CloudQC: A Network-aware Framework for Multi-tenant Distributed Quantum Computing

Ruilin Zhou*, Yuhang Gan*, Yi Liu, and Chen Qian
University of California, Santa Cruz

***Abstract*—Distributed quantum computing (DQC) that allows a large quantum circuit to be executed simultaneously on multiple quantum processing units (QPUs) becomes a promising approach to increase the scalability of quantum computing. It is natural to envision the near-future DQC platform as a multi-tenant cluster of QPUs, called a Quantum Cloud. However, no existing DQC work has addressed the two key problems of running DQC in a multi-tenant quantum cloud: placing multiple quantum circuits to QPUs and scheduling network resources to complete these jobs. This work is the first attempt to design a circuit placement and resource scheduling framework for a multi-tenant environment. The proposed framework is called CloudQC, which includes two main functional components, circuit placement and network scheduler, with the objectives of optimizing both quantum network cost and quantum computing time. Experimental results with real quantum circuit workloads show that CloudQC significantly reduces the average job completion time compared to existing DQC placement algorithms for both single-circuit and multi-circuit DQC. We envision this work will motivate more future work on network-aware quantum cloud.**

I. Introduction

Quantum computing technologies have shown great potential to solve complex problems and provide speedups such as quadratic speed up in database search [1], and simulations for physical sciences [2]. Current quantum computing techniques are in the Noisy Intermediate-Scale Quantum (NISQ) era [3], characterized by limited qubits, limited connectivity, and prevalent noise. These challenges greatly hinder the usage of quantum computing since it is commonly assumed that at least millions of qubits are required according to the current qubits error rates to execute a practical quantum algorithm [4]. Recent advances show that increasing the number of qubits on a single quantum processor is challenging due to hardware limitations such as crosstalk errors [5], qubit addressability [5], and fabrication difficulty [6]. More importantly, the significance of these challenges usually increases with the size of quantum hardware [7]. Instead of waiting for a powerful single quantum computer, a promising approach to increase scalability is distributed quantum computing (DQC) [7]–[15]. DQC makes use of the resources of multiple quantum computers to perform large quantum computing jobs by separating each job onto different quantum computers and using inter-processor communication [8]. Quantum computers working on the same DQC task rely on a quantum network to exchange quantum bits (qubits) [7]–[11], [13]–[16].

It is natural to envision the near-future QDC platform as a shared cluster of quantum computers [17], [18], called a *Quantum Cloud*, similar to today's cloud computing. This is because quantum hardware is more expensive for individual users. For example, industry companies like IBM [19], Microsoft [20], and NVIDIA [21] have already launched their quantum cloud services. Recently developed quantum multi-programming [18], [22], [23] enables multiple quantum computing jobs (also called as quantum circuits) to be executed on the same quantum computer simultaneously. This technology further allows a multi-tenant quantum cloud that supports DQC to become a reality in the near future.

One fundamental research problem of DQC is to divide a quantum computing job, represented by a *quantum circuit*, into sub-circuits and place them onto different quantum computers such that each quantum computer has sufficient resources to assign to these sub-circuits [7], [9], [11], [24]. Finding a placement that optimizes inter-quantum-computer communication is crucial because sub-circuits on different quantum computers will cause quantum network communication that spends Einstein-Podolsky-Rosen (EPR) pairs (also called *entanglements*), whose generation is known to be expensive [7], [9], [11], [24]. However, the prior solutions [7], [9], [11], [24] focus on placing a single circuit, and none considers a multi-tenant cloud environment, where multiple users request their quantum circuits to be executed in the cloud simultaneously.

We envision that circuit placement in a multi-tenant cloud will become a new line of research because 1) it aligns with the trend of developing a high-throughput, low-latency, quantum cloud; 2) this problem *significantly differs* from existing research on placing one single circuit in DQC and that on placing virtual machines (VMs) in classic cloud computing.

Circuit placement in a multi-tenant cloud cannot directly use single-circuit solutions [7], [9], [11], [24] because, 1) an optimized placement for one circuit could leave difficult remaining resource to place other circuits. Hence, the process of placing one circuit should include the consideration of whether the remaining resource is difficult for other placements; 2) prior solutions do not consider scheduling network resources (EPR pairs) among different circuits, while network resource contention should be a concern in a multi-tenant cloud.

On the other hand, existing network optimization methods for VM placement in a classic cloud [25]–[27] do not fit into a quantum cloud either. There are two main reasons: 1) different separations of the same quantum circuit will cause significantly different communication costs, while the network traffic between different VMs will not change a lot by their placement; 2) one key feature of quantum network operations

[1]Authors contributed equally to this research.

is its probabilistic nature [28]. EPR generation does not always succeed. Hence the quantum communication in DQC could fail even when sufficient resource is allocated. In a classic cloud, network communication is assumed to be successful if the bandwidth allows.

To our knowledge, there is no existing work that has addressed the above concerns for a multi-tenant quantum cloud. **This work is the first attempt to design a circuit placement and resource scheduling framework for a multi-tenant quantum cloud. Regardless of how the exact quantum computing technologies will evolve, the research problem studied in this work is a necessary step for multi-tenant quantum computing platforms.**

The proposed framework is called CloudQC, which includes two main functional components: circuit placement and network scheduler. The circuit placement function allows the quantum cloud to assign resources to quantum circuits as a bunch (at system initialization) or a sequence (when requests from users come sequentially). The placement of one circuit will optimize the remaining resources for other circuits. The network scheduler function assigns network resources (EPR pairs) among different sub-circuits. For important quantum gates, CloudQC will assign them redundant network resources to avoid potential backlogs that could cause long latency for the entire circuit. CloudQC can also be used for single-circuit placement and it improves existing methods. The implementation of our simulation is available to the public [29].

Our main contributions are summarized as follows:

- We are the first to model a multi-tenant quantum cloud architecture that supports multiple QDC jobs and optimizes both quantum network cost and quantum computing performance.
- With the goal of improving both utilization, throughput, and performance of the quantum cloud, we propose new placement and scheduling methods that allow quantum circuits to be executed on multiple QPUs. We design a network scheduler that considers the quantum network's probabilistic nature and allocates communication qubits to each remote quantum gate.
- Simulation results with real quantum circuit workloads show that CloudQC significantly reduces the job completion time compared to existing DQC placement algorithms **for both single-circuit and multi-circuit DQC**.

The rest of the paper is organized as follows: we first introduce necessary background knowledge. Then, we will introduce how we model components in our proposed quantum cloud and how we model essential quantum operations in our work.

II. Background

Qubits and entanglement. A quantum bit (qubit) is the fundamental unit in quantum computing and quantum networks. A qubit can be in the superposition state of 1 and 0 [30]. Such a state can expressed mathematically as: $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$. Upon the measurement on a qubit, the quantum state of the qubit will become either state $|0\rangle$ or state $|1\rangle$ with probability $|\alpha|^2$ or $|\beta|^2$. An important feature of the quantum state is called an entanglement of two qubits. Two entangled qubits are called an EPR pair [30], which is a fundamental unit in quantum communication.

