

UNIQ: Communication-Efficient Distributed Quantum Computing via Unified Nonlinear Integer Programming

Hui Zhong, Jiachen Shen, Lei Fan, Xinyue Zhang, Hao Wang, Miao Pan and Zhu Han

Abstract—Distributed quantum computing (DQC) is widely regarded as a promising approach to overcome quantum hardware limitations. A major challenge in DQC lies in reducing the communication cost introduced by remote CNOT gates, which are significantly slower and more resource-consuming than local operations. Existing DQC approaches treat the three essential components—qubit allocation, entanglement management, and network scheduling—as independent stages, optimizing each in isolation. However, we observe that these components are inherently interdependent, and therefore adopting a unified optimization strategy can be more efficient to achieve the global optimal solutions. Consequently, we propose UNIQ, a novel DQC optimization framework that integrates all three components into a non-linear integer programming (NIP) model. UNIQ aims to reduce the circuit runtime by maximizing parallel Einstein–Podolsky–Rosen (EPR) pair generation through the use of idle communication qubits, while simultaneously minimizing the communication cost of remote gates. To solve this NP-hard formulated problem, we adopt two key strategies: a greedy algorithm for efficiently mapping logical qubits to different QPUs, and a JIT (Just-In-Time) approach that builds EPR pairs in parallel within each time slot. Extensive simulation results demonstrate that our approach is widely applicable to diverse quantum circuits and QPU topologies, while substantially reducing communication cost and runtime over existing methods.

Index Terms—Distributed quantum computing, qubit allocation, network scheduling, EPR pair generation.

I. INTRODUCTION

Due to its unique quantum properties, quantum computing offers the potential to solve problems at an exponential speed compared to classical computing [1]–[3]. As a result, quantum computing is well-suited for solving complex problems [4]–[6], especially in domains such as molecular simulation [7], drug discovery [8] and risk analysis [9]. To fully realize the advantages of quantum computing, practical quantum algorithms often require millions of qubits. However, current quantum hardware is still in the Noisy Intermediate-Scale Quantum (NISQ) era, where only a few hundred qubits are available [10]–[14]. Moreover, due to fabrication challenges [15], crosstalk errors [16] and quantum decoherence [17], scaling up the number of reliable qubits remains difficult in the short term, which restricts the advancement of quantum computing. To address this limitation, a widely accepted approach in both academia and industry is distributed quantum computing (DQC) [18]–[21]. Similar to classical distributed computing, DQC enhances the computational power of quantum systems by interconnecting multiple small-scale quantum chips (QPUs)

via a quantum network, thereby effectively increasing the total number of available qubits. For example, connecting two 128-qubit QPUs in a DQC system can functionally emulate a 256-qubit quantum computer.

The general workflow of DQC is typically divided into three main components: *qubit allocation*, *entanglement management*, and *network scheduling* [22]–[25]. Qubit allocation divides large-scale quantum algorithms into multiple subcircuits and then assigns them to different QPUs for collaborative execution of the overall task. Gates within the same QPU are referred to as local gates, while those spanning across QPUs are referred to as remote gates. Since remote gates are much slower and more resource-consuming than local gates [26], a key optimization objective is to minimize the number of remote gates. Entanglement management is responsible for establishing EPR pairs and communication channels between QPUs to facilitate the execution of remote gates. A commonly adopted approach is the Cat-Comm protocol, which relies on cat entanglement and disentanglement procedures [27]. This protocol uses specially designed circuits to entangle qubits across QPUs, while preserving the quantum information of the involved qubits. Network scheduling determines the execution order of remote gates by analyzing their dependency relationships. For example, if two remote CNOT gates share a common qubit, they must be executed in sequence. This is typically modeled as a directed acyclic graph (DAG) [28], where each node corresponds to a remote gate, and edges denote execution dependencies.

Several studies have explored the optimization of the DQC framework. Mao et al. (INFOCOM 2023) [29] focus on the qubit allocation problem for DQC and prove its NP-hardness. They propose an MHSa-based approach that combines local search and simulated annealing techniques, achieving better performance compared to existing methods. Similarly, Kan et al. (QCE 2024) [30] also concentrate on qubit allocation by merging sub-circuits. They dynamically adjust the partitioning strategy based on the resource constraints of qubits, thereby reducing the number of circuit cuts. In contrast, Chandra et al. (TPS-ISA 2024) [28] specialize in network scheduling for DQC, comparing a Resource-Constrained Project Scheduling Problem (RCPSP) framework with a greedy heuristic. Their results demonstrate that the two methods are suitable for different application scenarios, depending on circuit complexity. Further extending the scope, Zhou et al. [31] shift the focus from a single-tenant to a multi-tenant DQC setting.

They propose the CloudQC framework, which progressively optimizes qubit allocation and network scheduling to minimize the circuit runtime.

However, we found that most existing methods suffer from the following *limitations*. First, prior work typically divides DQC into the three components mentioned above and optimizes each of them separately. However, these components are inherently interdependent. Optimizing a single stage in isolation lacks a global view and often fails to achieve minimal circuit runtime. Second, since EPR generation time far exceeds gate execution time, the serial approach of establishing EPRs one by one before each remote gate leads to excessive latency. So, we need to find more efficient strategies to reduce the total EPR generation time, such as parallel EPR establishment or EPR reuse. Third, existing studies lack a comprehensive evaluation methodology. Some works compare their overall DQC frameworks with prior systems [22], [30], while others only evaluate specific algorithms against known baselines (e.g., simulated annealing) [29], [31]. So the overall advantages of their DQC frameworks cannot be fully validated.

Therefore, this paper novelly proposes UNIQ, a unified optimization framework for DQC which further reduces the communication cost of remote gates and minimizes the circuit runtime. Specifically, we first integrate qubit allocation, entanglement management, and network scheduling into a unified Nonlinear Integer Programming (NIP) model, thereby obtaining a more efficient feasible solution. Second, we cut the runtime of the entire circuit into uniform time slots. Within each slot, we minimize the EPR generation time by utilizing idle communication qubits to enable parallel EPR establishment. Third, our approach conducts a comprehensive evaluation that includes both algorithm-level comparisons with established methods and system-level comparisons with frameworks under similar problem settings. We also evaluate performance across various quantum circuits and DQC topologies. Our contributions can be summarized as follows:

- We propose a novel DQC optimization framework called UNIQ that integrates the three general DQC steps into an NIP model. This approach reduces remote gate communication costs and minimizes total circuit runtime simultaneously.
- Our framework enables partial pre-establishment of EPR pairs by proactively connecting available communication qubits in advance. These pre-established links are then utilized by upcoming remote CNOT gates, effectively reducing the total EPR generation time.
- We conducted extensive simulations on different quantum circuits across various QPU topologies. Our approach outperforms existing algorithms and DQC frameworks, achieving both acceptable algorithm execution time and significantly reduced circuit runtime.

The rest of this paper is organized as follows. Section II introduces the background of quantum computing, DQC, and quantum entanglement. Section III describes our DQC model. Sections IV and V detail the problem formulation and the cor-

TABLE I: Comparison of EPR pair generation latency.

Operation	Variable Name	Latency
Single-qubit gate	t_{1q}	~ 0.1 CX
CX and CZ gate	t_{2q}	1 CX
Measurement	t_{ms}	5 CX
EPR preparation	t_{ep}	~ 12 CX

responding optimization algorithms, respectively. Section VI presents extensive simulations to evaluate the effectiveness of our framework. Finally, we discuss future research directions and summarize our work in Section VII.

II. PRELIMINARIES

In this section, we first introduce the backgrounds of quantum computing [32]–[36], and then outline the general workflow of DQC for solving large-scale quantum circuits.

