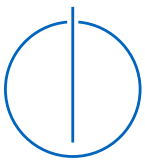
SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Games Engineering

Scalable Quantum Cloud Scheduling: Optimizing Resource Allocation for Efficient NISQ Computing

Dmitry Lugovoy



SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

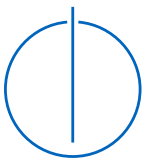
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Games Engineering

Scalable Quantum Cloud Scheduling: Optimizing Resource Allocation for Efficient NISQ Computing

Skalierbares Quantum Cloud Scheduling: Optimierung der Ressourcenzuweisung für Effizientes NISQ-Computing

Author:	Dmitry Lugovoy
Supervisor:	Prof. Pramod Bhatotia
Advisors:	Emmanouil Giortamis, Francisco Romão
Submission Date:	15.01.2024



I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 15.01.2024 Dmitry Lugovoy

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Prof. Pramod Bhatotia, for providing me with the opportunity to delve into the fascinating realm of quantum computing and write this thesis under the chair of Computer Systems.

I am deeply thankful to my advisors, Emmanouil Giortamis and Francisco Romão, whose dedication and commitment have been invaluable throughout this journey. Their organization of weekly check-in meetings, insightful feedback, encouragement, and steering of the thesis's direction have significantly contributed to its development. Their mentorship has not only refined the quality of my work but has also served as a constant source of motivation to propel me forward.

Additionally, I extend my appreciation to the entire team at the chair of Computer Systems for providing me with access to essential computational resources. The availability of these resources has been crucial in conducting the experiments and analyses necessary for this research.

Abstract

In the continually evolving domain of quantum computing, there has been a notable surge in interest, with quantum cloud providers assuming pivotal roles in facilitating widespread access to quantum capabilities. Nevertheless, the contemporary landscape of quantum cloud computing, especially within the Noisy Intermediate-Scale Quantum (NISQ) epoch, confronts formidable challenges. The manual selection of quantum computers for job execution emerges as a critical bottleneck, giving rise to issues such as uneven load distribution, suboptimal fidelity of quantum circuits, and prolonged waiting times. These challenges collectively impede the realization of the complete potential inherent in quantum computing technologies.

This thesis addresses the imperative demand for efficient task scheduling in the quantum cloud, aiming to mitigate the consequences of the current manual selection paradigm. The proposed system introduces a balanced approach to job waiting time and fidelity by employing multi-objective optimization techniques. Notably, the system enables the simultaneous scheduling of multiple jobs on multiple quantum computers, taking into account system preferences regarding the importance of different objectives. Through evaluation in an environment closely mirroring real quantum cloud conditions, the proposed system demonstrates robust scalability. To enhance its decision-making capabilities, a dataset comprising more than 7,000 job executions is collected and utilized to train a model capable of accurately predicting job execution times. The complete source code of the system, along with the collected dataset, is publicly available on GitHub - `https://github.com/Gr1dlock/ScalableQuantumCloudScheduling`.

Contents

1 Introduction

In recent years, quantum computing has captivated researchers, scientists, and enthusiasts due to its potential for tackling intractable problems that classical computers cannot. This paradigm shift stems from the principles of superposition and entanglement, enabling quantum computers to perform computations at speeds exceeding classical limits.

The significance of quantum computing extends to diverse domains such as cryptography, drug discovery, and optimization problems. At the forefront of this development are NISQ devices, characterized by an intermediate number of qubits and inherent operational noise. Despite imperfections, these devices offer insight into the transformative capabilities of quantum computation.

The accessibility of quantum computing has been facilitated by the emergence of quantum cloud services, democratizing access to quantum capabilities. This open access accelerates quantum research and introduces a diverse range of individuals to quantum computing, fostering innovation and exploration.

However, the current quantum cloud model introduces a critical challenge – users must manually select the quantum computers on which they execute their quantum programs. This manual selection process often results in suboptimal execution fidelity, uneven distribution of resources, and prolonged waiting times for job completion.

While there is ongoing research in selecting optimal Quantum Processing Units (QPUs) for individual quantum circuits, the focus has primarily been on optimizing execution time and fidelity for single circuits. This leaves a notable gap as this approach proves non-scalable in real-world scenarios with a multitude of jobs, each comprising multiple circuits.