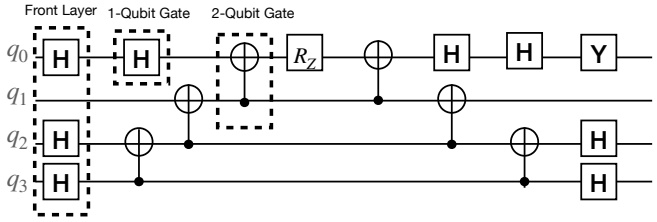

Fig. 1. Quantum circuit of a 4-qubit VQE algorithm

Quantum gates and quantum circuits. Most quantum algorithms adopt a *quantum circuit* model, expressing quantum operations as *quantum gates*. Mathematically, a quantum gate can be considered as a matrix that works on the quantum state. For example, one Hadmard gate (H gate) can be represented by a unitary matrix $\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ which transforms the state $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}, |1\rangle$ to $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$. In general, one gate that works on n qubits can be represented by a $2^n \times 2^n$ matrix. Fig.1 shows an example of a 4-qubit variational quantum eigensolver (VQE) circuit. Each horizontal line describes the time evolution of the state of one qubit from left to right. Gates operating independently on different qubits can be executed simultaneously, for example, the H gates operating on q_0 and q_2. Gates on the same qubit have to respect the order of the gates. For example, the CNOT gate (denoted as $\oplus$) operating on q_0 and q_1 must wait until the H gate on q_0 and the CNOT gate on q_1 and q_2 are finished. As another important term, a *front layer* is defined by the set of all gates that have no unexecuted predecessors in the circuit. These gates can be executed instantly and concurrently from a software perspective. As shown in Fig. 1, the first three H gates on q_0, q_2, q_3 are the front layer.

Distributed quantum computing (DQC). DQC uses multiple quantum processing units (QPUs) to complete a large quantum computing job jointly via a quantum network [12]–[16]. When a gate is operated by two qubits on different QPUs, called a remote gate, DQC uses remote communication to establish an EPR pair between the QPUs.

III. Quantum Cloud Model

This section introduces the system model for the quantum cloud discussed in this work, an overview of our model is shown in Fig. 2.

QPU model. Each QPU is also equipped with a classical computer, which is used to manipulate and manage the quantum processor and transmit classical information such as measurement results to other QPUs and the controller. Each QPU includes two types of qubits: computing qubits to perform quantum gates and communication qubits that assist computing qubits for remote gates.

Controller. The main responsibility of the quantum cloud controller is to find the placement for each submitted circuit, and after that, it needs to decide resource allocation for

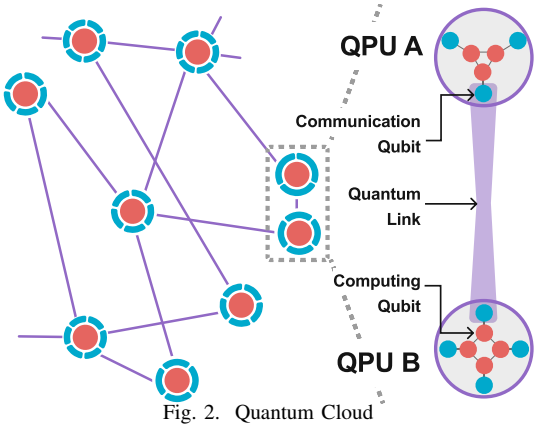

Fig. 2. Quantum Cloud

Operation	Latency
Single-qubit gates	$t_{1q} \sim 0.1\,\mathrm{CX}$
CX and CZ gates	$t_{2q} = 1\,\mathrm{CX}$
Measure	$t_{ms} \sim 5\,\mathrm{CX}$
EPR preparation	$t_{iep} \sim 10\,\mathrm{CX}$

TABLE I
SUMMARY OF OPERATIONS AND LATENCY

all currently placed circuits to complete their execution. It also monitors the status of each QPU, such as the available computing and communication qubits.

Quantum links and network topology. In our model, a quantum link is a channel for establishing entanglement between QPUs. These links carry flying photonic qubits, with devices such as beam splitters and photon detectors to realize the entanglement process. Each QPU is assumed to have a limited number of outgoing quantum links. Hence each QPU is connected to several (but not all) other QPUs by quantum links, forming a fixed topology of the QPU network.

Models for local gates and remote gates. In DQC, the latency of a remote gate includes the time for EPR preparation, local gate, and measurement. To model the latency of these operations, we use the measurement results of the latency of different quantum operations from IBM's quantum platform [19] and recent experiments [16]. As shown in Table I, We can see a remote gate consumes much longer time than a local gate. In Table I, we compare different operations to the execution time of one CX gate. Besides long execution time, another property of EPR pair generation is that its success is probabilistic. A failed EPR generation also consumes communication qubits.

IV. Problem Formulation

To execute multiple quantum circuits concurrently in a cloud setting, we need to address the following questions:

1) Which circuits should we place if the current resource cannot support all circuits? Given a quantum circuit, which QPUs should the quantum cloud select to execute the circuit?
2) After placing the circuits onto QPUs, how do we decide the strategy for communication resource allocation?

These two sub-problems are the main focus of this work. We define the first problem as ***multi-tenant circuit placement*** and the second problem as ***quantum network scheduling***.

A. Design Objectives

We outline some key considerations of achieving optimal placement of quantum circuits and making resource allocation decisions.

1) **Minimizing the network cost**: Remote gates that depend on EPR pairs are much more expensive and time-consuming than local gates. When partitioning a quantum circuit, we need to ensure the sum of the cuts is minimized so that the expensive remote communication will also be minimized.
2) **Dynamics in quantum cloud**: The resource availability in a quantum cloud changes dynamically with incoming jobs. Thus, we need to enforce the capability constraint. Moreover, we cannot focus solely on optimizing the allocation of each single circuit, which might lead to hard allocation for other circuits in the queue.
3) **Minimizing job completion time and maximizing quantum resource utilization**: Reducing job completion time is essential for improving user experience. This objective can also be reflected in improving resource utilization and avoid wasting qubits.

Our main design aims to achieve these goals. Specifically, during the multi-tenant circuit placement step, we strive to minimize the number of remote operations while maximizing quantum service utilization to avoid wasting qubits. In the quantum network scheduling step, we aim to further improve job completion time by effectively capturing the structure of the quantum circuits. With these objectives in mind, we present the formulation of our two problems.

B. The Multi-tenant Circuit Placement Problem

The model of the quantum cloud is depicted by $G = (V, E)$, where V represents the collection of QPUs and E denotes the set of connections between them, with an edge e existing between any two QPUs connected by a quantum link. Given a batch of quantum circuits $T_k = \{T_1, T_2, \ldots, T_N\}$, we use a binary variable x_k to denote whether the k-th circuit will be chosen to find placement in the current decision round. We use D_{ij}^k to denote the number of 2-qubit gates between q_i^k and q_j^k. We use C_{ij} to denote the communication cost of remote gate operations between QPU i and QPU j. C_{ij} can be defined in many ways since executing one remote gate may require multiple EPR pair generation attempts. It also depends on the distance between two QPUs since it may require entanglement swapping at intermediate nodes. For illustration, in our circuit placement step, we define C_{ij} as the length of the path between QPU i and QPU j. Define $\pi : [1, \ldots, N] \to [1, \ldots, |V|]$, where $\pi(q_i^k)$ indicates which QPU the i-th qubit of the k-th circuit will be mapped to. Also, $\mathrm{Rem}(V_i)$ is the remaining qubits on V_i. We formulate the multi-tenant circuit placement as follows:

$$\min \sum_{k=1}^{N} \sum_{i=1}^{n} \sum_{j=1}^{n} x_k D_{ij}^k C_{\pi(q_i^k)\pi(q_j^k)} \tag{1}$$