A. Qubits and Quantum Gates

The basic unit of quantum computing is the qubit, which resides in a two-dimensional Hilbert space spanned by $|0\rangle$ and $|1\rangle$, similar to the 0 and 1 in classical computing. $|0\rangle$ and $|1\rangle$ can be represented as vectors $|0\rangle = (1, 0)^T$ and $|1\rangle = (0, 1)^T$. A quantum state describes the condition of a quantum system and can be classified as either a pure state or a mixed state. A pure state is a deterministic state. In a single-qubit system, a pure state can be either $|0\rangle$ or $|1\rangle$, or a superposition state $|\psi\rangle = \alpha_1|0\rangle + \alpha_2|1\rangle = (\alpha_1, \alpha_2)^T \in \mathbb{C}^2$, where α_1 and α_2 are complex numbers satisfying $|\alpha_1|^2 + |\alpha_2|^2 = 1$. This means that a qubit can represent multiple values simultaneously. In an n qubit system, a pure state can be represented as $|\psi\rangle = \sum_{i=1}^{2^n} \alpha_i |i\rangle \in \mathbb{C}^{2^n}$, where the coefficients satisfy $|\alpha_1|^2 + \dots + |\alpha_{2^n}|^2 = 1$. When the system interacts with the environment or experiences noise, it may evolve into a mixed state. A mixed state is described by a density matrix of the form $\rho = \sum_{i=1}^{2^n} p_i |\psi_i\rangle \langle \psi_i|$, where n is the number of qubits. Each $|\psi_i\rangle$ is a pure state and p_i denotes the probability of the system being in pure state $|\psi_i\rangle$ with $\sum_{i=1}^{2^n} p_i = 1$.

Quantum gates are fundamental operations on qubits, analogous to logical gates in classical computing. Mathematically, each quantum gate is represented by a unitary matrix U , which satisfies $U^\dagger U = U U^\dagger = I$. $U^\dagger$ denotes the conjugate transpose of U , and I is the identity matrix. These gates govern the evolution of quantum states, and different sequences of quantum gates implement different quantum algorithms. Specifically, when the quantum state passes through a quantum gate U , the original states will become a new quantum state $\rho' = U \rho U^\dagger$. Moreover, any multi-qubit quantum gate can be decomposed into a series of basic quantum gates (such as single-qubit gates and CNOT gates). Common single-qubit gates include the X gate ($X|0\rangle = |1\rangle$, $X|1\rangle = |0\rangle$), analogous to the classical NOT gate. Common two-qubit gates include the CNOT gate, which flips the target qubit if the control qubit is $|1\rangle$.

B. Distributed Quantum Computing

DQC integrates multiple small quantum chips (QPU) through network infrastructure to increase the number of available qubits, thereby enabling large-scale quantum computing

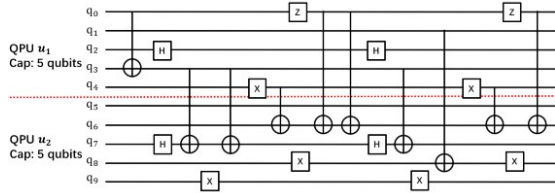


Fig. 1: Solving a large-scale problem via DQC. A quantum circuit is partitioned into two subcircuits, each assigned to a different QPU for parallel execution.

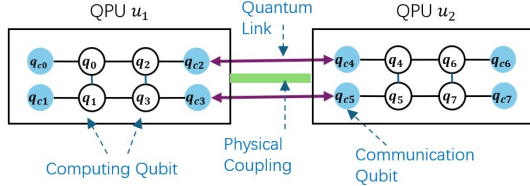


Fig. 2: DQC architecture with two QPUs.

tasks. Take Fig. 1 as an example: assuming that the task requires 10 qubits, while each QPU can only accommodate 5 qubits. Therefore, this circuit must be distributed across at least two QPUs. Figure 2 shows a DQC architecture example consisting of two QPUs. Each QPU contains two types of qubits: computing qubits (white circle) for executing local gates and communication qubits (blue circles) for establishing channels between QPUs, also referred to as quantum links. When two QPUs are physically connected (physical coupling), their communication qubits can establish quantum links, enabling the execution of remote CNOT gates. In addition, although the numbers of computational and communication qubits per QPU in DQC are not identical, they are assumed to be approximately equal. The DQC framework is primarily divided into three components: qubit allocation, entanglement management, and network scheduling.

Qubit allocation involves splitting a quantum circuit and assigning subcircuits to different QPUs in a rational manner. The primary objective is to minimize the number of remote CNOT gates, which occur across QPUs and incur higher communication costs compared to local gates. As an example, shown in Fig. 1, the first CNOT gate operates on two qubits within the same QPU and is referred to as a local CNOT gate; while the second CNOT gate spans two QPUs and is referred to as a remote CNOT gate. Entanglement management is a key technique for implementing remote CNOT gates. Currently, the most widely used method is Cat-Comm, as illustrated in Fig. 3. QPU u_1 and QPU u_2 first generate an EPR pair by establishing a quantum link between their respective communication qubits. This step is the most time-consuming in the entire process, as shown in Table I. The EPR pair is then combined with local gates, measurements, and classical communication to implement a remote CNOT gate between qubits q_0 and q_1 . Another commonly used method is TP-Comm [22]. However, this approach requires measuring the qubit q_0 , which collapses its quantum state. It

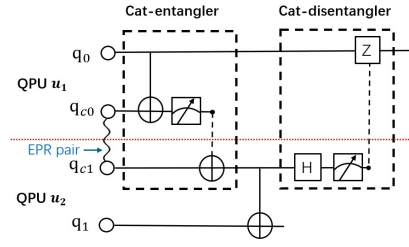


Fig. 3: Cat-Comm implementation of one remote CNOT gate. q_0 and q_1 are computing qubits; q_{c0} and q_{c1} are communication qubits. q_0 and q_{c0} belong to QPU u_1 ; q_1 and q_{c1} belong to QPU u_2 .

alters the original circuit structure and will introduce additional retransmission latency. Therefore, we only adopt Cat-Comm to implement remote gates in this paper. Network scheduling refers to the execution order of quantum gates based on their dependencies. Specifically, any two gates that share at least one qubit must follow a sequential execution order. For example, in Fig. 1, the third CNOT gate can only be applied after the second CNOT has been completed. In contrast, gates can be executed in parallel when quantum resources are available. DAG is commonly used to represent the dependency and execution constraints in quantum circuits. For more details, refer to [31].

III. UNIQ DQC MODEL

This section outlines the fundamental assumptions, operations, and objectives of our proposed UNIQ-DQC model.

Time Slot Modeling and Gate Simplification. We divide the total execution time of the quantum circuit into multiple time slots of length t , where t corresponds to the EPR pair establishment time (t_{ep} in Table I). Since the execution time of a remote gate is significantly longer than that of a local gate, multiple local gates can be executed within a single time slot. Therefore, the execution time of local gates is negligible and can be ignored in our model. We retain only local CNOT gates and remote CNOT gates in the subsequent circuit representation, as illustrated in Fig. 4.

Unified Optimization of DQC Stages. Most of the previous work dealt with the three stages of DQC separately. Each stage is typically optimized with its own objective function. Such design flow lacks global coordination and often results in suboptimal solutions. Indeed, we observe that the three stages of DQC are inherently interdependent. Circuit allocation affects the number and location of remote gates, while these remote gates determine the need and timing of entanglement generation. The above two steps directly constrain the feasibility and efficiency of network scheduling. Therefore, we propose a unified modeling approach to achieve globally optimized and more efficient solutions.