This thesis addresses the need for scalable quantum cloud scheduling. The core issue lies in the absence of an automated, holistic approach to quantum cloud task scheduling that considers the interdependence of multiple quantum circuits within the larger context of a user's job submission. The practice of individual circuit scheduling falls short of meeting the demands of modern quantum computing scenarios with thousands of jobs.

To overcome these challenges, our proposed system takes a comprehensive approach. It accepts job submissions from users, estimates the fidelity and execution times of their circuits on all available QPUs, and employs a multi-objective optimization algorithm to

find optimal scheduling solutions. The final step involves selecting a single scheduling solution that best aligns with the preferences for the importance of different objectives.

Building upon the aforementioned context, this work makes several key contributions to address the challenges in quantum cloud scheduling:

- **Formulation of the Scheduling Problem and Multi-Objective Optimization:** In Subsection 4.4.1, we define the quantum cloud scheduling problem, which revolves around scheduling multiple jobs on various quantum computers while adhering to program size limits inherent in quantum computing. Subsection 4.4.2 presents our solution approach, leveraging a genetic algorithm to solve the scheduling problem efficiently. This algorithm aims to find optimal solutions by considering multiple objectives simultaneously, thereby addressing the inherent trade-offs in quantum cloud scheduling between execution fidelity and runtime.
- **Dataset Collection and Model Training:** In Subsection 4.3.2, we describe the collection of a dataset comprising over 7000 job executions in a quantum cloud environment. We then employ it to train a predictive model for estimating execution times, a critical component in our scheduling system.
- **Model Evaluation and Feature Analysis:** Section 6.2 details the evaluation of our trained model, including an in-depth feature analysis. By identifying the job features that most significantly impact execution time, we gain valuable insights into the factors influencing quantum cloud job performance.
- **Task-Specific Scheduling Evaluation:** In Section 6.3, we assess the scheduling system's effectiveness on individual tasks. Our findings demonstrate the system's ability to strike a balance between fidelity and waiting time.
- **End-to-End System Evaluation:** Section 6.5 provides a comprehensive end-to-end evaluation of the scheduling system. We analyze real quantum cloud load conditions, subsequently constructing a simulation resembling this load. The results affirm the system's capability to handle workloads twice as extensive as current quantum cloud conditions while effectively balancing fidelity and waiting time. This showcases substantial scalability potential in our proposed quantum cloud scheduling approach.

2 Background

This chapter offers fundamental information essential for grasping the context of this thesis. It encompasses quantum computing, scheduling problems, objective optimization, and regression models.

2.1 Quantum Computing

2.1.1 Foundations

In delving into the fundamentals of quantum computing, it's crucial to grasp the core principles that differentiate it from classical computing. Two foundational concepts, interference, and entanglement, are instrumental in harnessing the computational power of quantum systems:

- **Quantum interference**, a key principle, arises from the wave-like nature of quantum particles. It involves the constructive or destructive combination of probability amplitudes associated with different quantum states. This phenomenon allows quantum computers to process information in parallel, exponentially increasing their computational capacity compared to classical counterparts.
- **Entanglement** is a phenomenon where quantum particles become intricately correlated and share a joint quantum state. Alterations to the state of one entangled particle instantaneously influence the state of the other, even when separated by vast distances. This phenomenon provides quantum computers with a powerful mechanism for creating highly interconnected states, forming the basis for achieving quantum computational advantages.

2.1.2 Quantum Circuits

In the realm of quantum computing, quantum circuits serve as the analog to classical computer programs, facilitating the execution of quantum algorithms. To navigate the complexity of quantum computing, one must delve into the fundamental components that constitute quantum circuits: qubits, gates, and measurements.

At the heart of quantum circuits lie quantum bits, or qubits, the quantum counterparts to classical bits. Unlike classical bits that exist in states of either 0 or 1, qubits possess the unique capability to exist in a superposition of both states simultaneously. This distinctive characteristic forms the cornerstone of quantum computing, enabling the parallel processing of information and exponentially expanding the scope of computations that can be undertaken.