$$\min \sum_{i=1}^{n} \text{Rem}(V_i) \tag{2}$$

$$\text{s.t.} \sum_{i:\pi(i)=j} |Q_i| \leq \text{Capacity}(V_j), \quad \forall j \in \{1, 2, \ldots, n\} \tag{3}$$

$$x_k \in \{0, 1\}, \quad \forall k \tag{4}$$

$$\sum_{k=1}^{N} x_k \geq 1 \tag{5}$$

$$R(V_j) \leq \epsilon, \quad \forall j \in \{1, 2, \ldots, n\} \tag{6}$$

where

$$R(V) = \sum_{k=1}^{N} \sum_{i=1}^{n} \sum_{j=1}^{n} D_{ij}^k \cdot \delta(\pi(q_i^k), \pi(q_j^k), V_j), \quad \forall j \in \{1, 2, \ldots, n\} \tag{7}$$

and $\delta(\pi(q_i^k), \pi(q_j^k), V_j)$ is defined as:

$$\delta(\pi(q_i^k), \pi(q_j^k), V_j) = \begin{cases} 1 & \text{if } \pi(q_i^k) \neq \pi(q_j^k) \\ & \text{and } (\pi(q_i^k) = j \text{ or } \pi(q_j^k) = j) \\ 0 & \text{otherwise} \end{cases}$$

The function $\delta(\pi(q_i^k), \pi(q_j^k), V_j)$ is an indicator function that returns 1 if qubits q_i^k and q_j^k are mapped to different QPUs and one of them is mapped to V_j, and 0 otherwise. The term $R(V_j)$ represents the number of remote operations involving QPU V_j. It is calculated as the sum of all 2-qubit gates between qubits mapped to different QPUs, where one of the QPUs is V_j. Objective 1 minimizes the total communication cost for all circuits. Objective 2 minimizes the sum of unused computing qubits among all QPUs to maximize resource utilization. Inequation 3 ensures that the sum of used qubits on each QPU does not exceed its capacity. 4 ensures that x_k is a binary variable. Inequation 5 ensures at least one circuit must be selected which avoids the case that all $x_k = 0$. Inequation 6 enforces that the number of remote operations involving each QPU does not exceed a threshold ϵ.

Complexity analysis and NP-hardness. We first consider the simplest setting of our problem, where we only find the placement of a single circuit with only compatibility constraints. Given C_{ij} and D_{ij} are real values, the single circuit placement problem we consider here falls into the Quadratic Assignment Problem (QAP) [31], which is known to be the most difficult problem in NP-hard class [32]. The reduction can be constructed easily with the following: Given one instance of the QAP problem with facilities F and locations L, a weight function $w : F \times F \longrightarrow R$, and a distance function $d : L \times L \longrightarrow R$. The corresponding circuit placement problem can be constructed by: $Q = F$, $E = L$, $D_{ij} = W_{ij}$ and $d_{ij} = C_{ij}$. And by this construction, we can get an equivalent QAP problem.

To extend this problem to the multi-tenant setting, consider multiple instances of the QAP, one for each circuit. For each circuit k, we create qubits corresponding to the facilities in the QAP instance and treat each circuit as an independent QAP. We then define the communication costs C_{ij} to model interactions between qubits from different circuits. Then, we can formulate the multi-circuit problem as one Multi-objective Linear Programming problem. Due to the page limitation, we omit the details here. This reduction shows that the multi-tenant circuit placement problem is at least as hard as multiple instances of the QAP. It easily gives that the multi-tenant quantum circuit placement problem is also NP-hard.

C. Network Scheduling

Network scheduling example. Fig. 3 (a) shows all remote gates of a 13-qubit multiplier circuit, and (b) shows the corresponding DAG. To be noticed, we ignore single-qubit gates and only keep two-qubit inter-QPU remote gates, we With this example, we can see various contentions on communication qubits exist. For example, the first CX gate between q_0 and q_5 and the second CX gate between q_1 and q_6: These two gate spans the same set of QPUs (QPU A and QPU B) and will both rely on communication qubits on two QPUs. Also, the CX gate between q_6 and q_{12} and the CX gate between q_0 and q_7, both gates will rely on the communication qubits on QPU B. However, **communication qubits on each QPU is limited, and generating EPR pairs is a probabilistic operation. To perform a remote gate, we must ensure that at least one EPR pair between communication qubits on two QPU is ready. Thus, we need to determine how many pairs of communication qubits we need to prepare to perform EPR pairs generation for each remote gate on the remote DAG.**

1) Problem Formulation: The network scheduling problem in our quantum setting can be formulated as follows. Let M_i denote the number of communication qubits on QPU i. After the circuit placement step, each job will be mapped to a set of QPUs by a function: $f : J \rightarrow \mathcal{P}(Q)$ where $\mathcal{P}(Q)$ denotes the power set of QPUs. For each job J_i, we can get a DAG $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ where each node $u \in \mathcal{N}$ represents a remote operation between two machines Q_j, Q_k, where j and k depends on previous circuit placement step. Let x_u be the number of communication qubits allocated to remote operation u. Let c_u denote the makespan (total completion time) of node u. Then, the objective is to minimize the makespan of all remote operations while respecting the logical dependencies of DAG $\mathcal{G}$ and communication resources constraint on each QPU. The success probability of each remote operation u depends on the allocated resources x_u, denoted by $p_u(x_u)$.

$$\min \max_{u \in \mathcal{N}} c_u$$

$$\text{s.t.} \sum_{u \in \mathcal{N}_i} x_u \leq M_i \qquad \forall i \tag{8}$$

$$s_{u_j} \geq c_{u_i} \qquad \forall (u_i, u_j) \in \mathcal{A} \tag{9}$$

$$c_u = s_u + d_u(x_u) \qquad \forall u \in \mathcal{N} \tag{10}$$

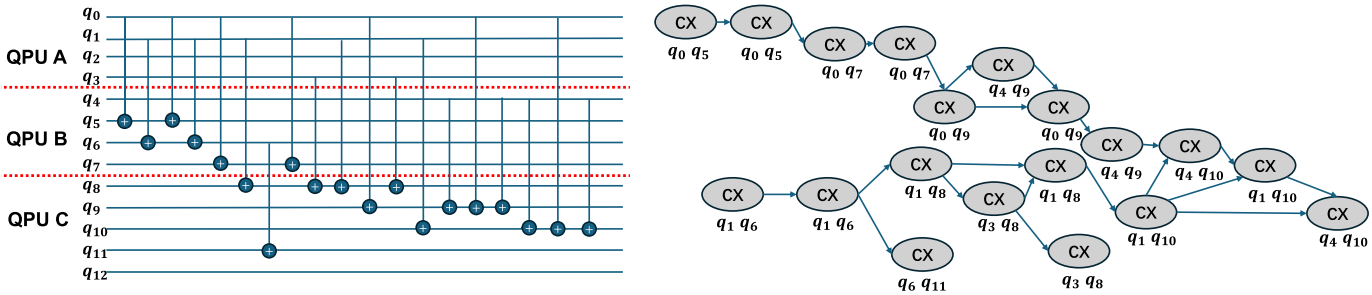

Fig. 3. (a) An example circuit spams three QPUs. (b)Corresponding remote DAG that only contains inter-QPU remote gates