EPR Pre-establishment for Remote Gate Efficiency. Although prior research has recognized that remote CNOT gates are time-consuming and attempted to reduce this (e.g., using the same EPR pair for consecutive remote gates [24]), these strategies face some limitations. They often require that

the involved gates reside on the same pair of QPUs and disallow any intermediate operations. To overcome these constraints, we propose an alternative approach: pre-establishing EPR pairs for future remote gates. When there are still idle communication qubits available in a time slot, additional EPR pairs can be created in advance. These entangled pairs can then be directly used by subsequent remote operations. By establishing multiple EPR pairs in parallel within a single time slot, the overall circuit runtime can be significantly reduced.

Design Objectives of UNIQ-DQC. We employ a unified objective function to jointly optimize the following objectives:

- Minimize the communication cost of remote gates: The QPUs involved in remote gates may not be adjacent, so the system must identify the nearest path to execute the remote gate efficiently.
- Minimize total task runtime under acceptable algorithm execution time: The circuit runtime should be reduced as much as possible, while ensuring that the algorithm's execution time remains within an acceptable cost.

IV. PROBLEM FORMULATION

TABLE II: Summary of symbols and definitions.

Parameters	Description
Q	logical qubits.
$\mathcal{G}$	CNOT gate. $g = (i_g, j_g)$ denote the two logical qubits of CNOT gate g .
$\mathcal{P}$	precedence relations. $(g' \rightarrow g)$ indicates that the gate g' must be executed before the gate g .
$\mathcal{U}$	QPU nodes.
$\mathcal{E} = \{E_u, u \in \mathcal{U}\}$	number of communication qubits on QPU u .
C_{uv}	communication cost on link (u, v) . We set $C_{uu} = 0$.
Cap_u	The maximum number of qubits in QPU u (QPU capacity).
$H = \mathcal{G} $	time-slot upper bound.
α	weight coefficients in the objective function.
Decision Variables	Description
$\pi_{q,u} \in \{0, 1\}$	assign qubit q to QPU u .
$z_{g,t} \in \{0, 1\}$	gate g is executed in slot t .
$y_{g,t} \in \{0, 1\}$	EPR pair for gate g is built in slot t .
Auxiliary Variables	Description
$s_{u,v,t} \in \mathbb{Z}_{\geq 0}$	EPR stored on QPUs (u, v) after time slot t .
$\delta_g \in \{0, 1\}$	$\delta_g = 1$ if logical qubits i_g, j_g are on different QPUs; 0 otherwise.

In this section, we formulate the DQC problem based on the system model described in Section III. The goal is to execute a large-scale quantum task over a DQC network while satisfying the design objectives outlined earlier. To achieve

this, we jointly consider the three interdependent stages of DQC (qubit allocation, entanglement management, and network scheduling) to coordinate quantum gate operations across the entire circuit. The parameters, decision variables, and auxiliary variables used in our formulation are summarized in Table II. First, we introduce our objective function:

$$\min \alpha \sum_{t=1}^H \sum_{g \in \mathcal{G}} t \cdot z_{g,t} + \beta \sum_{t=1}^H \sum_{\substack{u,v \in \mathcal{U} \\ u \neq v}} \sum_{g=(i_g, j_g)} C_{u,v} \cdot \pi_{i_g,u} \cdot \pi_{j_g,v} \cdot z_{g,t}. \quad (1)$$

The objective function consists of two parts: to minimize the total task runtime and to reduce the communication cost of remote CNOT gates. The first component of Eq. (1) aims to schedule all gates as early as possible within the allowed time slots, effectively minimizing the overall circuit runtime. The second component focuses on reducing the communication cost incurred by remote CNOT gates. C_{uv} is defined as the shortest path length between QPU u and QPU v as [31]. If $\pi_{i_g,u} \cdot \pi_{j_g,v} = 1$, it indicates that the two qubits involved in the remote gate g are located on QPUs u and v ; otherwise, they do not. α and β are two weighting parameters to balance these two components.

Next, we will introduce nine constraints of our model. To facilitate understanding, each constraint is illustrated with a corresponding circuit example, as shown in Figs. 4 and 5.

a) *Mapping Validity*: Each logical qubit q must be assigned to exactly one QPU u . As shown in Fig. 4, q_0 and q_1 are mapped to QPU u_1 ; q_2 and q_3 are mapped to QPU u_2 .

$$\sum_{u \in \mathcal{U}} \pi_{q,u} = 1, \quad \forall q \in Q. \quad (2)$$

b) *QPU Capacity*: The number of allocated qubits q on a QPU u cannot exceed its capacity. As shown in Fig. 4, at most two qubits can be assigned to QPU u_1 or u_2 .

$$\sum_{q \in Q} \pi_{q,u} \leq Cap_u, \quad \forall u \in \mathcal{U}. \quad (3)$$

c) *Gate Scheduling*: Each gate g must be and can only be assigned an exact time slot t . As shown in Fig. 4, gate g_1 is executed only at time slot t_1 ; gate g_2 is executed only at time slot t_2 .

$$\sum_{t=1}^H z_{g,t} = 1, \quad \forall g \in \mathcal{G}. \quad (4)$$

d) *Same-QPU Indicator*: A binary variable δ_g indicates whether the two qubits of CNOT gate $g = (i_g, j_g)$ reside on different QPUs. If $\delta_g = 0$, it means that the two qubits of gate g are on the same QPU (local CNOT gate); if $\delta_g = 1$, it means that the two qubits of gate g are on different QPUs (remote CNOT gate). As shown in Fig. 4, g_3 is local CNOT and $\delta_g = 0$; g_4 is remote CNOT and $\delta_g = 1$.

$$\delta_g = 1 - \sum_{u \in \mathcal{U}} \pi_{i_g,u} \pi_{j_g,u}, \quad \forall g \in \mathcal{G}. \quad (5)$$

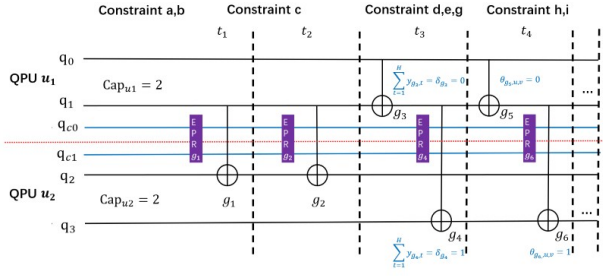


Fig. 4: Example circuit illustrating constraints a-e, g-i.

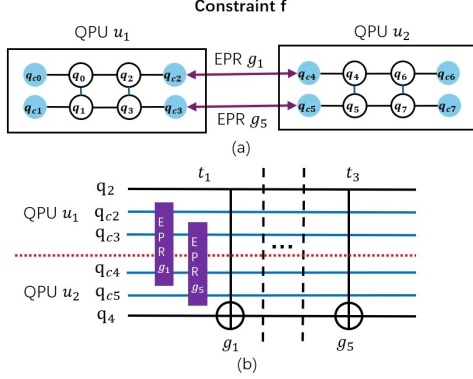


Fig. 5: (a) Two EPR pairs for g_1 and g_5 are generated in parallel and completed together in the same time slot. (b) Example circuit illustrating constraint f.

e) EPR Generation Requirement: If gate g is a remote CNOT gate ($\delta_g = 1$), then one EPR pair for that gate is required. As shown in Fig. 4, g_3 does not require EPR and $\sum_{t=1}^H y_{g_3,t} = 0$; g_4 requires EPR and $\sum_{t=1}^H y_{g_4,t} = 1$.