Quantum gates, akin to classical logic gates, manipulate qubits to perform specific operations. The quantum world introduces single-qubit gates, which operate on individual qubits, two-qubit gates, which introduce interactions between pairs of qubits, and multi-qubit gates, which allow simultaneous operations on multiple qubits. Among the plethora of quantum gates, several stand out as the most well-known:

- Hadamard Gate (H): This gate places a qubit into a superposition of the 0 and 1 states.
- Pauli-X Gate (X): Analogous to a classical NOT gate, the Pauli-X gate flips the state of a qubit from 0 to 1 and vice versa.
- CNOT Gate (Controlled-NOT): A two-qubit gate, the CNOT gate flips the target qubit's state only if the control qubit is in state 1.
- SWAP Gate: Exchanges the states of two qubits.
- Toffoli Gate: A three-qubit gate, the Toffoli gate performs a NOT operation on the target qubit only if both control qubits are in the state 1.

Quantum circuits culminate in measurements, a process that collapses the quantum superposition into classical bits. Measuring a qubit yields either a 0 or 1 result, based on the probabilities determined by the quantum state. Measurements are crucial for obtaining classical results from quantum computations, enabling the extraction of meaningful information.

2.1.3 Quantum Processing Units

The current state of QPUs is aptly termed NISQ. This term encapsulates the realization that quantum computers of this era, while surpassing classical counterparts in certain computations, grapple with inherent noise during operations. This noise originates from various sources, including gate errors and measurement errors, posing a challenge to achieving error-free quantum computations.

Achieving fault-tolerant quantum computation requires implementing error-correction codes, and constructing a logical qubit free from errors necessitates hundreds of phys-

ical qubits. However, current quantum computers are not yet equipped with the requisite number of qubits to fully mitigate errors and achieve fault tolerance.

As of the latest advancements in quantum computing, several quantum computers have scaled up to host as many as 1000 qubits. It's essential to recognize that the landscape of quantum devices is diverse, with each quantum computer possessing distinct characteristics. These characteristics encompass the number of qubits, qubit layout, gate errors, measurement errors, gate length, and various other parameters.

Differences in the number of qubits and their connectivity are crucial aspects that influence the computational capabilities of a quantum computer. The layout of qubits, often referred to as the QPU's topology, determines which qubits can interact directly with each other. This connectivity is a critical factor in designing and optimizing quantum algorithms for specific quantum processors.

Moreover, the error rates associated with quantum gates and measurements, vary among different quantum computers. These error rates play a significant role in the reliability and accuracy of quantum computations. Additionally, characteristics like gate length, which represents the duration of quantum operations, contribute to the overall performance of quantum processors.

It's important to note that the characteristics of quantum backends are subject to recalibration. Quantum computers are dynamic systems that undergo frequent recalibrations to address variations in qubit properties, environmental conditions, and other factors. As a result, the characteristics of a quantum backend, including error rates and gate durations, can change significantly over time.

2.1.4 Execution Workflow

Similar to the compilation process in classical computing, the preparation of a quantum circuit for execution on a specific QPU involves a step known as transpilation. Transpilation in the context of quantum computing refers to the transformation of a quantum circuit written in a high-level language into a form compatible with the architecture and constraints of a particular quantum device.

The need for transpilation arises due to the inherent differences in the architectures of various quantum processors. Different quantum devices have distinct qubit layouts, connectivity constraints, and gate sets. Therefore, a quantum circuit designed for one quantum processor may not be directly executable on another. Transpilation serves as an intermediary step, adapting the circuit to the specific characteristics and constraints of the target quantum hardware.

The transpilation process involves:

1. **Qubit Remapping:** Adjusting the mapping of logical qubits in the original circuit to the physical qubits available on the target quantum device. This is crucial

because the connectivity between qubits may vary between different quantum processors.

2. **Gate Decomposition:** Breaking down complex quantum gates into a sequence of simpler gates that are natively supported by the target quantum device. Quantum devices may have limitations on the types of gates they can directly implement, and gate decomposition ensures compatibility.

3. **Optimization:** Streamlining the circuit to enhance its efficiency and reduce the overall quantum gate count. This step aims to minimize the impact of noise and errors during execution.