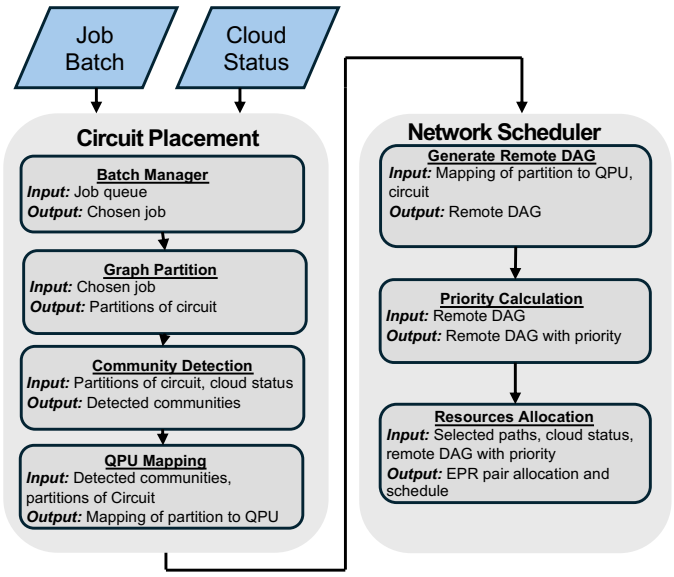

Fig. 4. Overview of our scheduler workflow

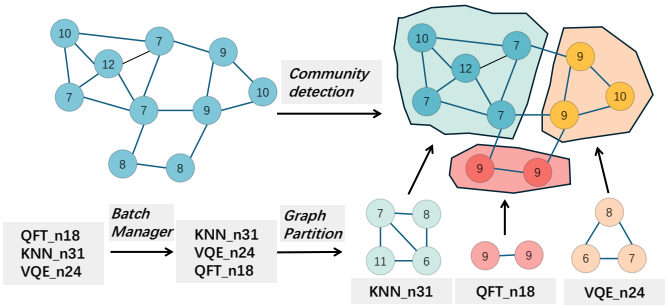

Fig. 5. Example of a quantum cloud with three DQC jobs.

Inequation 8 denotes that, for each QPU i, the total allocated resources at any time do not exceed the available communication qubits on each machine i. Here, $\mathcal{N}_i$ is the set of operations involving machine i. Inequation 9 ensures that an operation u_j can only start after its predecessor u_i has successfully completed, respecting the logical dependencies in the DAG. Eq. 10 denotes that the completion time of an operation u is the start time plus the duration, which depends on the allocated resources. It is important to note that when allocating communication qubits to each remote gate that spans Q_i and Q_j, the corresponding resources on both QPUs need to decrease by the allocated amount to reflect the consumption of communication resources on each involved QPU.

V. CloudQC's Design

A. Design Overview

We present the workflow of CloudQC in Fig. 4, which encompasses two main steps: circuit placement and network scheduling. CloudQC first determines the optimal placement for a batch of circuits, which involves circuit partitioning and assigning each partition to a suitable QPU to reduce remote communication costs. This starts with determining several partitioning strategies for a quantum circuit, followed by a feasibility assessment to select a compatible set of QPUs with the proposed community-detection method which will be introduced in detail later. The optimal placement is then selected through a scoring methodology. Upon finalizing the partitioning approach and corresponding QPUs, the network scheduler compiles the job and allocates the communication resource to each remote operation. Leveraging the optimal placement, CloudQC then calculates the network resource allocation for the jobs based on the priorities we defined. This includes determining the number of EPR pair generation attempts required for each remote gate.

B. Circuit Placement

Fig. 5 illustrates an example of the circuit placement algorithm. Three circuits are submitted to the quantum cloud. First, we determine the processing order of the jobs using the batch manager discussed below. Next, for each job, CloudQC determines how to partition the quantum circuit using a graph-partitioning algorithm. CloudQC then applies a modularity-based community detection algorithm [33] to identify a set of QPUs capable of running the job. We then map each partition to a QPU. For each circuit, we identify several possible placements. We use a scoring-and-filtering method to evaluate the quality of each placement and select the best one.

Batch manager. CloudQC works for two job processing modes: The batch mode and the incoming job mode. In the incoming job mode, jobs arrive one after another and CloudQC processes them in a first-in-first-out order. In the batch mode, multiple jobs arrive at the same time and CloudQC needs to decide the optimal order of executing the jobs. One naive approach will use a greedy approach: sorting the circuits based

Algorithm 1: Circuit Placement

Input: Job J, imbalance factor list $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$
Output: Best Placement for J

$InteractionGraph, DAG \leftarrow$ Preprocessing(J.circuit)
if *J.circuit.size* $< \min(\{QPU.available \mid QPU \in Cloud\})$ **then**
  Allocate J to the QPU with sufficient available resources
else
  Initialize an empty list for potential placements: $placement\_list \leftarrow \emptyset$
  foreach α_i *in* α **do**
    for i *in range(*$\lceil \frac{J.circuit.size}{QPU.num} \rceil$*, number of QPUs in Cloud)* **do**
      $res \leftarrow$ graph_partition(α_i, i)
      $placement \leftarrow$ Find_placement(res)
      $time \leftarrow$ estimate_time(DAG, res)
      $communication\_cost \leftarrow$ calculate_cost($res, InteractionGraph$)
      $placement$.score $\leftarrow$ score($time, communication\_cost$)
      Add $placement$ to $placement\_list$
    end
  end
  $best\_placement \leftarrow$ Find_highest_score($placement\_list$)
end
return $best\_placement$

on a metric and allocating as many circuits as possible. However, such a method may result in partitioning one circuit into too many pieces which will result in massive communication costs and thus will eventually result in longer JCT.

We determine the order of the circuits by this metric:

$$I_i = \lambda_1 \cdot \frac{\#CNOTs}{n_i} + \lambda_2 \cdot \ n_i + \lambda_3 \cdot d_i \tag{11}$$

where n_i denotes the number of qubits in i_th job in the circuit, and d_i denotes the circuit depth of i_th job. Our metric is based on the following considerations: 1) $\frac{\#CNOTs}{n_i}$ denotes how 'dense' one circuit is in terms of 2-qubit interactions. One circuit with more 2-qubit interaction will likely have more remote gates when distributed to multiple QPUs. Consequently, circuits with a higher density of 2-qubit gates are more likely to suffer from increased latency and reduced fidelity due to the overhead of managing these remote interactions. 2) n_i represents the number of qubits in the job, which directly correlates to the job's resource requirements. 3) d_i denotes the depth of the circuit, which reflects the execution time of the circuit. Deeper circuits will take longer to execute, and thus their impact on overall throughput and job completion time is significant. By incorporating these factors, the metric I_i aims to balance the trade-offs between communication costs, resource utilization, and execution time. The weights λ_1, λ_2, and λ_3 allow for tuning the importance of each factor based on specific priorities and goals.

Circuit placement summary. After determining the job we need to execute in the batch, we need to find the placement of each job. The objective of job placement is to find the best placement that distributes a quantum circuit to several QPUs of the quantum cloud. Our method is summarized in Algorithm. 1. Our method works in a filtering-and-scoring fashion: For each circuit to be executed, we first find several methods to partition the circuit by tuning the imbalance factor of the graph partition algorithm, detailed later. Then, for each part, we first use the community detection algorithm to find a set of QPUs and apply a simple heuristic to map the partition to select QPUs. Then, for each placement we find, we evaluate it by the following scoring function: $S = \alpha \times \frac{1}{T} + \beta \times \frac{1}{C}$, where T is the estimated running time of the quantum circuit, C is the communication cost, and with this function, we consider both the performance and execution time of the circuit.