$$\sum_{t=1}^H y_{g,t} = \delta_g, \quad \forall g \in \mathcal{G}. \quad (6)$$

f) EPR Before Execution Ordering: The EPR pair for the remote gate g must be generated before the gate is executed. As shown in Fig. 5, g_5 is executed at t_3 , and thus its required EPR pair must be established no later than t_3 . Suppose that communication qubits q_{c3} and q_{c5} are idle at t_1 ; in this case, the EPR pair for g_5 can be pre-established at t_1 . Moreover, since two EPR pairs for g_1 and g_5 are generated in parallel within the same time slot, only one EPR setup slot is consumed instead of two, thereby reducing the circuit runtime.

$$\sum_{\tau=1}^t y_{g,\tau} \geq z_{g,t} - (1 - \delta_g), \quad \forall g \in \mathcal{G}, \forall t = 1, \dots, H. \quad (7)$$

g) Precedence: If two CNOT gates have qubit overlap, then they have execution order. As shown in Fig. 4, g_4 must be scheduled no later than g_6 .

$$\sum_{\tau=1}^t z_{g',\tau} \leq \sum_{\tau=1}^t z_{g,\tau}, \quad \forall (g', g) \in \mathcal{P}, \forall t = 1, \dots, H. \quad (8)$$

h) Undirected Mapping Indicator: A dummy variable $\theta_{g,u,v}$ indicates whether gate g spans QPUs u and v . As shown

in Fig. 4, g_5 is local CNOT, and $\theta_{g_5,u,v} = 0$; g_6 is remote CNOT, and $\theta_{g_6,u,v} = 1$.

$$\theta_{g,u,v} = \pi_{i_g,u} \pi_{j_g,v} + \pi_{i_g,v} \pi_{j_g,u}, \quad \forall g \in \mathcal{G}, \forall u, v \in \mathcal{U}, u \neq v. \quad (9)$$

i) Concurrent EPR Generation and Inventory: EPR pairs can be stored and used, subject to inventory limits. As shown in Fig. 4, the number of EPR pairs at t_4 is $s_{u,v,t_4} = 0$. Since there are no remaining EPR pairs from t_3 , one EPR pair is generated at t_4 and one is consumed at the same time.

$$s_{u,v,0} = 0, \quad \forall u, v \in \mathcal{U}, \quad (10)$$

$$s_{u,v,t} = s_{u,v,t-1} + \sum_{g \in \mathcal{G}} y_{g,t} \theta_{g,u,v} - \sum_{g \in \mathcal{G}} z_{g,t} \theta_{g,u,v}, \quad \forall u \neq v, t = 1, \dots, H, \quad (11)$$

$$0 \leq \sum_{v \in \mathcal{U}} s_{u,v,t} \leq E_u, \quad \forall u \in \mathcal{U}, \forall t = 1, \dots, H. \quad (12)$$

V. FRAMEWORK DESIGN

A. Design Overview

Now we introduce our UNIQ-DQC framework. The inputs to the UNIQ include the logical qubit set Q , the CNOT multiset $\mathcal{G}$ with partial order $\mathcal{P}$, a cloud configuration $(\mathcal{U}, \text{Cap}, E, C)$, and a slot horizon $H \leq |\mathcal{G}|$. We adopt a Greedy-JIT (Just-In-Time [37]–[40]) constructor to generate a full execution plan that maps the quantum circuit onto the distributed quantum processor. This plan ensures that all hardware constraints are satisfied, including qubit capacity limits, precedence, and communication requirements between QPUs. It is constructed according to the rules defined by the hybrid QAP-RCPS (Quadratic Assignment Problem-Resource Constrained Project Scheduling Problem) model.

The UNIQ workflow is structured as a two-stage pipeline consisting of qubit allocation and network scheduling. In the first stage, logical qubits are greedily mapped to physical QPUs based on communication weights. This mapping is determined once at the beginning and stored in the π , which remains fixed throughout the process. In the second stage, each gate is scheduled in the earliest available time slot that satisfies communication constraints, based on the DAG precedence and the fixed placement π . If a gate spans two QPUs, the required EPR pair is generated at the same time slot or earlier, thereby avoiding long-term reservation of communication qubits.

By separating qubit allocation and network scheduling, the entire process becomes predictable and consistent, producing the same output for a given input without randomness. This separation also ensures that it runs in polynomial time with respect to the circuit size. Since the scheduling layer does not alter the allocation decisions, both stages can be refined or optimized independently. The Greedy-JIT constructor thus provides a reproducible and efficient warm start solution for more sophisticated optimization algorithms, while ensuring feasibility under all constraints.

B. Greedy Qubit–QPU Mapping

We first fix a qubit–to–QPU assignment, which remains unchanged during the subsequent scheduling stage. This placement phase is feasibility-driven: each logical qubit must be mapped to exactly one QPU and no QPU exceed its qubit capacity (Constraints a–b). Among all feasible placements, we prioritize those that are likely to reduce remote communication costs during execution. To guide this process, we construct an *interaction graph* $G_{\text{int}} = (Q, w)$, whose vertices are the logical qubits and whose edge weights quantify two-qubit gate interactions. Specifically, the weight $w_{ij} = |\{g \in \mathcal{G} \mid \{i_g, j_g\} = \{i, j\}\}|$, $w_{ii} = 0$, counts the number of CNOT gates between qubits i and j . Heuristically, assigning vertices with large w_{ij} to the same QPU reduces the number of remote CNOTs, thereby lowering communication cost and EPR pairs purchased.

The procedure performs a single deterministic sweep over all qubits. For each qubit q , we first compute its total interaction weight $W(q) = \sum_{j \in Q} w_{qj}$ and process the qubits in non-increasing order of W (ties are broken by the qubit index so that the algorithm is reproducible). At each step, only QPUs with remaining capacity are considered. Let $S(q) = \{u \in \mathcal{U} \mid r_u > 0\}$ denote the set of QPUs with remaining capacity, where r_u counts free seats on device u . For each QPU $u \in S(q)$, we evaluate the new communication cost incurred by assigning qubit q to u : $\Delta(q, u) = \sum_{\substack{j \in Q \\ \pi_{j,u}=1}} w_{qj} C_{u,\pi(j)}$, where $\pi(j)$ denotes the QPU currently assigned to qubit j . The sum only includes qubits already placed on QPU u ; interactions with qubits on other devices are unaffected by the choice of u and are thus excluded. The qubit is then assigned to the QPU u^* that minimizes $\Delta(q, u)$ (once again the ties are resolved by the smallest QPU index). The assignment $\pi_{q,u^*} = 1$ is recorded and the residual capacity r_{u^*} decreased. Since a qubit is placed only when $r_{u^*} > 0$, the uniqueness and capacity invariants are preserved throughout, and no backtracking is required.

Once all qubits have been mapped, the placement matrix π is used to derive the gate indicators required by the model. For each gate $g = (i_g, j_g)$, $\delta_g = 1 - \sum_{u \in \mathcal{U}} \pi_{i_g,u} \pi_{j_g,u}$ marks whether the gate is local ($\delta_g = 0$) or remote ($\delta_g = 1$). For each ordered pair $u \neq v$, $\theta_{g,u,v} = \pi_{i_g,u} \pi_{j_g,v} + \pi_{i_g,v} \pi_{j_g,u}$ identifies the (unordered) QPU endpoints of a remote gate. These arrays, along with π , are passed unchanged to the scheduling phase.

Two edge cases are handled explicitly. First, if a qubit never participates in a CNOT ($W(q) = 0$), all candidate devices give $\Delta(q, u) = 0$, and the deterministic tie-breaker chooses the first device with remaining capacity. Second, if at any point, $S(q) = \emptyset$ (that is, the total capacity $\sum_u \text{Cap}_u$ is insufficient), the procedure ends and reports the infeasibility of the instance under Constraints a–b. This entire procedure is summarized in Algorithm 1.

