Dynamic Distribution of Quantum Circuits with Minimum Circuit-Execution Time

***Abstract*—Distribution of quantum circuits (DQC) is a promising strategy to enable large-scale quantum computations by utilizing a network of quantum computers. However, executing distributed quantum programs entails generating entanglements (to execute remote gates), which can incur significant latency and lead to qubits' decoherence. Prior works on the problem of distributing quantum circuits with minimum circuit-execution time have only considered a *static* allocation of qubits to network nodes, presumably for simplicity. In this work, we show that distributing quantum circuits with a dynamic allocation of qubits, i.e., qubit allocation (over the quantum computers) that changes during the circuit execution, can lower circuit execution time. For this DQC problem wherein qubit allocation can be dynamic (i.e., change over time), we design efficient algorithms and demonstrate the effectiveness of designed techniques via extensive simulations over NetSquid, a quantum network simulator; our techniques outperform the best prior work (using static qubit allocation) by up to 50%.**

I. Introduction

Scalability is a cornerstone of advancing quantum computing, as the ability to handle larger numbers of qubits is crucial for solving complex problems in fields like cryptography, optimization, and materials science. However, scaling quantum systems faces significant challenges, including physical constraints on qubit count, maintaining qubit coherence, and mitigating noise and errors in larger systems. Error-correcting codes can be used to overcome noisy gate operations, but that results in a blowup in the number of qubits, which further exacerbates the qubit capacity hurdle. Distributing quantum circuits over a quantum network solves these challenges by connecting smaller quantum computers (QCs) through quantum communication channels. However, almost all quantum communication protocols require entangled pairs (to execute remote gates) which can incur significant generation latency and, thus, lead to decoherence of qubits. Thus, in this work, we consider the problem of distributing quantum circuits across a quantum network with the optimization objective of minimizing the circuit execution time, which includes latency incurred in generating the required entanglements to execute the remote gates under decoherence constraints.

Distributing quantum circuits entails mapping the circuit's qubits to the quantum network's qubit memories, determining the best quantum communication operations to execute remote gates (gates spanning multiple QCs), and concurrently generating the entangled pairs required for communication. In our proposed approach, we first determine an initial allocation of qubits to qubit memories so that the estimated execution time is minimized. We then select an efficient set of teleportations and telegates (or cat-entanglements) to execute remote gates; teleportations result in a dynamic qubit allocation, but help minimize the circuit execution time. Finally, we use efficient strategies to execute the gates using required entanglements with minimum latency under network resource and decoherence constraints.

Prior Work and Our Approach. The problem of distributing quantum circuits in quantum networks has gained significant attention in recent years, resulting in the development of efficient solutions tailored to various settings and objectives. However, almost all works on distributing quantum circuits have focused on the objective of minimizing the number of maximally-entangled pairs (EPs) either by minimizing the number of cat-entanglements [9, 13] or the number of teleportations [14, 18, 22]. Two recent works have focused on the objective of minimizing circuit execution time. In particular, [4] minimizes, the number of time slots to generate EPs required to execute the remote gates; they make the simplistic assumptions: (i) each EP 's generation takes a single unit of time, and (ii) allocation of qubits to computers is given. In the work closest to ours, [19] focuses on minimizing the execution time of a quantum circuit by concurrently generating EPs. They, however, restrict the designed techniques to use a *static* qubit allocation (by disallowing teleportation); i.e., their designed algorithms choose a fixed qubit allocation at the start of the circuit execution, and use this qubit allocation to determine the remote gates and the EPs that need to be generated. However, as shown in our work, a lower circuit execution time can be achieved by allowing the qubit allocation to change during circuit execution; this is the focus of our work. In particular, in this work, we consider the problem of distributing quantum circuits to minimize the circuit execution time under the network resource and decoherence constraints while allowing for a dynamic allocation of qubits (by allowing the use of teleportation of qubits during circuit execution).

Our Contributions. In the above-described context of distributing quantum circuits with a dynamic qubit allocation, we make the following contributions:

- We formulate the dynamic distributed quantum computing problem and show that, in some cases, a dynamic qubit allocation may result in a lower circuit execution time than when the qubit allocation is constrained to be static.
- We design multiple two-step algorithms, which entail determining the *initial* qubit allocation in the first step, and then determining the "execution scheme" in the second step. The execution step, in turn, involves determining the communication steps (viz., teleportations, telegates, or cat-entanglements) and generating the desired EPs.
- For the second step of developing efficient execution schemes, we design two efficient algorithms: (i) A dynamic programming algorithm that determines a series of points in the input circuit at which the qubit allocation is

changed via teleportations; other communication means (telegates or cat-entanglements) are then used to execute remote gates within the resulting sub-circuits. (ii) An online algorithm that scans the circuit from left to right (or right to left) and, for each remote gate encountered, it determines how to execute the gate best (e.g., via teleporting the operand qubits to a single quantum computer, or by executing the gate in-place using a telegate or cat-entanglement).

- We demonstrate the effectiveness of our techniques by evaluating them on randomly generated circuits and known benchmarks (§VIII) over NetSquid [3], and observe that our techniques outperform the best-known prior techniques by up to 50%.

II. Background

Quantum Circuit Representation. We represent an *abstract quantum circuit* C over a set of qubits $\mathcal{Q} = \{q_1, q_2, \ldots\}$ as a sequence of gates $\langle g_1, g_2, \ldots\rangle$ where each g_t is either binary $CNOT$ gate or a unary gate (a universal set of gates). We represent binary gates as triplets (q_i, q_j, t) where q_i and q_j are the two operands and t is the time instant (defined below) of the gate in the circuit, and unary gates as pairs (q_i, t) where q_i is the operand and t is the time instant.