Preprocessing. CloudQC first generates a Directed Acyclic Graph (DAG) representing the logical dependency of the gates of each quantum circuit. Each single-qubit gate has to wait until its parent gate is finished, and similarly, a remote $\mathrm{CNOT}\,(q_i, q_j)$ gate can be executed only when all previous gates on q_i and q_j have finished. The interaction graph is a weighted graph where the vertices are qubits of the circuit and the edge denotes the interaction of two qubits, the weight describes how many 2-qubit gates two qubits have. As shown in Fig. 3, (a) shows an example 13-qubit adder circuit, and (b) shows the corresponding DAG.

Partitioning quantum circuit. When partitioning quantum circuits, there are the following considerations: 1) Minimize communication cost. A remote gate depends on the remote EPR pair and is much more expensive and time-consuming than a local 2-qubit gate. Thus, we want to minimize remote communication. 2) Reduce execution time. The execution time determines the job completion time.

We first apply a graph-partition algorithm [34] to the interaction graph based on these considerations. The algorithm aims to divide a graph into smaller, interconnected subgraphs, with the goal of minimizing the number of edges across different subgraphs while maintaining a certain load-balancing level. This is because extremely uneven partitions may not fit the resource availability on QPUs. Thus, in our implementation, we also tune the imbalance factor, which defines how imbalanced the result of graph partitioning can be.

Finding feasible QPU sets. For each quantum circuit partition result, CloudQC needs to find a set of QPUs with available resources to accommodate such partition. This is a challenging task because the choice of QPUs affects not only the performance of the current circuit but also future circuits. We need to consider the following factors in addition to resource avaiability: 1) Communication cost. When mapping the partition to QPUs, we need to ensure that the parts with large inter-communication costs should be placed onto QPUs with short network distances, because multi-hop communication will further increase the network cost and job

failure rate. 2) Future resource availability. When choosing a set of QPUs, we hope such placement can provide resource availability for future jobs. One good example is that after the current placement, the remaining QPU resources are still within short network distances. To achieve these goals, we use a modularity-based community detection algorithm [33] to find feasible QPU sets. A community detection algorithm identifies groups of nodes in a network that are more densely connected internally than with the rest of the network. A modularity-based community detection algorithm optimizes the modularity metric, which quantifies the quality of the division by comparing the density of links within communities to the density of links between communities.

Additionally, we can embed the number of computing qubits into the edge weight. This ensures that the selected QPUs have both strong connectivity and abundant computing qubits, capturing the dynamics of the quantum cloud and reflecting the capability of the selected QPUs. The community detection method provides a powerful framework for profiling each QPU. This profiling can help measure the performance of QPUs in various aspects. For example, in future modular quantum computer designs, we might consider the reliability of quantum links between QPUs and the reliability of each QPU. This information can be easily encoded into the edge weights.

Mapping partitioned circuits to selected QPUs. We use a heuristic algorithm to map the partitioned circuits to QPUs. We calculate the graph center of the found community and the interaction graphs, which minimizes the longest topological distance to all other nodes. Then, we map the center of the remote interaction graph to the center of the detected community graph. The rest of the mapping will be expanded from the node with the highest weight edge. We perform a breadth-first search around the first logical center to place each qubit of the circuit to an available QPU with the least distance to the center. The details of the heuristic are shown in the Algorithm. 2. **This simple heuristic ensures that two partitions with a high communication cost will be mapped to two close QPUs in the cloud.**

Algorithm 2: Find Placement

Input: Partition $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$, Quantum Cloud QPU Topology Graph G_c, Remote Partition Interaction Graph G_p
Output: Partition to QPU Mapping:mapping[]

Initialize mapping[] = -1
C ← CommunityDetection(G_c)
C_c ← GraphCenter(C)
C_p ← GraphCenter(G_p)
$mapping[C_p] \leftarrow C_c$
q ← BfsQueueGen(G_c)
while *q is not empty* **do**
  $q_i \leftarrow q.front()$
  C_c ← GraphCenter(C)
  if *mapping[q_i] = -1* **then**
    $mapping[q_i] \leftarrow C_c$
  end
  $N(q_i) \leftarrow GetNeighbors(q_i)$
  $N(C_c) \leftarrow GetKClosestNode(C_c)$
  $mapping[N(q_i)] \leftarrow N(C_c)$
  $q.pop()$
end

Algorithm 3: Network Scheduler

Input: Remote DAG

Compute priorities for all nodes in the DAG
Initialize front layer based on in-degree and out-degree
while *Remote DAG is not empty* **do**
  Define resource availability for each partition
  Identify competing sets for resource allocation
  Allocate resources between competing sets
  foreach *node in front layers* **do**
    Determine resources allocation based on the priority of node
    if *node is successfully executed* **then**
      Update status table
      Prepare for node deletion and successor addition
    end
  end
  Update front layer and DAG based on node execution
end

C. Network Scheduling

We would like to achieve the following desirable properties of network scheduling:

- Effectiveness: More 'important' sub-jobs in the remote DAG should receive more communication resources to prevent subsequent tasks from being backlogged. When multiple sub-jobs can be executed concurrently, more important jobs should receive more resources.
- Starvation freedom: When multiple jobs compete for communication resources on the same QPU, no job should be starved forever.

We use the same example in n Fig. 3 to show our observations: q_0 and q_5 and the second CX gate between q_1 and q_6: These two gates span the same set of QPUs and can be processed in parallel. However, these two gates have different 'importance' on the DAG: the first gate is on the critical path, and there are also more gates on its corresponding paths meaning that the failure of this gate may cause more gates to be backlogged. Thus, we should assign redundant resources to the first gate to increase failure tolerance. Another example is the seventh gate between q_6 and $q_1 1$, which crosses QPUs B and C, and the eighth gate between q_1 and q_8, which crosses QPUs A and B. Both gates will use communication qubits on QPU B, but we can see the gate between q_1 and q_8 is far less important than the first gate. Thus, CloudDC will allocate more communication resources to the ninth gate than the eighth gate.

With these observations, CloudDC first quantifies the impor-

tance of each node on remote DAG with the priority we define. We denote the priority of node n_i as p_i and $\mathcal{P}(n_i)$ as the set of paths connecting node n_i with any remote DAG leaves. The priority p_i can now be computed by: $p_i = \max_{P \in \mathcal{P}(n_i)} |P|$, where P represents any path from n_i to a leaf node, and $|P|$ is the length of path P in terms of the number of edges. This priority calculation is based on the depth of the longest path from node n_i to any leaf node in the DAG. With this priority value, we can determine whether the node is on the critical path of remote DAG and how many nodes will be blocked if the execution of the node is unsuccessful.

With these observations and our definition on priority, we summarize our method for network scheduling in Algorithm. 3. The network scheduling algorithm first calculates the priority of each node. Then, it initializes the front layer, which is defined by the remote operations that can be processed in parallel. It then identifies contentions between different QPU partitions and then allocates resources to them. The network scheduler repeatedly checks the current front layer on whether each gate is executed successfully and updates the front layer in the graph if some gates are completed. The above process repeats until the remote DAG is empty.

VI. Evaluation

In this section, we use experimental evaluations to answer the following questions:

- How does the proposed job placement method compare to existing algorithms (Sec. VI. B)?
- How much can network scheduling improve compared to other resource allocation strategies in terms of job completion time (VI. C)?
- How is CloudQC's general performance in a multi-tenant setting (VI. D)?

A. Evaluation Setting

Workloads. We use the quantum circuits from an existing benchmark [35], whose characteristics are shown in Table II.