C. JIT Scheduling with EPR Generation

Once the mapping matrix π (and therefore the indicators δ, θ in Constraints d–g) is fixed, each CNOT gate $g \in \mathcal{G}$

Algorithm 1 Greedy Qubit–QPU Mapping

Require: Interaction weights w_{ij} , QPU capacities Cap_u , communication costs C_{uv}
Ensure: Mapping $\pi_{q,u}$ with $\sum_u \pi_{q,u} = 1$ and $\sum_q \pi_{q,u} \leq \text{Cap}_u$

- 1: $\pi_{q,u} \leftarrow 0$ for all q, u ; $r_u \leftarrow \text{Cap}_u$ for all u
- 2: $W(q) \leftarrow \sum_{j \in Q} w_{qj}$ for all q
- 3: $\mathcal{O} \leftarrow$ qubits sorted by non-increasing $W(q)$, breaking ties by index
- 4: **for** $q \in \mathcal{O}$ **do**
- 5: $S(q) \leftarrow \{u \in \mathcal{U} \mid r_u > 0\}$
- 6: **if** $S(q) = \emptyset$ **then**
- 7: **return** INFEASIBLE $\triangleright \sum_u \text{Cap}_u < |Q|$
- 8: **end if**
- 9: $u^* \leftarrow \arg \min_{u \in S(q)} \sum_{\substack{j \in Q \\ \pi_{j,u}=1}} w_{qj} C_{u,\pi(j)}$ $\triangleright$ break ties by the smallest u
- 10: $\pi_{q,u^*} \leftarrow 1$; $r_{u^*} \leftarrow r_{u^*} - 1$
- 11: **end for**

has to be assigned in a time slot $\tau(g) \in \{1, \dots, H\}$ while obeying precedence, single-slot assignment, and the per-slot communication–qubit (EPR) budgets (Constraints c, e–g, i). The scheduler operates on the precedence DAG $(\mathcal{G}, \mathcal{P})$ under a fixed topological order. For each gate g , we compute the earliest slot that can possibly host it, $t_{\min}(g) = 1 + \max\{\tau(g') \mid (g', g) \in \mathcal{P}\}$, with the maximum over the empty set is 0 by convention. This choice directly encodes the precedence relation into a lower bound on the start time, so any slot $t \geq t_{\min}(g)$ automatically respects Constraint f.

Starting from $t_{\min}(g)$, the scheduler linearly scans the timeline and selects the first slot which satisfies all resource constraints. This earliest–feasible rule promotes a compact schedule by greedily minimizing the surrogate objective $\sum_t t z_{g,t}$ and avoids delaying gate execution unnecessarily toward the end of the horizon. Once a feasible slot is found, the gate is assigned to it, which ensures Constraint c.

When $\delta_g = 0$ (both qubits of g reside on the same QPU), the first tested slot is accepted because no inter–QPU communication resource is consumed. In this case, the EPR–related constraints (Constraints e–f, i) are vacuous, whereas precedence (Constraint g) and single-slot assignment (Constraint c) are already satisfied by the earlier construction. When $\delta_g = 1$ (remote CNOT), the scheduler additionally verifies that both endpoint QPUs have sufficient communication capacity at the candidate slot t : $\sum_{v \neq u} s_{u,v,t} < E_u$, for $u = \pi(i_g), \pi(j_g)$, where $s_{u,v,t}$ is the EPR inventory maintained by the recursion (Constraint i). This test counts the EPR pairs already reserved on each endpoint at slot t ; the gate is allowed to reserve an additional pair. Therefore, the inequality ensures the per-slot budget in Constraint i.

Once a feasible execution slot t^* is found, we set $z_{g,t^*} = 1$ and record $\tau(g) = t^*$. For a remote gate, we then choose a generation time $t_{\text{gen}}(g) \in \{1, \dots, \tau(g)\}$, and set $y_{g,t_{\text{gen}}(g)} =$

1. The default choice is *in-slot* generation $t_{\text{gen}}(g) = \tau(g)$, which minimizes the EPR lifetime and thus inventory pressure. Alternatively, we may take the latest feasible earlier slot in $[1, \tau(g) - 1]$ to smooth peak demand. In both cases, the same per-slot budget check is enforced at $t_{\text{gen}}(g)$ for both endpoint QPUs. This satisfies the requirement of generating exactly one EPR pair if and only if the gate is remote (Constraint d) and the ordering requirement that the pair be available no later than execution (Constraint e).

After committing (z, y) , the inventory s is updated through the recursion (Constraint i): one unit is added on the undirected link (u_1, u_2) at $t_{\text{gen}}(g)$ and one unit is removed at $\tau(g)$. For in-slot generation, these two updates occur in the same slot and cancel immediately. Maintaining s in this way guarantees consistency for future capacity checks and ensures that the equalities and bounds in Constraint h hold inductively.

Since the scan for each gate only moves forward in time and the horizon is chosen as $H \geq m$, we are guaranteed to find a feasible slot unless the instance itself is infeasible under the given capacities. Throughout, the algorithm maintains: (i) the fixed completion times $\tau(\cdot)$ for computing new $t_{\text{min}}(\cdot)$; (ii) the binary matrices z and y ; and (iii) s (or an equivalent per-QPU per-slot aggregation from which s can be reconstructed). Since no decision is ever revisited, the routine terminates after a single pass over $\mathcal{G}$. The complete procedure is summarized in Algorithm 2.

D. Theoretical Analysis

The following two theorems establish two key properties of the procedure: (i) it is guaranteed to terminate after a polynomial number of basic computational steps; (ii) it invariably returns a schedule that satisfies the full set of constraints defined in Section IV. Overall, these results confirm that the Greedy-JIT constructor provides a reliable and predictable fast start for any subsequent optimization stage.

Theorem 1 (Convergence). *For every finite instance $(Q, \mathcal{G}, \mathcal{P}, \mathcal{U}, \text{Cap}, E, C, H)$, the Greedy-JIT constructor terminates after at most $O(n \log n + np + m + e + mH + p^2 H) = O(m^2)$ primitive operations when $n, e, H = \Theta(m)$ and $p = o(m)$.*

Proof. The placement routine of Section V-B begins by sorting all n logical qubits, which takes $O(n \log n)$ time, followed by scanning at most p QPUs for each qubit, contributing an additional np operations. Next, a topological sort of the precedence graph (representing gate dependencies) is performed, which visits all m vertices and all e arcs and therefore costs $m + e$ operations. In the scheduling phase (Section V-C), the algorithm inspects at most H candidate slots for each of the m gates. After all gates are fixed, it updates a $p \times p \times (H + 1)$ inventory array, which takes $p^2 H$ steps. Summing these contributions yields the bound stated in the theorem. Under the scaling $n, e, H = \Theta(m)$ and $p = o(m)$, the leading term is m^2 . $\square$