Time Instants. Each gate occurs at a time instant. In addition to the instants where the gates occur, we introduce time instants between consecutive gates and enforce that teleportations occur only at these additional time instants so that they don't occur concurrently with the gate operations.

Quantum Network (QN). A quantum network is a network of quantum computers (QCs) represented as a connected graph with nodes as QCs and edges representing (quantum and classical) communication links. Each computer has a certain amount of quantum memory to store the data/circuit qubits.

A. Distribution of Quantum Circuits over QNs

Given a quantum network (QN) and a quantum circuit, we seek to distribute and execute the given circuit over the network so that the execution incurs minimum time. Distribution of a circuit over a network essentially entails two aspects: (i) distributing the circuit qubits across the QCs/nodes of the QN, and (ii) executing the "remote" binary gates (i.e., gates with operands on different QCs) efficiently.

Executing Remote Quantum Gates. To execute such remote gates, we need to bring all operands' values into a single QC via quantum communication. Since direct transmission of qubit is subject to unrecoverable errors, we consider the following ways to communicate qubits across network nodes.

Teleportation. An alternative approach to physically transmit a qubit from a node A to B is via *teleportation* [2] which requires an a priori distribution of an EP over A and B. See Fig. 1(a). Teleportation can be used to bring operands of a remote gate to a single QC for local execution.

Telegates. One way to execute remote gates without communicating the gate operands are via the telegate protocol [4].

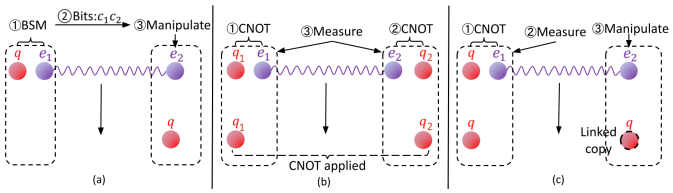

Fig. 1: Quantum communication. (a) Teleportation, (b) Telegate, (c) Cat-Entanglement. Dashed boxes are the network nodes; the initial (final) state of the qubits is at the top (bottom).

The telegate protocol (see Fig. 1(b)) is a sequence of local operations that effectuates the execution of a remote $CNOT$ gate with operands at nodes A and B; the protocol requires an a priori distribution of an EP over A and B.

Cat-Entanglement: Creating "Linked Copies" of a Qubit. Yet another approach to execute remote gates is by creating *linked read-only copies* of a qubit across QCs via *cat-entanglement* operations [7, 21]. Creating a linked copy of a qubit q at node A to a node B requires a priori distribution of an EP over A and B. See Fig. 1(c). The linked copy at B can be used to execute binary gates where q is the (read-only) control qubit, until a unary operation needs to be executed on q. When a unary operation needs to be executed on q, a disentanglement operation is done, which destroys the linked copies, and then, the unary operation can be done on the original qubit q at A.

Teleportation vs. Telegates vs. Cat-Entanglements. First, observe that each of these three communication protocols requires a single EP. Teleportation transports the qubit operand to the computer where the gate is executed. In contrast, the telegate protocol executes a remote gate without moving qubits. The main advantage of cat-entanglement is that one cat-entanglement (and, thus, one linked copy) can sometimes be used to execute several binary gates; however, cat-entanglements are best used for circuits with only *CZ* binary gates (see §VII). In this work, we consider all the above communication mechanisms, viz., teleportations, telegates, and cat-entanglements, for executing remote gates.

B. Generating EPs over Remote Nodes

Each of the above communication modes requires an apriori distributed EP. In a quantum network, EPs over long distances are generated using a sequence of "entanglement-swapping" operations over shorter-distance EPs. The stochastic nature of the ES operations entails that an EP is successfully generated only after many failed attempts; thus, the generation of an EP incurs significant *generation latency*. To generate a set of EPs concurrently, we must allocate the network resources appropriately to each EP's generation. *The techniques developed here work do not assume any particular model for generation latencies, except that a (monotonic) function is available that returns total generation latency for generating a set of EPs concurrently.*

III. Problem Formulation and Related Work

We start by defining some terms and models before moving on to the problem formulation.

Qubit-Allocation Function. Let the set of network nodes/QCs in the given quantum network be $\mathbb{P} = \{P_1, P_2, \ldots, P_N\}$, where each QC P_i is equipped with a set $Q(P_i)$ of qubit memories for data storage. For a given quantum circuit C, let $Q(C)$ be the set of qubits in C. To execute a circuit over a quantum network, we must first distribute the circuit qubits over the qubit memories in the QCs. To that end, we define the *qubit allocation function* η as the function that maps the circuit qubits $Q(C)$ to the network qubits $\bigcup_i Q(P_i)$, i.e., $\eta : Q(C) \rightarrow \bigcup_i Q(P_i)$; the qubit-allocation function must be one-to-one, i.e., only one circuit-qubit can be mapped to a qubit memory in the network.

Remote and Local Gates. For a given circuit C and a qubit allocation function η, a gate (q_i, q_j, t) in C is considered to be *remote* if q_i and q_j are assigned to different *computers* by the qubit allocation function, i.e., $\eta(q_i) \in Q(P_i)$ and $\eta(q_j) \in Q(P_j)$ where $i \neq j$. Every gate that is *not* a remote gate is considered a *local gate*.

Static vs. Dynamic Qubit Allocation. Prior works have focused on distributing quantum circuits under the restriction of static qubit allocations. They start with an initial allocation of qubits and execute the resulting remote gates using telegates. However, allowing the qubit allocation function to change during the circuit execution could result in fewer subsequent remote gates and lower execution time. See Example 1 below. Allowing such dynamic allocation of qubits entails determining when to perform teleportations. We now formally define the dynamic distribution of quantum circuits problem.