Implementation. Since there is no publicly available quantum cloud simulator, we developed a customized discrete-event simulator in Python. For quantum circuit analyzing, we use PytKet [36]. We use PyMetis, a Python version package for Metis [34] for graph partition. The simulation code is available to the public [29].

Topology setting. We set the default number of QPUs in a quantum cloud to be 20; each QPU is equipped with 20 computing qubits and 5 communication qubits. We use a random topology, and we set the probability of generating an edge to be 0.3. We set the success probability of generating an EPR pair to be 0.3, consistent to existing work and the experiments [16], [37].

B. Evaluation on Circuit Placement

We first evaluate how CloudQC performs on placing single circuits. The metric is the communication cost $\sum_{i=1}^{n}\sum_{j=1}^{n} D_{ij}^{k} C_{\pi(q_i^k)\pi(q_j^k)}$(In single circuits case we omit x_k term) defined in Section III. We use the following baselines in comparison for circuit placement.

TABLE II
Quantum circuit characteristics

Name	# of Qubits	# of 2-Qubit Gates	Circuit Depth
ghz_n127	127	126	128
bv_n70	70	36	40
bv_n140	140	72	76
ising_n34	34	66	16
ising_n66	34	130	16
ising_n98	98	194	16
cat_n65	65	64	66
cat_n130	130	129	131
swap_test_n115	115	456	60
knn_n67	67	264	36
knn_n129	129	512	67
qugan_n71	71	418	72
qugan_n111	111	658	112
cc_n64	64	64	195
adder_n64	64	455	78
adder_n118	118	845	132
multiplier_n45	45	2574	462
multiplier_n75	75	7350	1300
qft_n63	63	9828	494
qft_n160	160	25440	1270
qv_n100	100	15000	701

TABLE III
Number of Remote Operations of single-circuit placement

Circuit	SA	Random	GA	CdQC-BFS	CdQC
ghz_n127	145	161	90	10	8
bv_n70	41	38	17	26	18
bv_n140	96	98	54	101	53
ising_n34	38	36	6	2	2
ising_n66	100	110	36	6	8
ising_n98	214	250	96	10	10
cat_n65	52	44	20	5	3
cat_n130	153	145	92	10	8
swap_test_n115	398	472	294	352	192
knn_n67	158	230	106	168	100
knn_n129	528	720	374	376	220
qugan_n71	334	482	278	180	144
qugan_n111	838	1080	718	404	248
cc_n64	45	44	44	46	44
adder_n64	269	450	142	33	33
adder_n118	748	1225	613	60	37
multiplier_n45	596	1452	493	611	462
multiplier_n75	2100	6809	2255	1993	1766
qft_n63	2504	3202	2368	3012	2358
qft_n160	12326	15514	14246	14814	11132

- Random Placement: It starts with a random node and does a random search to select a set of QPUs that meet computing constraints.
- Simulated Annealing (SA): SA is a meta-heuristic that is widely used in optimization problems and placement in the cloud. Here, we use the strategy of a recent work [11], which uses SA for qubit allocation for a single DQC job.
- Genetic Algorithm (GA) [38]: GA is also a meta-heuristic that is widely used in optimization problems.
- CloudQC-BFS: Also a method proposed by us. It differs from CloudQC in using a BFS search to find feasible QPU for each partition instead of community detection.

1) Circuit Placement for Single Circuit with Default Setting: We show the results on circuit placement in Table III. We can see that: 1) CloudQC (and CloudQC-BFS) significantly outperforms other baselines for most circuits. For

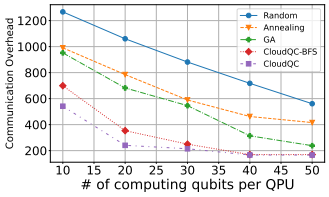

Fig. 6. Overhead vs # of computing qubits: qugan_n111

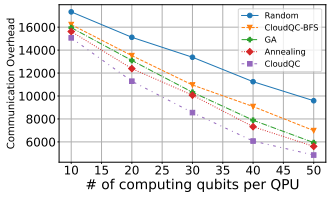

Fig. 7. Overhead vs # of computing qubits: qft_n160

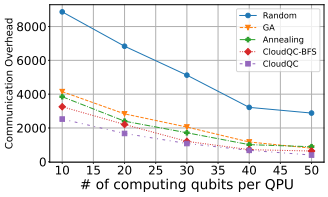

Fig. 8. Overhead vs # of computing qubits: multiplier_n75

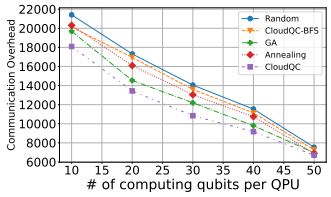

Fig. 9. Overhead vs # of computing qubits: QV_n100

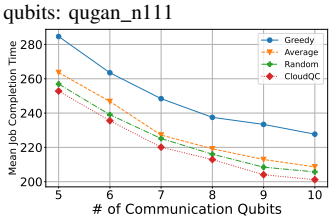

Fig. 10. Job Completion Time vs # of communication qubits: qugan_n111

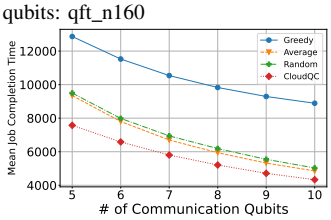

Fig. 11. Job Completion Time vs # of communication qubits: qft_n160

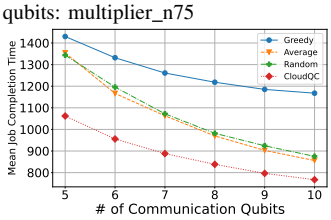

Fig. 12. Job Completion Time vs # of communication qubits: multiplier_n75

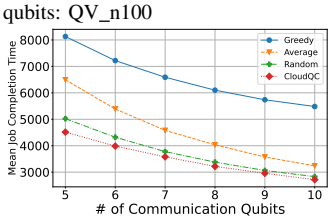

Fig. 13. Job Completion Time vs # of communication qubits: QV_n100

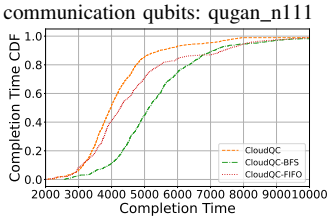

Fig. 14. Job Completion CDF Time with Mixed Workloads

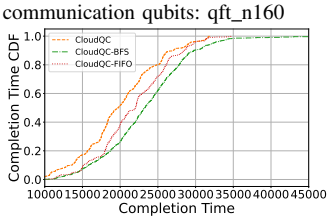

Fig. 15. Job Completion Time CDF with QFT Workloads

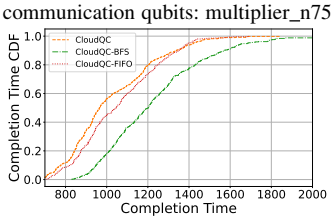

Fig. 16. Job Completion Time CDF with Qugan Workloads

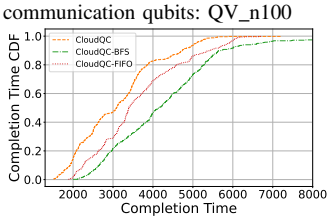

Fig. 17. Job Completion Time CDF with Arithmetic Workloads

cc_n64, bv_n70, knn_n96, cat_n65, cat_n130, ising_n34 and ghz_n127, CloudQC provides similar results to CloudQC-BFS. The reason is that these circuits have fewer 2-qubit gates, and graph partition contributes most to the final result. However in larger circuits with more complicated inter-QPU interactions, such as qft_n63, qft_n160, multiplier_n75, and swap_test_n115, CloudQC performs significantly better than CloudQC-BFS, which shows the effectiveness of the proposed mapping method and community detection. 3) SA works better than random placement but much worse than CloudQC. The reason is that the performance of SA highly depends on the initial placement. We also observed a long running time (>1 hour) of SA and GA while CloudQC finishes within 2 minutes for most circuits.