Algorithm 2 JIT schedule with EPR generation

Require: fixed mapping π , indicators δ ; precedence DAG $(\mathcal{G}, \mathcal{P})$; horizon H

```

1:  $z_{g,t} \leftarrow 0, y_{g,t} \leftarrow 0 \quad \forall g, t$ 
2:  $s_{u,v,t} \leftarrow 0 \quad \forall u \neq v, t = 1, \dots, H$ 
3: for  $g$  in topological order of  $(\mathcal{G}, \mathcal{P})$  do
4:    $t \leftarrow 1 + \max_{g' \prec g} \tau(g') \quad \triangleright 0$  if no predecessor
5:   while  $t \leq H$  do
6:     if  $\delta_g = 0$  then
7:        $z_{g,t} \leftarrow 1; \tau(g) \leftarrow t \quad \triangleright$  local gate, no
       cross-QPU resource
8:       break
9:     else
10:       $u_1 \leftarrow \pi(i_g), u_2 \leftarrow \pi(j_g)$ 
11:       $\text{used}_{u_1} \leftarrow \sum_{v \neq u_1} s_{u_1,v,t}, \text{used}_{u_2} \leftarrow \sum_{v \neq u_2} s_{u_2,v,t}$ 
12:      if  $\text{used}_{u_1} + 1 \leq E_{u_1}$  and  $\text{used}_{u_2} + 1 \leq E_{u_2}$ 
       then
13:         $z_{g,t} \leftarrow 1; \tau(g) \leftarrow t$ 
14:         $y_{g,t} \leftarrow 1 \triangleright$  in-slot EPR generation (can be
        moved earlier if desired)
15:         $s_{u_1,u_2,t} + = 1; s_{u_2,u_1,t} + = 1$ 
16:        break
17:      else
18:         $t \leftarrow t + 1$ 
19:      end if
20:    end if
21:  end while
22: end for
```

Theorem 2 (Feasibility). *Let $(\pi, z, y, \delta, \theta)$ be the schedule returned by the constructor. Then all nine constraints a-i from section IV are satisfied.*

Proof. *a-b (mapping).* A qubit is mapped only once and only to a QPU with residual capacity, hence $\sum_u \pi_{q,u} = 1$ and $\sum_q \pi_{q,u} \leq \text{Cap}_u$.

c (gate uniqueness). In Algorithm 2, a gate g is committed to the first feasible slot and the loop breaks immediately (local case: Lines 7–9; remote case: Lines 13–17). Thus exactly one $z_{g,t}$ is set to 1 and $\sum_{t=1}^H z_{g,t} = 1$.

d (same-QPU indicator). After placement δ_g is computed as in Eq. (5), and both cases $\delta_g \in \{0, 1\}$ are obtained directly from π .

e-f (EPR requirements). If $\delta_g = 0$, no EPR is generated ($\sum_t y_{g,t} = 0$); if $\delta_g = 1$, the scheduler sets $y_{g,\tau(g)} = 1$ and no other entry, so $\sum_t y_{g,t} = 1$ and $\sum_{\tau \leq t} y_{g,\tau} \geq z_{g,t} - (1 - \delta_g)$.

g (precedence). Because each candidate slot t is tested in non-decreasing order starting from $1 + \max_{g' \prec g} \tau(g')$, we have $\tau(g) > \tau(g')$ for all $g' \prec g$, and therefore $\sum_{\tau \leq t} z_{g',\tau} \leq \sum_{\tau \leq t} z_{g,\tau}$.

h (unordered QPU pair). After mapping, $\theta_{g,u,v}$ is set exactly as required by Constraint h.

i (EPR inventory). The inventory is initialized at Algorithm 2 Line 3 with $s_{u,v,0} = 0$ and updated at Line 16 is

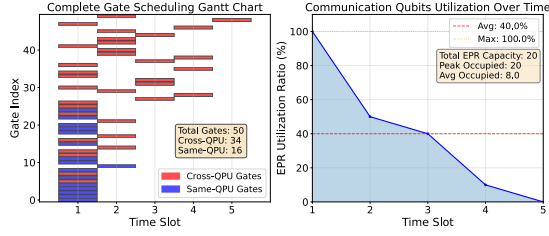


Fig. 6: Performance of representative example.

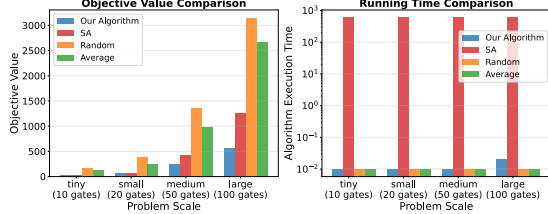


Fig. 7: Performance on different user-defined circuits.

consistent with the recursion in Constraint i. Before accepting a remote gate, the algorithm checks (Algorithm 2 Lines 12–13) that $\sum_{v \neq u} s_{u,v,t} + 1 \leq E_u$ for both endpoints, which enforces the per-slot budget. Hence $0 \leq \sum_v s_{u,v,t} \leq E_u$ for all u, t , and the update equations in Constraint i hold.

Therefore, each constraint condition has been satisfied. $\square$

VI. EVALUATION

TABLE III: Characteristics of user-defined circuits.

Circuits	Gates	Qubits	QPU	Comput_Qubit	Commu_Qubit
tiny	10	10	2	5	10
small	20	15	3	5	10
medium	50	32	4	8	10
large	100	60	5	12	10

TABLE IV: Characteristics of real-world quantum circuits.

Circuits	Qubits	2 Qubit Gates	Circuit Depth
bv_n70	70	36	40
cat_130	20	15	3
ghz_n127	50	32	4
qugan_n111	100	60	5

A. Evaluation Setting

Implementation. Since no public simulator supports distributed quantum clouds with per-slot EPR budgets, we implemented a lightweight discrete-event simulator in Python. Quantum circuits are parsed via Qiskit [41], and graph routines rely on NetworkX [42]. The greedy constructor and the simulated-annealing improver are written in pure NumPy for reproducibility and speed. All experiments were run on a single CPU core unless stated otherwise.

Topology Settings. Unless otherwise specified, our DQC framework is configured with 5 QPUs by default, each equipped with 20 computing qubits and 10 communication qubits. The inter-QPU topology is randomly generated, meaning that the paths between QPUs are assigned at random.

Evaluation Metrics. We evaluate system performance using four metrics: circuit runtime, algorithm execution time, the objective value defined in Eq. (1), and EPR pairs utilization. UNIQ is designed to minimize the first three metrics while maximizing EPR pairs utilization.

B. Representative Example

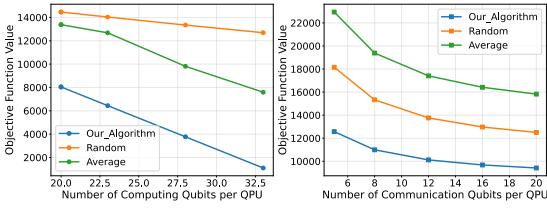
We demonstrate UNIQ effectiveness through a representative example involving a quantum circuit with 50 qubits and 50 CNOT gates, each assigned to one time slot. The gates are sequentially indexed from 1 to 50 and randomly placed across the qubits. The left diagram shown in Fig. 6 visualizes the time slot scheduling of the gates, where red boxes denote remote CNOT gates and blue boxes denote local CNOT gates. For clarity, we provide an example explanation: gates indexed 1–5 are local gates and are all scheduled in t_1 ; gates indexed 11, 17, 21 are remote gates and are all scheduled in t_2 . This dense packing of gates in early slots highlights UNIQ’s ability to prioritize independent operations, thereby minimizing total circuit runtime. The right diagram presents the EPR pairs utilization across time slots. A clear concentration of EPR consumption in earlier time slots reflects UNIQ’s efficiency in managing limited communication resources.

C. Algorithm Comparison

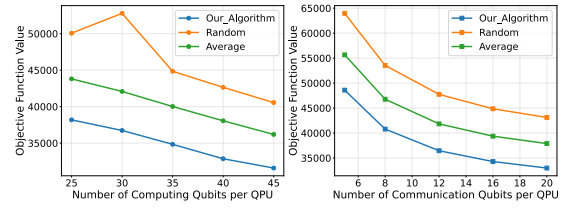
We compare UNIQ with existing algorithms across various quantum circuits and QPU topologies. The following algorithms are used as baselines. a) Simulated Annealing (SA): A heuristic method that probabilistically escapes local optima to find near-optimal solutions. b) Random: Randomly map qubits to QPUs, sample any precedence-respecting gate order, and defer conflicts to the earliest feasible slot. c) Average: Inter-QPU communication capacity is uniformly distributed, giving each cross gate an equal share.