Transpilation is a critical component of the quantum programming workflow as it enables quantum algorithms designed in a hardware-independent manner to be executed on specific quantum processors. This flexibility allows researchers and developers to write quantum algorithms at a higher level of abstraction, focusing on the algorithmic logic without being entangled with the specifics of individual quantum hardware.

Following transpilation, the quantum circuit is ready for execution on a QPU. Yet, owing to the inherent noise in quantum computations, a single run is insufficient to yield meaningful results. This is why quantum circuits undergo thousands of executions, or shots, as this practice aids in mitigating the probabilistic nature of quantum measurements, enabling researchers to extract patterns and trends from the aggregated outcomes.

A key metric in evaluating the performance of quantum circuits is the concept of execution fidelity. Execution fidelity quantifies how closely the actual outcomes of a quantum computation align with the expected ideal outcomes in the absence of noise. It provides a measure of the reliability and accuracy of the quantum computation.

The execution fidelity is calculated using the following formula:

$$F = \frac{\text{Number of Correct Outcomes}}{\text{Total Number of Shots}} \tag{2.1}$$

In this formula, the "Number of Correct Outcomes" represents the instances where the observed results match the expected or ideal outcomes, and the "Total Number of Shots" denotes the overall number of times the quantum circuit is executed.

The best possible value for execution fidelity is 1, indicating a perfect match between the actual and expected outcomes. However, achieving a fidelity of 1 is challenging in practical scenarios due to the presence of noise and imperfections in quantum devices.

2.1.5 Quantum Cloud

In the present landscape, major cloud service providers, including IBM, Microsoft Azure, Google Cloud, and Amazon Web Services, have integrated quantum computing capabilities into their platforms. This integration democratizes access to quantum computing, allowing users to leverage it for their applications.

The current access model to quantum resources follows a user-centric approach. Users submit their quantum programs to the cloud service provider's quantum computing service, with the flexibility to choose the specific quantum computer for their computations. Subsequently, the quantum program undergoes transpilation to adapt to the architecture of the selected quantum processor. The program is then executed on the chosen quantum computer, and users are billed based on a pay-as-you-go model, reflecting the actual quantum resource usage time.

The figure below illustrates this access model:

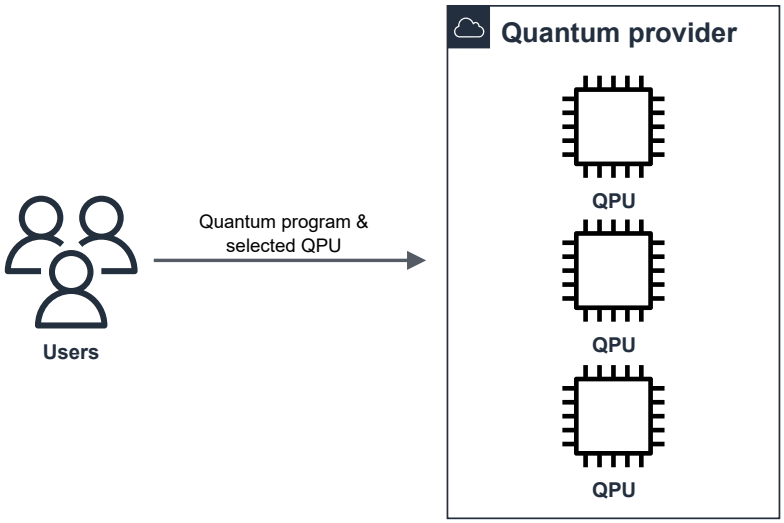


Figure 2.1: Quantum Cloud Access Model

This model eliminates the necessity for on-site infrastructure, allowing users to harness quantum computing capabilities seamlessly. Moreover, in the rapidly evolving field of quantum computing, this approach ensures that users have access to the latest quantum devices, thereby aiding the research process.

2.2 Scheduling

2.2.1 Definition and Solved Problems

Scheduling in classical computing refers to the strategic allocation of resources and management of tasks within a computer system to optimize performance and ensure efficient utilization of available resources. The history of scheduling is deeply intertwined with the evolution of computing systems.

In the early days of computing, particularly in mainframes, scheduling primarily revolved around batch processing. Mainframe computers executed a sequence of jobs submitted in batches, and the challenge lay in determining the most effective order in which to execute these jobs to minimize idle time and maximize overall system throughput.