Dynamic Distribution of Quantum Circuits (DDQC) Problem. Given a quantum network and a quantum circuit C, the DDQC problem is to distribute and execute C over the given network with minimum circuit execution time under the network constraints (in particular, memory at each network node, EP generation resources, and decoherence). The DDQC problem entails (i) selecting an initial qubit-allocation function, (ii) identifying teleportations and telegates to execute remote gates, and (iii) determining an execution scheme involving the generation of necessary EPs and execution of gates such that the total circuit execution time is minimized. See Fig. 4 (Example 1).

In the above formulation, for clarity, we only use teleportations and telegates to execute remote gates; we discuss using cat-entanglements in §VII. The above DDQC problem can be shown to be NP-Hard by a reduction from the min-quadratic-assignment problem [16].

A. Related Work

The distribution of quantum circuits (DQC) problem has been studied before in various settings and objectives. Prior work can be categorized into two broad categories: works that minimize the *number* of EPs required and works that minimize the *circuit execution time*. We discuss these below.

Minimizing the Number of EPs. The DQC problem is to distribute a given quantum circuit over a quantum network with some optimization objective. In contrast to the DDQC problem, the DQC problem ignores the coupling graphs within each computer [5, 9, 13, 14], and thus, the swap operations needed. The optimization objective used in almost all of these works has been to minimize the communication cost—defined as the *number* of EPs used; a couple of recent works [11, 12] consider EPs with arbitrary (but fixed) cost and develop a simulated-annealing heuristic for the qubit-allocation part of the DQC problem. Our schemes in this work use a similar two-step strategy as some [9, 17, 18] of the DQC works, but otherwise differ significantly as we need to consider the stochastic and concurrent generation of required EPs to minimize execution time; we also consider telegates to execute remote gates while the two-step DQC works consider entanglements [9, 17] and/or teleportations [17, 18].

Minimizing Circuit Execution Time. The works close to ours [4, 8, 19] address the DQC problem with minimizing circuit execution time, but with a static qubit allocation, slightly different settings, and certain limitations, as discussed below. In particular, [8] focuses on estimating only the *worst-case* overhead in executing a circuit over a network, by the worst-case linear network topology; the overhead considered is in terms of an increase in circuit "layers" and EPs required to execute remote gates. [4] focuses on the efficient execution of remote gates using telegates, *given* a qubit allocation (they consider qubit allocation to be out of the scope of their work), to minimize the number of circuit layers; they assume generation latency of each EP to be uniform (one time-slot), and ignore decoherence constraints.

In the work closest to ours, [19] focuses on minimizing the execution time of a quantum circuit by concurrently generating EPs. They consider a simple model with a static qubit allocation function. That is, a single allocation function is chosen at the start of the algorithm, and this allocation determines the EPs that need to be generated. In §VIII, we compare our approaches with their best approach and observe a significant reduction in circuit execution time.

IV. High-Level Approach

In the previous section, we formulated the DDQC problem as distributing a given circuit over a given network with minimum execution time. As mentioned above, the key challenges or subproblems in solving the DDQC problem is to determine the qubit-allocation function and the execution scheme such that the execution time is minimized. Thus, it is natural to tackle the DDQC problem as a sequence of two steps, similar to the approaches taken in prior works on the similar problem of distributing quantum circuits [9, 17, 18].

1) *Determine the Initial Qubit-Allocation Function.* In this step, we determine an initial qubit-allocation function that will yield an execution scheme in the second step with minimum total circuit execution time. Note that this allocation function may change in the second step as teleportations are performed.
2) *Determine the Execution Scheme.* In this step, given an initial qubit-allocation function, we determine the execution scheme, which entails two aspects: (i) First, we need