Different user-defined circuits. Table III presents the characteristics of user-defined circuits with various scales. The results are shown in Fig. 7, where the left plot compares the objective values, and the right plot presents the corresponding algorithm execution times. All algorithms are configured to return a feasible solution within a 600-second time limit. The results demonstrate that UNIQ consistently performs well across all task sizes, especially on large circuits, while maintaining extremely short execution times (0.01 seconds). In contrast, the execution time of the SA algorithm is significantly longer (exclude it from subsequent simulations), and the objective values generated by the Random and Average algorithm are much higher.

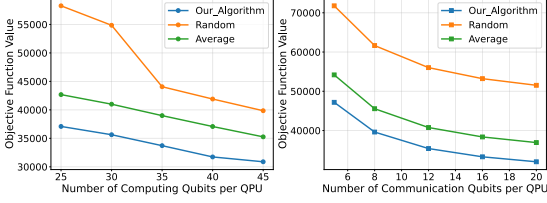
Different computing or communication qubits. We selected four real-world quantum circuits as representative benchmarks, whose characteristics are summarized in Table IV. We vary the number of computing and communication qubits independently to evaluate their impact on performance. As shown in Fig. 8, increasing either type of qubit generally leads to lower objective values. Throughout all settings, our algorithm consistently achieves the lowest objective value, outperforming all baselines.



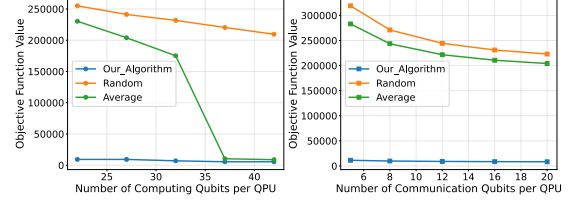
(a) bv_n70 circuit



(b) cat_n130 circuit



(c) ghz_n127 circuit



(d) qugan_n111 circuit

Fig. 8: Impact of computing or communication qubit counts across four real-world quantum circuits.

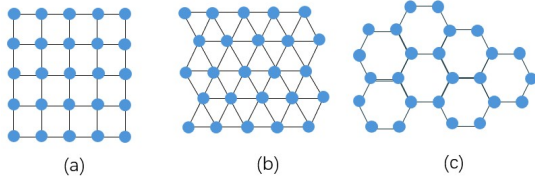


Fig. 9: QPU topologies. (a) Square topology, $E/N=1.60$. (b) Triangle topology, $E/N=2.24$. (c) Hexagonal topology, $E/N=3$.

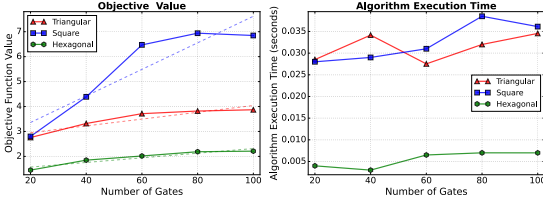


Fig. 10: Comparison of QPU topologies.

Different QPU topologies. The three topology structures are shown in Fig 9. Each topology contains 25 nodes (25 QPUs), and the edge to node ratio E/N (E is the number of edges and N is the number of nodes) reflects the degree of topology connectivity. As shown in Fig. 10, a higher E/N ratio leads to better system performance, as higher connectivity improves the flexibility of gate scheduling and reduces communication cost.

D. Framework Comparison

We compare UNIQ with existing DQC frameworks. To ensure a fair comparison, the baseline and our framework should under similar problem settings. Specifically, they should cover all three stages of DQC and adopt the Cat-Comm protocol for remote gate execution. Inspired by CloudQC [31], UNIQ is also suitable for multi-tenant scenarios. Specifically, our method can be viewed as globally sorting all CNOT gates across multi-tenants. Based on this similarity, we select CloudQC as the baseline framework for comparison.

We conducted three simulations for each circuit scale. As shown in Fig. 11(a), x- and y-coordinates of each blue point

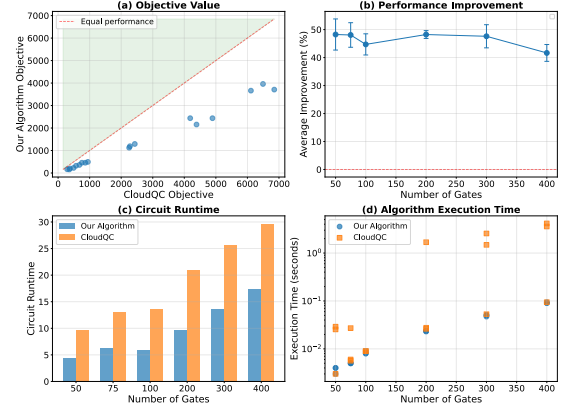


Fig. 11: Comparison with CloudQC.

represent the average objective values of CloudQC and UNIQ across different circuit scales. For instance, the last blue point indicates that CloudQC's objective is close to 7000, while UNIQ's is around 4000. Figure 11(b) further demonstrates that UNIQ significantly reduces the objective value—by nearly 50% compared to CloudQC. Figure 11(c) presents the average circuit runtime of CloudQC and UNIQ, while each point in Fig. 11(d) shows the comparison of algorithm execution time under each simulations. Across all evaluations, UNIQ consistently outperforms CloudQC in both circuit runtime and algorithm execution time.

VII. CONCLUSION

In this paper, we propose UNIQ, a novel optimization framework for DQC network. UNIQ unifies the three fundamental stages of the DQC workflow (qubit allocation, entanglement management, and gate scheduling) into a single NIP model, thereby obtaining a more optimal feasible solution. Furthermore, UNIQ proactively exploits idle communication qubits to pre-establish the time-consuming EPR pairs, enabling the parallel generation of multiple EPR pairs. This strategy significantly reduces the execution time of remote CNOT gates. We conducted comprehensive simulations across diverse

circuits and QPU topologies. Compared to existing algorithms and DQC frameworks, UNIQ minimizes total circuit runtime while reducing the communication cost of remote gates.