As Operating Systems (OSs) emerged, scheduling algorithms became their integral components. Early OS scheduling focused on time-sharing systems, where multiple users could interact with the computer concurrently. Time-sharing scheduling aimed to provide fair and equitable access to system resources, enhancing user experience.

Scheduling in classical computing addresses several critical problems:

1. **Resource Utilization:** Efficiently allocating resources such as Central Processing Unit (CPU) time, memory, and storage to tasks to maximize overall system utilization.

2. **Throughput:** Ensuring a high rate of task completion to enhance the overall system throughput.

3. **Fairness:** Providing fair access to resources for all tasks or users, preventing resource starvation or monopolization.

4. **Response Time:** Minimizing the time it takes for tasks to receive CPU time, particularly crucial for interactive applications.

5. **Load Balancing:** Distributing tasks evenly across resources to prevent bottlenecks and optimize performance.

The advent of cloud computing marked a paradigm shift, introducing a new dimension to scheduling challenges. Cloud environments involve dynamic resource allocation, virtualization, and the need to meet Service Level Agreements (SLAs) for diverse users and applications. In this context, scheduling plays a key role in orchestrating the allocation of virtual machines, containers, and other cloud resources to meet varying workloads and demands.

2.2.2 Problem Taxonomy

Drawing upon the classification proposed in [9], two major categories emerge: Resource Allocation and Task Scheduling.

- **Resource allocation** in cloud computing is paramount due to the inherent elasticity of cloud environments. This category involves decisions about when to utilize existing resources and when to deploy new ones. The goal is to efficiently manage the dynamic scaling capabilities of the cloud, ensuring that the right amount of resources is available to meet varying workloads.
- **Task scheduling**, the focus of this thesis, revolves around the assignment of tasks to different existing resources and determining their order of execution. Unlike resource allocation, which deals with provisioning and deprovisioning, task scheduling is concerned with optimizing the execution of tasks on the available resources.

Within the realm of task scheduling, two main subcategories are single-objective optimization and multi-objective optimization.

- **Single-objective optimization** involves optimizing a specific criterion to enhance the overall performance of the system. For example, a scheduling algorithm might aim to minimize the makespan — the total time taken to complete a set of tasks.
- **Multi-objective optimization** extends the scope by considering multiple conflicting objectives simultaneously. This involves finding a balance between competing goals, such as minimizing execution time and maximizing resource utilization. In the context of cloud computing, a multi-objective algorithm might optimize for both task completion time and energy efficiency, achieving a trade-off between these objectives.

The objectives at play often exhibit conflicting interests, making it challenging to identify a singular optimal solution. This complexity arises because improvements in one objective may come at the expense of another. To tackle this issue, the concept of Pareto dominance and the notion of a Pareto front emerge as foundational principles.

In multi-objective optimization problems, each potential solution is evaluated based on multiple objectives. Mathematically, let X represent a solution space, $f_i(x)$ denote the value of the i-th objective function for solution x, and m denote the number of objectives. A solution x_1 is said to dominate another solution x_2 (denoted as $x_1 \preceq x_2$) if, and only if, $f_i(x_1) \leq f_i(x_2)$ for all i in the range from 1 to m, and $f_j(x_1) < f_j(x_2)$ for at

least one j in the same range. This relationship signifies that x_1 is at least as good as x_2 in all objectives and strictly better in at least one.

The set of non-dominated solutions forms what is known as the Pareto front. Mathematically, for a given set X, the Pareto front PF is defined as:

$$PF = \{x \in X \mid \nexists x' \in X : x' \preceq x\} \tag{2.2}$$

In essence, the Pareto front comprises solutions that are not dominated by any other solutions in the set, highlighting a spectrum of trade-offs between conflicting objectives.

In the context of multi-objective optimization for task scheduling, the Pareto front becomes a collection of solutions that represent optimal trade-offs between, for instance, minimizing task completion time and maximizing resource utilization. The main benefit of utilizing the Pareto front lies in providing decision-makers with a range of solutions, allowing them to explore and select the most suitable compromise based on their specific priorities and preferences.