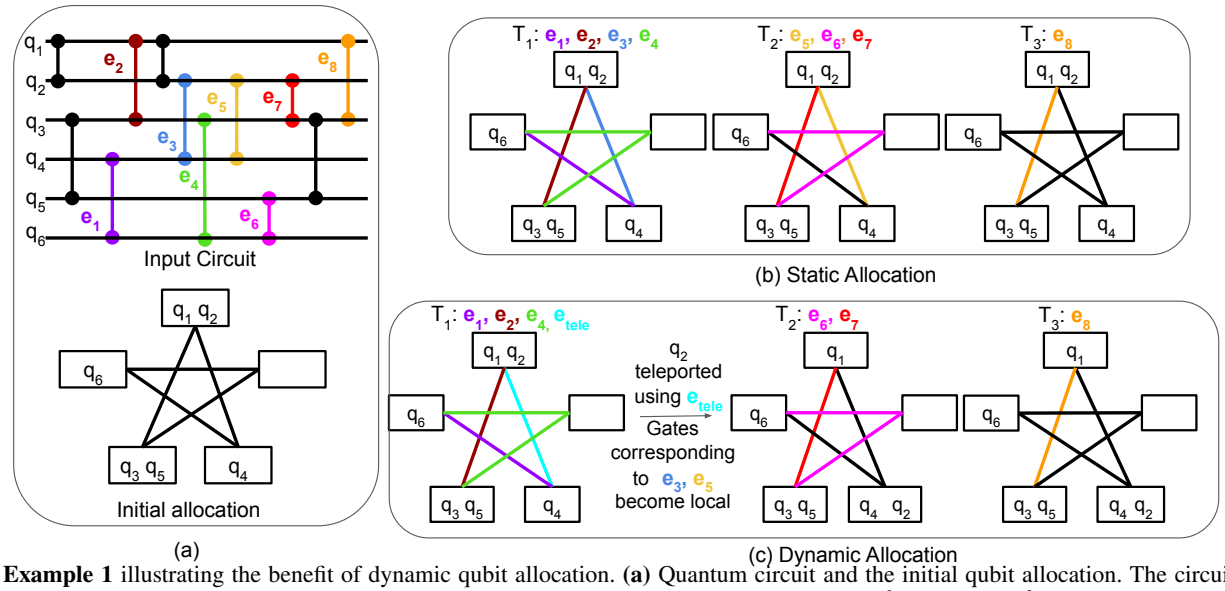

Fig. 2: **Example 1** illustrating the benefit of dynamic qubit allocation. **(a)** Quantum circuit and the initial qubit allocation. The circuit shows the remote gates (based on the qubit allocation) colored and labeled with the EPs required; thus, $\{e_1, e_2, \ldots, e_8\}$ represent the EPs required to execute the corresponding remote gates. For simplicity, we are not distinguishing between memories in a single node. **(b)** Execution scheme using a static qubit allocation. The scheme shows the batches and order in which the required EPs are generated; the colored paths represent the entanglement routes along which the corresponding EPs are generated. **(c)** Execution scheme using a dynamic qubit allocation. Here, we teleport q_2 to q_4's node after the fifth gate of the circuit is executed , to further minimize the number of EPs required. In particular, instead of generating the EPs e_3 and e_5 to execute the corresponding gates, we instead teleport q_2 to q_4's node (using an EP), which makes the gates corresponding to e_3 and e_5 local, and thus, saving one EP. The scheme in the figure shows the batches in which the set of desired EPs (including the EP required for teleportation) are generated. Assuming equal expected generated latency of EPs across all links, using a dynamic allocation reduces the overall execution latency.

to identify when to perform teleportations and when to perform telegates to facilitate execution of remote gates. (ii) Second, we must determine when and how to generate the EPs required to execute the remote gates.

We discuss the above steps in the following sections. For simplicity, we assume the local-gate execution latencies to be zero; we relax this assumption in §VII.

V. Step 1: Determining an Initial Qubit Allocation Function

This section describes an algorithm for determining an initial qubit-allocation function to yield an efficient execution scheme. We use the approach from [19], which we summarize here for completeness. We start with formulating the problem.

Qubit-Allocation Problem Formulation. Informally, the qubit-allocation problem essentially maps a circuit graph (defined over circuit qubits, with edge weights representing the number of gates between the qubit-pairs) onto a network-coupling graph (defined over network memories, with edge weights representing the cost of executing a gate over the qubits in the corresponding network memories); the optimization objective is to minimize an appropriately defined cost of the mapping. In effect, we want to map the circuit qubits to network memories so that qubit-pairs with more gates between them are mapped to memories that are "close by"; this can be formally captured by defining a mapping cost as the weighted sum of the product of the weights of the edge pairs that map to each other in the mapping. We formalize the above intuition below by defining the input graphs and the problem.

Circuit Graph G_C. The circuit graph is an edge-weighted graph defined over the set of qubits $Q(C)$ of a given circuit C with weight $w(q_i, q_j)$ associated with an edge (q_i, q_j) representing the number of (binary) gates in C over the qubits q_i and q_i.

Network-Coupling Graph G_N. The network-coupling graph is an edge-weighted graph defined over the set of qubit-memories $\bigcup_i Q(P_i)$ in the given quantum network with QCs $\{P_i\}$. The weight $w'(m_a, m_b)$ associated with an edge between two memories (m_a, m_b) is defined as follows.

- If m_a and m_b are in the same computer P_i, then the weight represents the time to execute the gate over qubits in m_a and m_b using swap operations.
- If $m_a \in Q(P_i)$ and $m_b \in Q(P_j)$, then the weight on the edge (m_a, m_b) is the latency of (independently) generating an EP over network nodes P_i and P_j.

For the sake of simplicity, the above weighting of the network-coupling graph assumes that each EP is generated independently. However, when we generate the EPs in the following step (Step 2 discussed in §VI), we generate them as concurrently as possible. We now formally define the qubit-allocation problem.

Qubit-Allocation Problem Formulation. Given a quantum circuit C and a quantum network, the qubit-allocation problem is to determine a mapping $\eta : Q(C) \to \bigcup_i Q(P_i)$ from the circuit qubits to the qubit-memories of the given network, such that the mapping-cost $\text{cost}(\eta)$, defined as below, is minimized.

$$\text{cost}(\eta) = \sum_{q_i, q_j \in Q(C)} w(q_i, q_j) \times w'(\eta(q_i), \eta(q_j))$$

The qubit-allocation problem can be shown to be NP-Hard by a reduction from the well-known `minimum quadratic assignment` problem (min-`QAP`) [16].

Qubit-Allocation Algorithm Based on Quadratic-Assignment. The prior work [19] poses the qubit allocation problem as an instance of the maximum-quadratic-assignment problem (max-`QAP`), and use a 4-approximation algorithm from [1] to obtain a qubit allocation. The approach from [19] can be summarized as follows.

1) Add dummy vertices to the circuit graph G_C to make it the same size as G_N. Then subtract the weights of G_C from a large constant M. Let G''_C be the resulting graph.
2) Solve the max-`QAP` over the graphs G''_C and G_N using the 4-approximation algorithm for max-`QAP`.

We use the above to determine our initial qubit allocation.

VI. Step 2: Determining the Execution Scheme

In this section, we determine the (distributed) execution scheme once the initial qubit allocation has been computed (in Step 1). As mentioned before, the execution scheme entails determining how to execute each of the "remote" gates and then generate the desired EPs. In particular, we consider two approaches: (i) `DPoverSubCircuits`, which essentially determines the execution scheme by determining the changes to the qubit allocation function over time (via teleportations), (ii) `Online-Tele`, which "scans" the given circuit and for each remote gate encountered, determines how to execute it. In both schemes, the set of EPs to be generated are then readily apparent—and are generated concurrently by batching.