2) Effect of Number of Computing Qubits: We vary the number of computing qubits on each QPU by 20, 30, 40, and 50. Due to the page limit, we select the following representative quantum circuits as benchmarks: *qv_n100*, *multiplier_n75*, *qft_n160*, and *qugan_n111*. We show the results in Figs. 6 to 9. We can see CloudQC performs the best for these circuits, and CloudQC-BFS also performs better than other baselines. Although it also tends to select close connected QPUs it may fail to capture the topology information of the cloud. SA and GA perform worse than our methods in that it fails to capture the communication information between different qubits in the circuit. GA performs better than SA, but we also observe a much longer running time.

C. Evaluation on Network Scheduling

We compare the following methods to evaluate the flowing scheduling in CloudQC.

- Greedy: It always allocates the maximum resources to the remote operation with the highest priority.
- Average: It distributes communication resources evenly among all remote operations.
- Random: Each remote operation has an equal probability of receiving communication resources.

1) Network Scheduling with Default Setting: Figure 22 illustrates the network scheduling results with default settings. CloudQC significantly achieves the least job completion time for most quantum circuits especially with more complex and random structures such as Quantum Fourier Transform (QFT), Multiplier, and Quantum Volume (QV) circuits. This is due to their ability to utilize the DAG topology. For circuits without complex DAG topology, such as BV, Qugan, KNN, and Swap test, CloudQC performs similarly to others. Greedy has the worst job completion time.

2) Effect of Number of Communication Qubits: We vary the number of communication qubits from 5 to 10 and show the corresponding results in Figs. 10 to 13. We use qugan_n111, qft_n63, multiplier_n75 and QV_n100 as representative benchmarks. From all results, we can see the number of communication qubits has a large impact on the completion time. Similar to previous observations for circuits with a more complicated structure, CloudQC achieves a significantly shorter completion time compared to other methods.

3) Effect of EPR success Probability: We vary the EPR success probability from 0.1 to 0.5, and the corresponding results are shown from Figs.18 to 21. We observe that increasing the EPR probability decreases job completion time. Across all scenarios, CloudQC consistently achieves shorter job completion time except for one data point (probability 0.1 for qugan_n111). With these results, we can see that improving the success rate of EPR pairs will be a crucial step in the future DQC hardware.

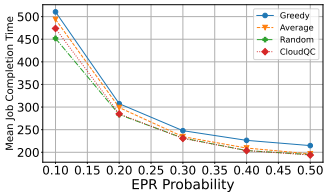

Fig. 18. Job Completion Time vs EPR probability: qugan_n111

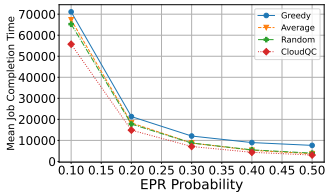

Fig. 19. Job Completion Time vs EPR probability: qft_n160

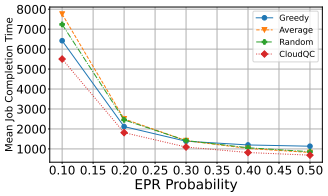

Fig. 20. Job Completion Time vs EPR probability:multiplier_n75

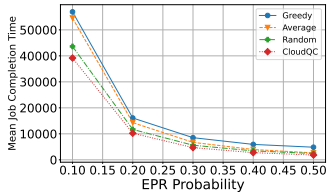

Fig. 21. Job Completion Time vs EPR probability: QV_n100

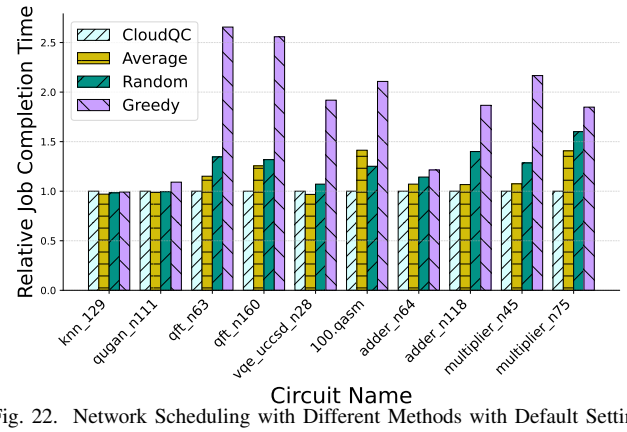

Fig. 22. Network Scheduling with Different Methods with Default Setting

D. Evaluation on Multi-Tennant Settings

1) Evaluation Setting: For multi-tenant experiments, we use the following workloads of different types of quantum circuits with different numbers of qubits:

- Mixed workloads: It contains different types of quantum circuits with different numbers of qubits: knn_n129, qugan_n111, qugan_n71, qft_n63, multiplier_n45, multiplier_n75.
- QFT workloads: It contains QFT circuits with different numbers of qubits: qft_n29, qft_n63, qft_n100.
- Qugan workloads: It contains Qugan circuits with different numbers of qubits:qugan_n39, qugan_n71, qugan_n111.
- Arithmetic workloads: It contains different types of quantum arithmetic circuits with different numbers of qubits: adder_n64, adder_n118, multiplier_n45, multiplier_n75

We compare CloudQC with the following methods:

- CloudQC-BFS: It uses the same batch manager as CloudQC but uses the BFS method in each circuit placement.
- CloudQC-FIFO: It uses the same placement and network scheduling methods as CloudQC but a First-In-First-Out processing order for batched jobs.

For each workload, we generate 50 batches, each of which includes 20 circuits selected randomly from the workload. Each batch is run 20 times with different network topologies.

We show the results from Figs. 14 to 17 as the CDF of the job completion time. We find that CloudQC performs better than other methods, followed by CloudQC-FIFO. In mixed workloads, CloudQC finishes around 88% in around 5000 units of time, and CloudQC-FIFO only finishes around 70% at the same time. In circuits with complicated patterns and longer depths, we can see CloudQC still performs significantly better than CloudQC-BFS and CloudQC-FIFO. However for circuits with shorter depths(Qugan), the differences are small. CloudQC-BFS does not perform well in multi-tenant scenarios although in placing single circuits, it achieves the smaller cost. Hence we find that CloudQC is the best choice for both single-circuit and multi-circuit placement and network scheduling.

VII. Related Work

Distributed Quantum Computing. Distributed quantum computing has been studied in recent years. Some of the works focus on qubit allocation and compilers [7], [11], [39]–[41]. Mao *et al.* [11] proposed to use a hybrid simulated annealing algorithm to determine the qubits allocation in distributed quantum computing. Andres-Martinez and Heunen [40] used partitioning hypergraph to minimize communication costs. Baker *et al.* [42] used remote SWAP gates to replace all remote CX gates in distributed quantum programs and obtain higher throughput. Autucomm [7] identified burst communication patterns in DQC and determined the best choice for using the cat-entangler method or teleportation method. These works mainly focus on optimizing communication of a single circuit and do not take the probabilistic nature of quantum communication into consideration.