The realm of multi-objective scheduling encompasses diverse scheduling tasks, and [8] identifies four major categories: Single Machine, Parallel Machines, Flow Shop, and Job Shop. Each category represents a distinct scheduling context with unique characteristics and challenges.

1. **Single Machine Scheduling:** In the single machine scheduling category, the challenge lies in optimizing the sequence in which a set of tasks is processed on a single machine. Objectives typically include minimizing makespan and reducing the overall flow time.

2. **Parallel Machines Scheduling:** Parallel machines scheduling involves scenarios where multiple machines are available for task execution simultaneously. Each machine can process a task independently, providing opportunities for parallelization. The optimization objectives include minimizing makespan, reducing machine idle time, or balancing the load across machines.

3. **Flow Shop Scheduling:** In the flow shop scheduling context, tasks move through a sequence of machines, with each machine dedicated to a specific operation. The challenge is to find the optimal order for processing tasks through these machines. Common objectives include minimizing makespan and achieving a balanced flow of tasks through the system.

4. **Job Shop Scheduling:** Job shop scheduling introduces a more flexible scenario, where tasks can be processed on a variety of machines in any order. The challenge is to determine the optimal sequence for executing tasks on machines, considering their specific requirements and constraints. Objectives often include minimizing makespan, reducing idle times, and balancing workloads.

In the context of quantum cloud scheduling, the optimization focus revolves around minimizing waiting times for jobs and maximizing execution fidelity. These objectives can at times be conflicting, posing a challenge in achieving an optimal trade-off. Given that quantum providers usually offer more than one QPUs and since all of them are identical in function, this scheduling task aligns with the domain of parallel machines scheduling.

2.3 Genetic Algorithms

2.3.1 Concept

Genetic Algoridthms (GAs) stand out as a prominent class of algorithms widely employed in the area of multi-objective optimization. Rooted in the principles of natural selection and genetics, GAs draw inspiration from the process of evolution to iteratively search for optimal solutions within a given solution space.

The fundamental idea behind genetic algorithms is to emulate the mechanics of natural selection and the survival of the fittest. Solutions to optimization problems are represented as individuals within a population. Each individual, akin to a potential solution, is characterized by a set of parameters or features encoding its genetic makeup. These parameters, often referred to as chromosomes or genes, represent the potential solutions' characteristics. Over successive iterations, the algorithm converges towards solutions that represent a Pareto front in multi-objective optimization.

2.3.2 Process

The underlying process of a Genetic Algorithm can be split into a series of iterative steps:

1. **Initialization:** The process commences with the creation of an initial population of potential solutions, often represented as chromosomes. Each chromosome corresponds to a candidate solution to the optimization problem, with its genes encoding the values of different variables or parameters.

2. **Evaluation:** The fitness of each individual in the population is assessed based on its performance in achieving the specified objectives. In the context of multi-objective optimization, the fitness of a solution is evaluated with respect to multiple conflicting objectives. The goal is to construct a diverse set of solutions that collectively represent a broad spectrum of trade-offs.

3. **Selection:** Individuals are selected from the current population to serve as parents for the next generation. The probability of selection is typically proportional to

the fitness of each individual, favoring solutions that perform better in terms of the objectives.

4. **Crossover:** The selected parents undergo crossover, during which segments of genetic information from two parent solutions are exchanged to create new offspring solutions. This mimics the concept of genetic recombination and introduces diversity into the population.

5. **Mutation:** A subset of the newly created offspring solutions undergoes mutation, where random changes are introduced to their genetic information. Mutation introduces exploration into the algorithm, allowing for the discovery of novel solutions that may not be accessible through crossover alone.

6. **Replacement:** The new offspring solutions, along with a subset of the current population, form the next generation. The selection of individuals for the next generation is often influenced by their fitness, favoring solutions that exhibit better performance with respect to the optimization objectives.

7. **Termination:** The algorithm iteratively proceeds through the steps of selection, crossover, mutation, and replacement until a termination criterion is met. This criterion could be a predetermined number of generations, convergence of solutions, or the attainment of a satisfactory set of Pareto-optimal solutions.