A. Dynamic Programming Approach

We now describe the `DPoverSubCircuits` algorithm that uses dynamic programming to determine an optimized sequence of sub-circuits along with the "bridging" teleportations, such that the overall circuit-execution time is minimized. The high-level idea is to first find a sequence of sub-circuits with suitable qubit allocations, teleportations to transform qubit allocations across sub-circuits, and telegates to execute remote gates within each sub-circuit. Before formally presenting the dynamic program, we define a few useful terms.

Sub-Circuit. A sub-circuit of a given circuit C is any contiguous part of C and can be represented by the starting and the ending time-instant. We use the algorithm from § V to compute a suitable qubit allocation for every sub-circuit of C. Thus, each sub-circuit has an associated cost equal to the execution time of the sub-circuit, provided telegates are used to execute remote gates.

Stitching Two Sub-Circuits. Consider two (adjacent) sub-circuits C_1 and C_2, and the corresponding qubit allocation functions μ_1 and μ_2. Then, stitching C_1 to C_2, for the given qubit allocation functions μ_1 and μ_2, signifies determining and using a set of teleportations that transform μ_1 to μ_2; the stitching cost equals the expected latency of generating EPs for the necessary teleportations.

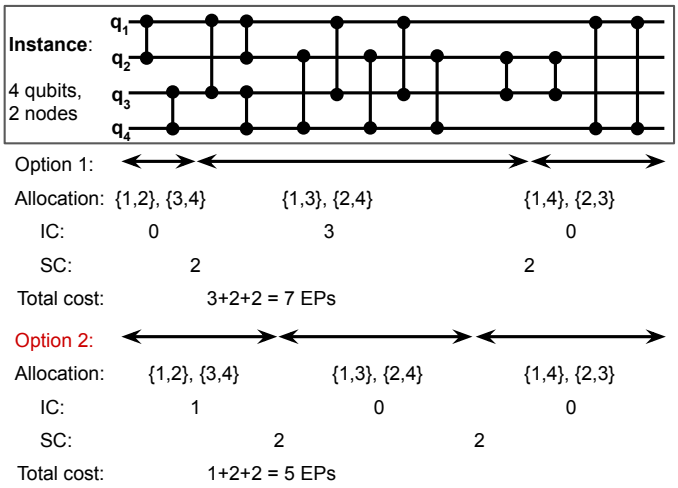

Fig. 3: `DPoverSubCircuits`: The figure shows two sequences of subcircuits and their cost estimates (number of required EPs).

Using the above concepts, we now describe the `DPoverSubCircuits` algorithm.

`DPoverSubCircuits`**:** We can divide a given circuit C into sub-circuits with minimum aggregate cost (includes sub-circuit and stitching costs) using dynamic programming as follows. Let $S[1, i, j]$ be the optimal cost for the sub-circuit $(1, j)$, formed by stitching together multiple sub-circuits within $(1, j)$ with the constraint that (i, j) is the last such sub-circuit. Our goal is to determine $\min_i S[1, i, m]$, where m is the last time instant. We determine $S[\ ]$ values recursively using dynamic programming as follows.

$$S[1, i, j] = min_{i'}\ S[1, i', i] + \text{IC}((i, j)) + \text{SC}((i', i), (i, j))$$

Above, $\text{SC}((i', i), (i, j))$ is a function that gives the cost of stitching the sub-circuit (i', i) to (i, j) for the given pre-computed qubit allocation functions. Note that the labeling of network nodes must be fixed throughout the execution of the circuit since the nodes and the links between nodes may be heterogeneous. Therefore, the set of teleportations (and EPs) required to switch between two-qubit allocations is straightforward. Additionally, $\text{IC}((i, j))$ is the internal cost of the sub-circuit (i, j) given $\mu_{(i,j)}$; the internal cost of (i, j) is the expected latency of generating the EPs required to execute the remote gates within (i, j) using telegates (see §VIII).

Motivation for Further Improvement. Note that `DPoverSubCircuits` currently computes qubit allocations for each sub-circuit independently. While this method yields low internal cost for each sub-circuit, it does not consider the stitching cost. Therefore, determining a sequence of qubit allocations that are "similar" and have low internal costs may reduce the expected latency of generating EPs. To incorporate the above insight, we modify the above `DPoverSubCircuits` algorithm as described below.

Improved `DPoverSubCircuits`**.** In the `Improved DPoverSubCircuits` algorithm, we allow the recomputation of the qubit allocation corresponding to the sub-circuit (i, j) based on the qubit allocation of the preceding sub-circuit (i', i). This modification is incorporated into the improved

stitching function SC*, as described momentarily. The improved dynamic programming formulation to determine $S[\ ]$ values recursively is as follows.

$$S[1,i,j] = min_{i'}\ S[1,i',i] + \text{IC}((i,j)) + \text{SC}^*(\mu_{(i',i)},(i,j))$$

Above, $\mu_{(i',i)}$ is the qubit-allocation function of the sub-circuit (i',i) in the solution $S[1,i',i]$. The improved stitching cost function SC* works in two steps: (i) For the input operand (i,j) sub-circuit, it computes a qubit-allocation function that "matches" as closely as possible with the input $\mu_{(i',i)}$—so that the cost of stitching the sub-circuit (i',i) to (i,j) is minimized, and (ii) it determines the teleportations required to stitch the sub-circuit (i',i) to (i,j) using the above qubit-allocation functions.