REFERENCES

- [1] J. Bub, “Quantum computation: Where does the speed-up come from,” *Philosophy of quantum information and entanglement*, January 2010.
- [2] R. Jozsa and N. Linden, “On the role of entanglement in quantum-computational speed-up,” *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 459, no. 2036, pp. 2011–2032, August 2003.
- [3] E. Rieffel and W. Polak, “An introduction to quantum computing for non-physicists,” *ACM Computing Surveys (CSUR)*, vol. 32, no. 3, pp. 300–335, September 2000.
- [4] R. Rietsche, C. Dremel, S. Bosch, L. Steinacker, M. Meckel, and J.-M. Leimeister, “Quantum computing,” *Electronic Markets*, vol. 32, no. 4, pp. 2525–2536, August 2022.
- [5] F. Bova, A. Goldfarb, and R. G. Melko, “Commercial applications of quantum computing,” *EPJ quantum technology*, vol. 8, no. 1, p. 2, January 2021.
- [6] L. Gyongyosi and S. Imre, “A survey on quantum computing technology,” *Computer Science Review*, vol. 31, pp. 51–71, February 2019.
- [7] O. Engkvist, P.-O. Åstrand, and G. Karlström, “Accurate intermolecular potentials obtained from molecular wave functions: Bridging the gap between quantum chemistry and molecular simulations,” *Chemical Reviews*, vol. 100, no. 11, pp. 4087–4108, October 2000.
- [8] Y. Cao, J. Romero, and A. Aspuru-Guzik, “Potential of quantum computing for drug discovery,” *IBM Journal of Research and Development*, vol. 62, no. 6, pp. 6–1, December 2018.
- [9] S. Wilkens and J. Moorhouse, “Quantum computing for financial risk measurement,” *Quantum Information Processing*, vol. 22, no. 1, p. 51, January 2023.
- [10] K. Ju, X. Qin, H. Zhong, X. Zhang, M. Pan, and B. Liu, “Harnessing inherent noises for privacy preservation in quantum machine learning,” in *IEEE International Conference on Communications (ICC)*, Denver, CO, June 2024.
- [11] Y. Zhao, H. Zhong, X. Zhang, C. Zhang, and M. Pan, “Bridging quantum computing and differential privacy: a survey on quantum computing privacy,” *arXiv e-prints*, pp. arXiv–2403, March 2024.
- [12] M. Brooks, “Beyond quantum supremacy: the hunt for useful quantum computers,” *Nature*, vol. 574, no. 7776, pp. 19–22, October 2019.
- [13] H. Zhong, K. Ju, M. Sista, X. Zhang, A. Li, X. Qin, X. Fu, and M. Pan, “Tuning quantum computing privacy through quantum error correction,” in *IEEE Global Communications Conference (GLOBECOM)*, Cape Town, South Africa, December 2024, pp. 3986–3991.
- [14] J. Preskill, “Quantum computing in the nisc era and beyond,” *Quantum*, vol. 2, p. 79, August 2018.
- [15] M. Brink, J. M. Chow, J. Hertzberg, E. Magesan, and S. Rosenblatt, “Device challenges for near term superconducting quantum processors: frequency collisions,” in *IEEE International Electron Devices Meeting (IEDM)*, San Francisco, CA, December 2018.
- [16] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, “Trapped-ion quantum computing: Progress and challenges,” *Applied physics reviews*, vol. 6, no. 2, June 2019.
- [17] H. E. Brandt, “Qubit devices and the issue of quantum decoherence,” *Progress in Quantum Electronics*, vol. 22, no. 5-6, pp. 257–370, September 1999.
- [18] M. Caleffi, M. Amoretti, D. Ferrari, J. Illiano, A. Manzalini, and A. S. Cacciapuoti, “Distributed quantum computing: a survey,” *Computer Networks*, vol. 254, p. 110672, December 2024.
- [19] D. Barral, F. J. Cardama, G. Diaz-Camacho, D. Faílde, I. F. Llovo, M. Mussa-Juane, J. Vázquez-Pérez, J. Villasuso, C. Piñeiro, N. Costas *et al.*, “Review of distributed quantum computing: from single qpu to high performance quantum computing,” *Computer Science Review*, vol. 57, p. 100747, August 2025.
- [20] A. S. Cacciapuoti, M. Caleffi, F. Tafuri, F. S. Cataliotti, S. Gherardini, and G. Bianchi, “Quantum internet: Networking challenges in distributed quantum computing,” *IEEE Network*, vol. 34, no. 1, pp. 137–143, November 2019.
- [21] R. Beals, S. Brierley, O. Gray, A. W. Harrow, S. Kutin, N. Linden, D. Shepherd, and M. Stather, “Efficient distributed quantum computing,” *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 469, no. 2153, p. 20120686, May 2013.
- [22] 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 *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Chicago, IL, October 2022.
- [23] A. Wu, Y. Ding, and A. Li, “Qucomm: Optimizing collective communication for distributed quantum computing,” in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, New York, NY, December 2023.
- [24] D. Ferrari, S. Carretta, and M. Amoretti, “A modular quantum compilation framework for distributed quantum computing,” *IEEE Transactions on Quantum Engineering*, vol. 4, pp. 1–13, August 2023.
- [25] J. Liu, L. Fan, Y. Guo, Z. Han, and Y. Wang, “Co-design of network topology and qubit allocation for distributed quantum computing,” in *International Conference on Quantum Communications, Networking, and Computing (QCNC)*, March 2025.
- [26] M. G. Davis, J. Chung, D. Englund, and R. Kettimuthu, “Towards distributed quantum computing by qubit and gate graph partitioning techniques,” in *IEEE International Conference on Quantum Computing and Engineering (QCE)*, Bellevue, WA, September 2023.
- [27] T.-Y. Luo, Y.-Z. Zheng, X. Fu, and Y.-X. Deng, “Automatic architecture design for distributed quantum computing,” *Chinese Physics B*, vol. 33, no. 12, p. 120302, November 2024.
- [28] N. K. Chandra, E. Kaur, and K. P. Seshadreesan, “Network operations scheduling for distributed quantum computing,” in *International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA)*, Washington, DC, October 2024.
- [29] Y. Mao, Y. Liu, and Y. Yang, “Qubit allocation for distributed quantum computing,” in *IEEE International Conference on Computer Communications (INFOCOM)*, New York, NY, May 2023.
- [30] S. Kan, Z. Du, M. Palma, S. A. Stein, C. Liu, W. Wei, J. Chen, A. Li, and Y. Mao, “Scalable circuit cutting and scheduling in a resource-constrained and distributed quantum system,” in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Montreal, QC, Canada, September 2024.
- [31] R. Zhou, Y. Gan, Y. Liu, and C. Qian, “Cloudqc: A network-aware framework for multi-tenant distributed quantum computing,” *arXiv preprint arXiv:2504.20389*, April 2025.
- [32] K. Ju, X. Qin, H. Zhong, X. Zhang, M. Pan, and B. Liu, “Privacy preserving quantum search mechanism using grover’s algorithm,” in *International Conference on Quantum Communications, Networking, and Computing (QCNC)*, Kanazawa, Japan, July 2024, pp. 204–210.
- [33] T. Hey, “Quantum computing: an introduction,” *Computing and Control Engineering*, vol. 10, no. 3, pp. 105–112, June 1999.
- [34] V. Vedral and M. B. Plenio, “Basics of quantum computation,” *Progress in quantum electronics*, vol. 22, no. 1, pp. 1–39, January 1998.
- [35] L. Fan and Z. Han, “Hybrid quantum-classical computing for future network optimization,” *IEEE Network*, vol. 36, no. 5, pp. 72–76, 2022.
- [36] C. Hirche, C. Rouzé, and D. S. França, “Quantum differential privacy: An information theory perspective,” *IEEE Transactions on Information Theory*, vol. 69, no. 9, pp. 5771–5787, May 2023.
- [37] Y. Zhao, K. Damevski, and H. Chen, “A systematic survey of just-in-time software defect prediction,” *ACM Computing Surveys*, vol. 55, no. 10, pp. 1–35, February 2023.
- [38] W. Zhan, R. Mo, Y. Jiang, and D. Wang, “Just-in-time prediction of software architectural changes through commit-level analyses,” *IEEE Transactions on Software Engineering*, July 2025.
- [39] A. Corominas, A. García-Villoria, N.-A. González, and R. Pastor, “A multistage graph-based procedure for solving a just-in-time flexible job-shop scheduling problem with machine and time-dependent processing costs,” *Journal of the Operational Research Society*, vol. 70, no. 4, pp. 620–633, April 2019.
- [40] J. Bryan and P. Moriano, “Graph-based machine learning improves just-in-time defect prediction,” *Plos one*, vol. 18, no. 4, p. e0284077, April 2023.
- [41] G. Aleksandrowicz, J. Gambetta, M. Treinish, C. Wood *et al.*, “Qiskit: An open-source framework for quantum computing,” January 2019.
- [42] A. Hagberg, P. J. Swart, and D. A. Schult, “Exploring network structure, dynamics, and function using networkx,” in *Python in Science Conference (SciPy)*, Pasadena, CA, January 2008.