One of the advantages of GAs lies in their adaptability, as they can be tailored to various problem types by adjusting the genetic operators. These operators, including crossover and mutation, can be modified to suit specific characteristics of optimization problems. For instance, in problems involving integer or binary variables, the choice and configuration of genetic operators can significantly impact the algorithm's performance. This flexibility makes GAs a versatile and customizable tool for addressing diverse multi-objective optimization problems.

2.3.3 NSGA

The Non-dominated Sorting Genetic Algorithm (NSGA) represents an extension of classical GA specifically designed for multi-objective optimization. NSGA introduces a novel sorting mechanism based on non-domination, which categorizes solutions into multiple fronts, each representing a different level of dominance. This enables NSGA to efficiently explore the Pareto front, preserving diversity among non-dominated solutions.

In NSGA, selection is performed based on the non-dominated ranks and crowding distances, promoting a balance between exploration and exploitation. The algorithm

incorporates elitism to ensure the survival of the best solutions from one generation to the next. NSGA excels in handling problems with a large number of objectives and offers an effective approach to multi-objective optimization.

NSGA-II is a refinement and enhancement of NSGA, introducing modifications to further enhance its performance. One notable improvement is the introduction of a fast non-dominated sorting algorithm, reducing the computational complexity and enhancing the algorithm's efficiency.

NSGA-II also incorporates a crowding distance assignment mechanism, promoting a well-distributed Pareto front. Moreover, NSGA-II utilizes a binary tournament selection scheme, contributing to a more robust selection process. These modifications collectively enhance the convergence and diversity maintenance capabilities of NSGA-II, making it one of the most adopted algorithms in the realm of multi-objective optimization.

2.4 Regression Analysis

2.4.1 Definition

Regression analysis is a statistical technique employed to model the relationship between a dependent variable and one or more independent variables. The fundamental goal of regression is to understand and quantify the impact of independent variables on the variation in the dependent variable. It provides a systematic framework for exploring patterns, making predictions, and uncovering associations within datasets.

In regression analysis, the dependent variable is considered to be influenced by the independent variables through a mathematical model. The regression model aims to capture the underlying patterns and trends in the data, facilitating the prediction of the dependent variable's values based on known values of the independent variables.

Regression analysis is broadly categorized into two main types: simple regression and multiple regression. Simple regression involves one independent variable, while multiple regression deals with two or more independent variables. The process typically includes estimating the coefficients of the regression equation, assessing the model's goodness of fit, and making predictions based on the established relationships.

2.4.2 Regression Models

Regression analysis encompasses various models, each tailored to capture different patterns and relationships within the data. Here are five commonly used regression models, each with its unique characteristics:

1. **Linear Regression:** Linear Regression is one of the simplest and widely used regression models. It assumes a linear relationship between the dependent

variable Y and one or more independent variables X. The equation for simple linear regression is:

$$Y = \beta_0 + \beta_1 X + \epsilon \tag{2.3}$$

where β_0 is the intercept, β_1 is the slope coefficient, X is the independent variable, and ϵ represents the error term. In multiple linear regression, the equation extends to include multiple independent variables.

2. **Polynomial Regression:** Polynomial Regression extends the linear model by allowing for polynomial relationships between the dependent and independent variables. The equation takes the form:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \ldots + \beta_n X^n + \epsilon \tag{2.4}$$

Here, n denotes the degree of the polynomial. Polynomial Regression is particularly useful when the relationship between variables is nonlinear.

3. **Decision Trees:** Decision Trees are a non-linear regression model that recursively splits the data into subsets based on the values of the independent variables. Each split is determined by a decision rule, leading to a tree-like structure. The prediction for a given observation is the average of the dependent variable values in the corresponding leaf node.

4. **Ridge Regression:** Ridge Regression is a regularized linear regression model designed to handle multicollinearity (high correlation among independent variables). It introduces a penalty term to the linear regression objective function, aiming to shrink the coefficients:

$$\min_{\beta} \left(\sum_{i=1}^{n} (Y_i - \beta_0 - \sum_{j=1}^{p} \beta_j X_{ij})^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right) \tag{2.5}$$

Here, p represents the number of independent variables, and λ controls the strength of the penalty.

5. **Support Vector Regression:** Support Vector Regression is a regression algorithm based on Support Vector Machines. It aims to find a hyperplane that best represents the data while allowing for a margin of error. The model seeks to minimize deviations from the hyperplane, incorporating a regularization parameter for fine-tuning.