Computing $\mu_{(i,j)}$ Based on $\mu_{(i',i)}$. We wish to compute a qubit allocation $\mu_{(i,j)}$ based on $\mu_{(i',i)}$ such that the sum of the internal cost of (i,j) and the stitching cost between $\mu_{(i,j)}$ and $\mu_{(i',i)}$ is minimized. We do this by solving a generalized version of the max-QAP problem described in §V. In the generalized version of max-QAP, there are three input graphs G_1, G_2 and G_3, and the objective is to find an assignment $\mu : V(G_1) \rightarrow V(G_2)$ to maximize the following quantity: $\sum_{1\leq i,j\leq n} w_1(i,j)w_2[\mu(i),\mu(j)] + \sum_{1\leq i\leq n} w_3[i,\mu(i)]$. The graph G_3 represents the benefit of assigning a vertex in G_1 to a vertex in G_2. When the graph G_2 satisfies the triangular inequality, there is a $4-$approximation algorithm for the generalized max-QAP problem (See [1]). To construct an instance of generalized max-QAP, we first construct the graphs $G''_C(G_1)$ and $G_N(G_2)$ as described in §V. Then, we construct a bipartite graph G_3 as follows-

- G_3^A contains the vertices of G''_C and G_3^B contains the vertices of G_N.
- Add an edge of weight $L - Latency(Stitching(\mu_{(i',i)}(q), \mu_{(i,j)}(q)))$ between the vertices q (in G_3^A) and $\mu_{(i,j)}(q)$ (in G_3^B). Here, L is the highest generation latency of a single EP in the network.

We solve the above instance of max-QAP and obtain a qubit allocation function for the sub-circuit (i,j).

B. Online Approach

In this section, we present an online algorithm, Online-Tele, for determining the execution scheme.

High-level Idea. The high-level idea behind Online-Tele is to execute each remote gate encountered in the 'best' way possible. This means executing the gate in the 'best' node possible or using a telegate, depending on which option incurs the least latency in the "near future." The 'best' node is determined based on the gates in the near future shared between the qubits in the node and the operands of the remote gate currently under consideration.

Online-Tele.

1) Scan the circuit from left to right.
2) Whenever a remote gate g is encountered, do-
   a) Consider the two operands q_1, q_2 of g.

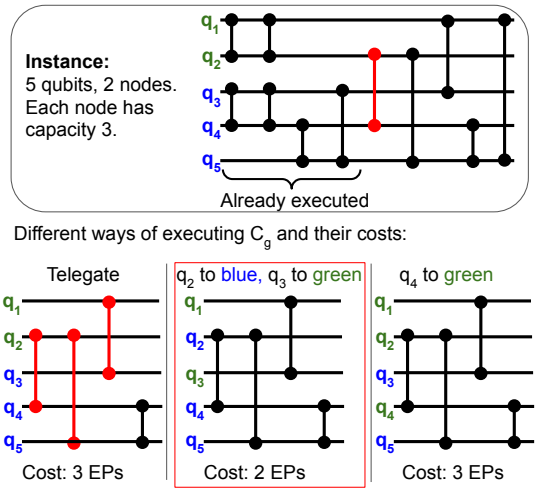

Fig. 4: Online-Tele. The figure shows different ways to compute the gate C_g (red, in the top figure) along with costs (in the number of EPs required).

   b) Construct a circuit C_g based on the next 'r' gates involving q_1 and/or q_2.
   c) For each node p, compute the expected latency t_p of executing the gates in C_g provided g is executed in p by teleporting q_1 and/or q_2.
   d) Additionally, compute the latency t of executing C_g provided g is executed using a telegate.
   e) Determine how and where to execute g such that the expected latency of executing C_g is minimized.

Best Computer to Execute g. More formally, to determine the best node for a remote gate g, we define and compute a "benefit" for each node and pick the node with the highest benefit. For a remote gate $g = (q_1, q_2, t)$, the benefit of a node p is define as the expected latency of generating the EPs necessary to execute the next r binary gates involving q_1 and/or q_2 provided q_1 and q_2 are teleported to p. Note that teleporting q_1 (and/or q_2) to p may require some qubits to be displaced in p due to storage constraints. In such cases, we select the qubit(s) in p that leads to the smallest increase in latency upon removal. We teleport this displaced qubit to the node that previously contained q_1 (or q_2).

C. Generating the Desired EPs

From the algorithms in sections VI-B and VI-A, we obtain the set of EPs $\mathcal{E}$ that must be generated. We now try to generate these EPs concurrently while satisfying decoherence constraints (which we represent, as in prior work [19], by enforcing that each EP qubit is generated and consumed with a decoherence threshold of τ). Note that when there is decoherence, EPs can decohere while and after (as they wait to be consumed) being generated. In particular, decoherence may require that we partition $\mathcal{E}$ into subsets of EPs (called *batches*) with each batch generated independently—as generation of the entire set $\mathcal{E}$ concurrently may result in some EPs having a generation latency of more than the decoherence threshold τ. We use the Static algorithms from [19] to obtain a sequence of batches of EPs, each of which can be generated concurrently without violating decoherence constraints.

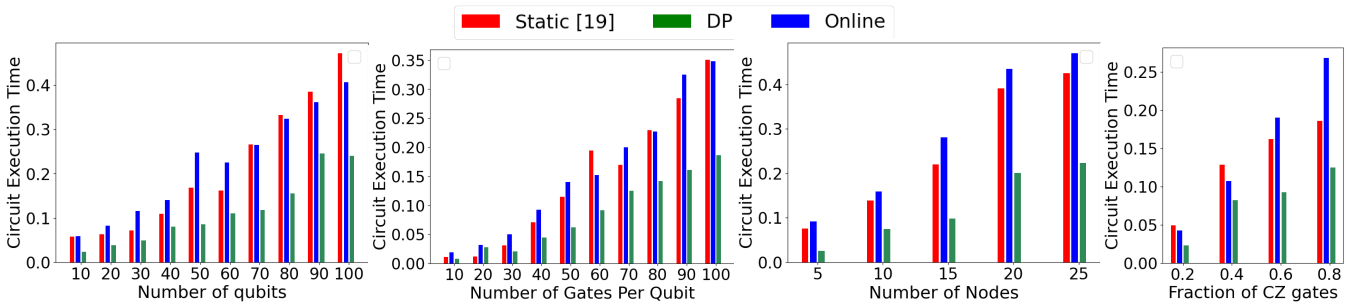

Fig. 5: Total execution time taken by various algorithms for varying parameters in *CZ* circuits.

