, IEEE, Oct. 2021.

[29] N. Ngoenriang, M. Xu, S. Supittayapornpong, D. Niyato, H. Yu, Xuemin, and Shen, "Optimal Stochastic Resource Allocation for Distributed Quantum Computing," Sept. 2022. arXiv: 2210.02886.

[30] R. Kaewpuang, M. Xu, D. Niyato, H. Yu, Z. Xiong, and J. Kang, "Stochastic Qubit Resource Allocation for Quantum Cloud Computing," Oct. 2022. arXiv: 2210.12343.

[31] L. Liu and X. Dou, "QuCloud+: A Holistic Qubit Mapping Scheme for Single/Multi-programming on 2D/3D NISQ Quantum Computers," arXiv:2207.14483.

[32] K. Wright, K. M. Beck, S. Debnath, J. M. Amini, Y. Nam, N. Grzesiak, J. S. Chen, N. C. Pisenti, M. Chmielewski, C. Collins, K. M. Hudek, J. Mizrahi, J. D. Wong-Campos, S. Allen, J. Apisdorf, P. Solomon, M. Williams, A. M. Ducore, A. Blinov, S. M. Kreikemeier, V. Chaplin, M. Keesan, C. Monroe, and J. Kim, "Benchmarking an 11-qubit quantum computer," *Nature Communications*, vol. 10, Dec. 2019.

[33] E. Younis and C. Iancu, "Quantum Circuit Optimization and Transpilation via Parameterized Circuit Instantiation," June 2022. arXiv:2206.07885.

[34] L. Liu and X. Dou, "QuCloud: A New Qubit Mapping Mechanism for Multi-programming Quantum Computing in Cloud Environment," in *Proceedings of the 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, (Seoul, Korea (South)), pp. 167–178, 2021.

[35] M. A. Serrano, J. A. Cruz-Lemus, R. Pérez-Castillo, and M. Piattini, "Quantum Software Components and Platforms: Overview and Quality Assessment," *ACM Computing Surveys*, 2022.

[36] S. Diadamo, J. Notzel, B. Zanger, and M. M. Bese, "QuNetSim: A Software Framework for Quantum Networks," *IEEE Transactions on Quantum Engineering*, vol. 2, pp. 1–12, 2021.

[37] T. Coopmans, R. Knegjens, A. Dahlberg, D. Maier, L. Nijsten, J. de Oliveira Filho, M. Papendrecht, J. Rabbie, F. Rozpedek, M. Skrzypczyk, L. Wubben, W. de Jong, D. Podareanu, A. Torres-Knoop, D. Elkouss, and S. Wehner, "NetSquid, a NETwork Simulator for QUantum Information using Discrete events," *Communications Physics*, vol. 4, no. 1, 2021.

[38] T. Jones, A. Brown, I. Bush, and S. C. Benjamin, "QuEST and High Performance Simulation of Quantum Computers," *Scientific Reports*, vol. 9, p. 10736, July 2019.

[39] J. Brennan, L. O'Riordan, K. Hanley, M. Doyle, M. Allalen, D. Brayford, L. Iapichino, and N. Moran, "QXTools: A Julia framework for distributed quantum circuit simulation," *Journal of Open Source Software*, vol. 7, p. 3711, Feb. 2022.

[40] H. Bian, J. Huang, J. Tang, R. Dong, L. Wu, and X. Wang, "PAS: A new powerful and simple quantum computing simulator," *Software: Practice and Experience*, vol. 53, no. 1, pp. 142–159, 2023.

[41] R. Mahmud, S. Pallewatta, M. Goudarzi, and R. Buyya, "iFogSim2: An extended iFogSim simulator for mobility, clustering, and microservice management in edge and fog computing environments," *Journal of Systems and Software*, vol. 190, no. 111351, pp. 1–17, 2022.

[42] A. Dahlberg and S. Wehner, "SimulaQron - A simulator for developing quantum internet software," *Quantum Science and Technology*, vol. 4, no. 1, pp. 0–15, 2019.

[43] D. S. Steiger, T. Häner, and M. Troyer, "ProjectQ: An Open Source Software Framework for Quantum Computing," *Quantum*, vol. 2, p. 49, Jan. 2018.

[44] J. Johansson, P. Nation, and F. Nori, "QuTiP: An open-source Python framework for the dynamics of open quantum systems," *Computer Physics Communications*, vol. 183, pp. 1760–1772, Aug. 2012.