Quantum Multi-programming and Quantum Cloud.The idea of quantum multi-programming was proposed in [22]; it considers fairness when allocating multiple quantum programs on one single QPU. [23] use similar ideas and evaluate their methods on various benchmarks on real devices. Liu *et al.* [43] proposed a quantum data center architecture composed of quantum random access memory and quantum networks and visioned three quantum applications. Ravi *et al.* [44] also surveyed various applications and resource utilization problems in a cloud environment for quantum computing.

VIII. Conclusion

This work presents a network-aware framework for DQC in a multi-tenant quantum cloud. The framework, called CloudQC, consists of two critical components: circuit placement and network scheduling. CloudQC is the first work to consider the placement and network scheduling for multiple concurrent DQC circuits. In addition, it incorporates the probabilistic nature of quantum networks to allow redundant network resources for important quantum gates in each circuit avoiding backlogs of later circuits and reducing the job completion time. The simulation results show that CloudQC significantly reduces the job completion time and cost for both single-circuit and multiple-circuit placement and scheduling. We believe our work is a valuable step in quantum

computing and networks as it envisions a practical quantum cloud infrastructure that will surely emerge in the near future.

ACKNOWLEDGMENT

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

REFERENCES

[1] 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.

[2] R. P. Feynman *et al.*, "Simulating physics with computers," *Int. j. Theor. phys*, vol. 21, no. 6/7, 2018.

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

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

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

[6] 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.

[7] 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.

[8] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L.-M. Duan, and J. Kim, "Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects," *Physical Review A*, vol. 89, no. 2, p. 022317, 2014.

[9] A. Wu, Y. Ding, and A. Li, "Qucomm: Optimizing collective communication for distributed quantum computing," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 479–493.

[10] D. Cuomo, M. Caleffi, K. Krsulich, F. Tramonto, G. Agliardi, E. Prati, and A. S. Cacciapuoti, "Optimized Compiler for Distributed Quantum Computing," *ACM Transactions on Quantum Computing*, vol. 4, no. 2, pp. 1–29, Jun. 2023.

[11] Y. Mao, Y. Liu, and Y. Yang, "Qubit allocation for distributed quantum computing," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023, pp. 1–10.

[12] S. Hermans, M. Pompili, H. Beukers, S. Baier, J. Borregaard, and R. Hanson, "Qubit teleportation between non-neighbouring nodes in a quantum network," *Nature*, vol. 605, no. 7911, pp. 663–668, 2022.

[13] J. Niu, L. Zhang, Y. Liu, J. Qiu, W. Huang, J. Huang, H. Jia, J. Liu, Z. Tao, W. Wei *et al.*, "Low-loss interconnects for modular superconducting quantum processors," *Nature Electronics*, vol. 6, no. 3, pp. 235–241, 2023.

[14] P. Magnard, S. Storz, P. Kurpiers, J. Schär, F. Marxer, J. Lütolf, T. Walter, J.-C. Besse, M. Gabureac, K. Reuer *et al.*, "Microwave quantum link between superconducting circuits housed in spatially separated cryogenic systems," *Physical Review Letters*, vol. 125, no. 26, p. 260502, 2020.

[15] Y. Li and J. Thompson, "High-rate and high-fidelity modular interconnects between neutral atom quantum processors," *arXiv preprint arXiv:2401.04075*, 2024.

[16] M. Pompili, S. L. Hermans, S. Baier, H. K. Beukers, P. C. Humphreys, R. N. Schouten, R. F. Vermeulen, M. J. Tiggelman, L. dos Santos Martins, B. Dirkse *et al.*, "Realization of a multinode quantum network of remote solid-state qubits," *Science*, vol. 372, no. 6539, pp. 259–264, 2021.

[17] 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 IEEE International Symposium on Workload Characterization (IISWC)*, 2021.

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

[19] "Utility-scale quantum computing on ibm cloud," 2024. [Online]. Available: https://ibm.com/blog/announcement/new-ibm-quantum-systems-on-the-ibm-cloud/

[20] "Azure quantum cloud service: Get innovative quantum hardware, software, and solutions in a single cloud service." 2024. [Online]. Available: https://azure.microsoft.com/en-us/products/quantum

[21] "Nvidia launches cloud quantum-computer simulation microservices." 2024. [Online]. Available: https://nvidianews.nvidia.com/news/nvidia-launches-cloud-quantum-computer-simulation-microservices

[22] P. Das, S. S. Tannu, P. J. Nair, and M. Qureshi, "A case for multi-programming quantum computers," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 291–303.

[23] S. Niu and A. Todri-Sanial, "Enabling multi-programming mechanism for quantum computing in the nisq era," *Quantum*, vol. 7, p. 925, 2023.

[24] P. Andres-Martinez, T. Forrer, D. Mills, J.-Y. Wu, L. Henaut, K. Yamamoto, M. Murao, and R. Duncan, "Distributing circuits over heterogeneous, modular quantum computing network architectures," Jul. 2023, arXiv:2305.14148 [quant-ph].

[25] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proceedings of IEEE INFOCOM*, 2010.

[26] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proceedings of ACM SIGCOMM*, 2011.

[27] M. Masdari, S. S. Nabavi, and V. Ahmadi, "An overview of virtual machine placement schemes in cloud computing," *Journal of Network and Computer Applications*, 2016.

[28] R. Van Meter, *Quantum networking*. John Wiley & Sons, 2014.

[29] https://github.com/embersax/CloudQC.

[30] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.

[31] E. M. Loiola, N. M. M. De Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido, "A survey for the quadratic assignment problem," *European journal of operational research*, vol. 176, no. 2, pp. 657–690, 2007.

[32] S. Sahni and T. Gonzalez, "P-complete approximation problems," *Journal of the ACM (JACM)*, vol. 23, no. 3, pp. 555–565, 1976.

[33] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[34] G. Karypis and V. Kumar, "Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," 1997.

[35] 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.

[36] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, "t— ket¿: a retargetable compiler for nisq devices," *Quantum Science and Technology*, vol. 6, no. 1, p. 014003, 2020.

[37] S. Shi and C. Qian, "Concurrent entanglement routing for quantum networks: Model and designs," in *SIGCOMM*, 2020, pp. 62–75.

[38] J. H. Holland, "Genetic algorithms," *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[39] J. M. Baker, C. Duckering, A. Hoover, and F. T. Chong, "Time-sliced quantum circuit partitioning for modular architectures," in *Proceedings of the 17th ACM International Conference on Computing Frontiers*, 2020, pp. 98–107.

[40] P. Andres-Martinez and C. Heunen, "Automated distribution of quantum circuits via hypergraph partitioning," *Physical Review A*, vol. 100, no. 3, p. 032308, 2019.

[41] S. DiAdamo, M. Ghibaudi, and J. Cruise, "Distributed quantum computing and network control for acc elerated vqe," *arXiv preprint arXiv:2101.02504*, 2021.

[42] J. M. Baker, C. Duckering, A. Hoover, and F. T. Chong, "Time-sliced quantum circuit partitioning for modular architectures." Catania Sicily Italy: ACM, May 2020, pp. 98–107.

[43] J. Liu, C. T. Hann, and L. Jiang, "Data centers with quantum random access memory and quantum networks," *Physical Review A*, vol. 108, no. 3, p. 032610, 2023.

[44] G. S. Ravi, K. N. Smith, P. Gokhale, and F. T. Chong, "Quantum computing in the cloud: Analyzing job and machine characteristics," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2021, pp. 39–50.