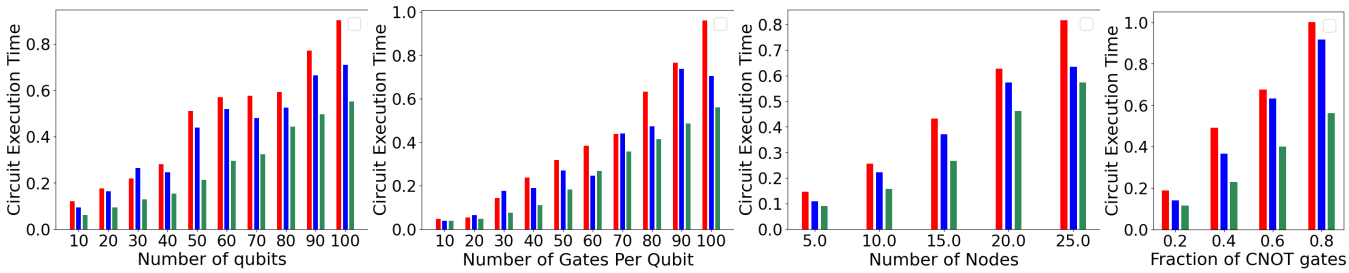

Fig. 6: Total execution time taken by various algorithms for varying parameters in $CNOT$ circuits.

VII. Generalizations

Cat-entanglements. As mentioned before, using cat-entanglements instead of telegates could result in lower circuit execution time, as a single cat-entanglement can help execute *multiple* remote gates. In the `DPoverSubCircuits` algorithm, we can use the procedure from [19] to select a set of cat-entanglements to execute gates within each sub-circuit. In the `Online-Tele` algorithm, we can modify Step 2(d) as follows. For each node p, compute the expected latency t'_p of executing C_g using cat-entanglements, provided g is executed in p by creating linked copies of q_1 and/or q_2.

Incorporating Non-zero Gate Execution Latency. Recall that we assumed, for simplicity, zero gate latencies; see §IV. Step 1 (§V) can be extended to incorporate non-zero gate latencies by modifying the network coupling graph to include the swap latency between qubit memories in the same network node. To extend Step 2, we make two changes. (i) When generating the EPs in batches, the gates and swaps are done in parallel with the generation of EPs of the *following* batch. Thus, gates and swaps do not add to the total execution time—as long as their latencies are less than the EP-batch latencies (which is expected to be largely true). (ii) We only consider batches whose total latency (including swap and gate operations) is less than the decoherence threshold τ.

Fidelity Constraints. The fidelity of EPs may degrade during the generation process due to the quantum operations (decoherence can also affect fidelity, but we have handled it separately). In our schemes, we use the purification technique to overcome fidelity degradation by requiring and creating multiple copies of EPs [6] (instead of just one) to execute each remote gate; we required the number of copies to be proportional to the entanglement path length l used to generate the EP (as fidelity degradation is somewhat proportional to l).

VIII. Evaluation

In this section, we evaluate our algorithms over NetSquid [3], a QN simulator on random and benchmark circuits. We use the NetSquid simulator to generate desired EPs.

Algorithms Compared. We compare our techniques with the greedy algorithm proposed in [19] (referred to as `Static`, here), as it performs the best in their empirical results, which compares all the known algorithms known for `DQC` to minimize circuit execution time. `Static` uses only telegates (or cat-entanglements) to execute remote operations and, hence, operates under a static qubit allocation function. Note that all approaches use the same initial qubit allocation function (from §V). We use `DP` and `Online` to refer to the algorithms that uses `DPoverSubCircuits` and `Online-Tele`, respectively, to determine the execution scheme. To compute the expected latency of generating EPs for teleportations and telegates, we use the linear programming algorithm from [10].

Generating Random and Benchmark Circuits. We evaluate the techniques on random quantum circuits using the following parameters: number of qubits (default=50), number of gates per qubit (default=50), and fraction of binary gates (default=0.5). Given the parameter values, we generate the random circuit one gate at a time. Gate operands are chosen randomly. We also evaluate on benchmark circuits corresponding to Quantum Fourier Transform (QFT), Quantum Phase Estimation (QPE), and GHZ state generation (GHZ), of various sizes obtained from the Munich Quantum Toolkit [15].

CNOT and CZ Circuits. Binary gates in the random circuits can be either all CNOT (referred to as CNOT circuits) or all CZ gates (referred to as CZ circuits). The benchmark circuits have only CNOT and unary gates. For evaluations over CNOT circuits (Figs. 6a-7c), we use the versions of the algorithms that use telegates. For evaluations over CZ circuits (Figs. 5a-5d), we use the versions of the algorithms that use cat-entanglements (as described in §VII and [19]).

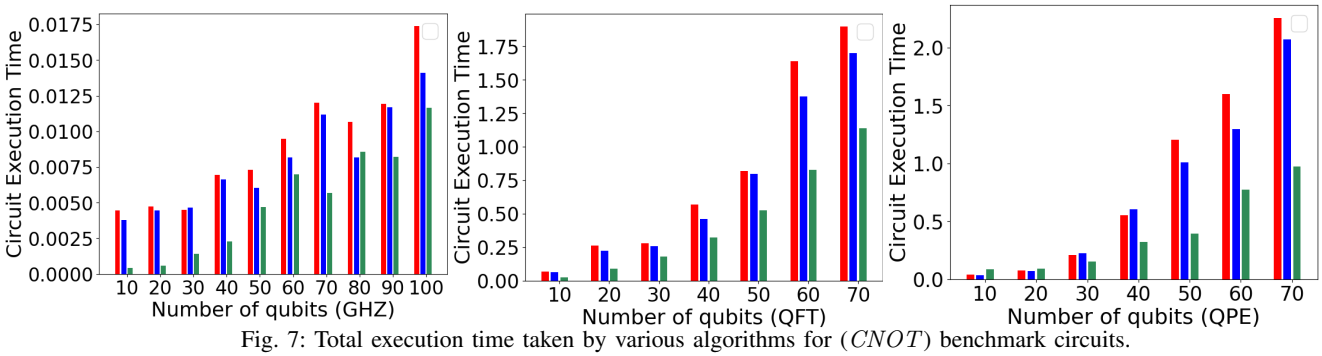

Fig. 7: Total execution time taken by various algorithms for ($CNOT$) benchmark circuits.

Generating Random Networks. We use a quantum network spread over an area of $100km \times 100km$. We use the Waxman model [20], which has been used to create Internet topologies. We vary the number of network nodes from 5 to 25, with 10 as the default. The total number of data memories in the network is equal to the number of circuit qubits, which are randomly distributed among the network nodes.

Network Parameters. We use network parameter values similar to the ones used in [10]. In particular, we set the atomic-BSM probability of success and latency to be 0.4 and 10μ seconds and the optical-BSM probability of success to be 0.3. We use atom-photon generation times and probability of success as 50μ sec and 0.33, and the decoherence threshold of 1 second.

Evaluation Results. We evaluate the algorithms over the circuits and networks as described above. We consider $CNOT$ and CZ gate-based circuits and vary one parameter at a time while keeping the other parameters fixed to their default values mentioned above. See Figs. 5-7. We observe the following:

- Among the DDQC algorithms (`DP` and `Online`), `DP` outperforms `Online` significantly. The best DDQC algorithm (`DP`) significantly outperforms the `Static` algorithm which uses a static qubit allocation—sometimes up to 50% (in CZ as well as CNOT circuits).
- In CZ circuits, `Online` sometimes performs worse than even `Static` by about 10%. This is likely due to the fact that when determining the choice of each cat-entanglement (which can help execute multiple future gates), we only look at a limited window of *r* gate which can skew the benefit estimation. In the case of telegates (which only helps execute a single gate), however, the benefit estimation is still expected to be accurate.

Runtime Overhead. Our polynomial-time schemes run in order of a few seconds for large circuits; this is a tolerable overhead, especially since these distribution strategies need to be run only once (for a given quantum algorithm).

IX. Conclusion

In this paper, we have demonstrated that "dynamic" allocation of qubits can offer a significant benefit in the efficient distribution of quantum circuits. To our knowledge, this is the first paper to use dynamic allocation to minimize circuit execution time. Our future work is focused on developing approximation algorithms for the problem or subproblems in this context. In addition, the more general problem of the distribution of multiple circuits over a quantum network is of great interest to us.

References

[1] E. M. Arkin et al. Approximating the maximum quadratic assignment problem. *Information Processing Letters*, 2001.

[2] C. H. Bennett et al. Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels. *PRL*, 1993.

[3] T. Coopmans et al. Netsquid, a discrete-event simulation platform for quantum networks. *Commun. Phys.*, 2020.

[4] D. Cuomo et al. Optimized compiler for distributed quantum computing. *ACM Trans. on Quantum Computing*, 2023.

[5] O. Daei et al. Optimized quantum circuit partitioning. *Intl. J. of Theoretical Phys.*, 2020.

[6] W Dür and H J Briegel. Entanglement purification and quantum error correction. *Reports on Progress in Phys.*, Jul 2007.

[7] J. Eisert et al. Optimal local implementation of nonlocal quantum gates. *Phys. Review A*, 2000.

[8] D. Ferrari et al. Compiler design for distributed quantum computing. *IEEE Transactions on Quantum Engineering*, 2021.

[9] R. G. Sundaram et al. Efficient distribution of quantum circuits. In *DISC*, 2021.

[10] M. Ghaderibaneh et al. Efficient quantum network communication using optimized entanglement swapping trees. *IEEE TQE*, 2022.

[11] Y. Mao et al. Probability-aware qubit-to-processor mapping in distributed quantum computing. In *QuNET*, 2023.

[12] Y. Mao et al. Qubit allocation for distributed quantum computing. In *IEEE INFOCOM*. IEEE, 2023.

[13] P. A. Martinez and C. Heunen. Automated distribution of quantum circuits via hypergraph partitioning. *Phys. Review A*, 2019.

[14] E. Nikahd et al. Automated window-based partitioning of quantum circuits. *Phys. Scripta*, 2021.

[15] N. Quetschlich et al. Mqt bench: Benchmarking software and design automation tools for quantum computing. *Quantum*, 2023.

[16] S. Sahni and T. Gonzalez. P-complete approximation problems. *JACM*, 1976.

[17] R. G. Sundaram et al. Distribution of quantum circuits over general quantum networks. In *IEEE QCE*, 2022.

[18] R. G. Sundaram et al. Distributing quantum circuits using teleportations. In *IEEE QSW*, 2023.

[19] R. G Sundaram et al. Distributed quantum computation with minimum circuit execution time over quantum networks. In *IEEE QCE*, 2024.

[20] B. M Waxman. Routing of multipoint connections. *IEEE J. on Selected Areas in Communications*, 1988.

[21] A. Yimsiriwattana and S. J. Lomonaco. Generalized ghz states and distributed quantum computing. *arXiv: Quantum Phys.*, 2004.

[22] M. Z. Moghadam et al. Optimizing teleportation cost in distributed quantum circuits. *Intl. J. of Theoretical Phys.*, 2018.