2.4.3 Ensemble Methods

Ensemble methods are machine learning techniques that combine the predictions of multiple individual models to create a more robust and accurate predictive model. The underlying idea is that by aggregating the opinions of diverse models, the overall performance can often surpass that of any individual model. Ensemble methods are effective in reducing overfitting, improving generalization, and enhancing the overall stability of the predictive model.

Several prominent ensemble methods include:

1. **Boostrap Aggregating (Bagging):** Bagging involves training multiple instances of the same base model on different subsets of the training data. The subsets are generated through bootstrapped sampling, where each instance is constructed by random sampling with replacement from the original dataset. The final prediction is typically an average or voting mechanism over the predictions of individual models. One representative of these algorithms is the Random Forest algorithm, which employs Bagging with decision trees as the base models.

2. **Boosting:** Boosting is an ensemble method that focuses on sequentially improving the performance of weak learners by assigning higher weights to misclassified instances. It trains a series of weak models, and each subsequent model corrects the errors made by the preceding ones. Adaptive Boosting (AdaBoost) is a well-known boosting algorithm commonly used with decision trees as weak learners.

3. **Stacking:** Stacking, also known as stacked generalization, combines the predictions of multiple diverse models using a meta-model. The base models are trained independently, and their predictions become input features for the meta-model. The meta-model then learns to combine these predictions for the final output. This approach leverages the strengths of each base model.

3 Overview

This chapter provides an overview of the proposed quantum scheduling system. It outlines the distinctive challenges inherent in quantum scheduling compared to classical scheduling paradigms. The chapter articulates the design goals guiding the development of the system and elucidates the systematic workflow that governs its operation.

3.1 System Overview

The proposed quantum computing scheduling system is illustrated in the figure below:

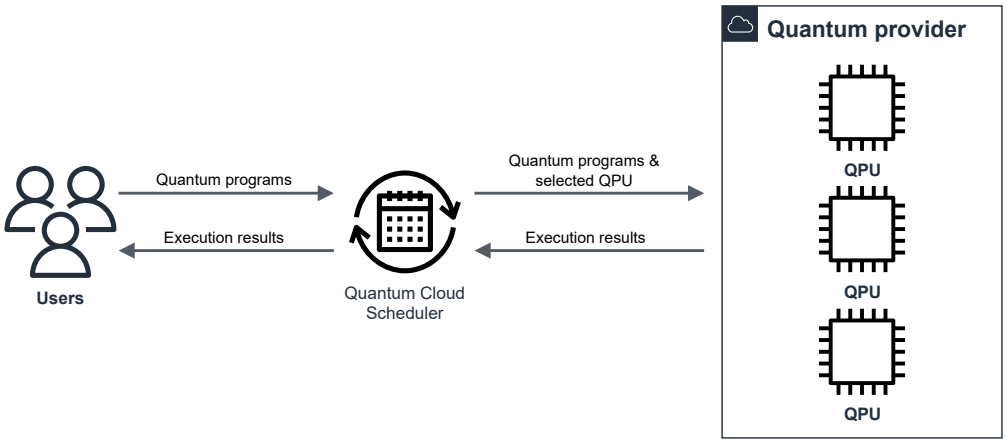


Figure 3.1: System Overview

This figure draws parallels with the access model presented in the Figure 2.1, outlining the interaction between users and cloud providers. However, a distinctive feature is the integration of our scheduling system, situated between users and quantum providers. The system serves as an intermediary, receiving quantum jobs from users, internally optimizing the selection of suitable backends for execution, and then forwarding these optimized job assignments to the quantum provider's quantum backends.

Following execution, the system communicates the results from the quantum provider back to the users.

Ideally, we envision the seamless integration of our system within the cloud provider's infrastructure. This integration would enhance the overall efficiency of quantum computing scheduling, offering users an optimized and streamlined experience while contributing to the advancement of quantum computing capabilities in the quantum cloud environment.

3.2 Challenges

The landscape of quantum computing scheduling introduces distinct challenges compared to classical scheduling paradigms, necessitating tailored solutions to address the intricacies of quantum systems. Three primary challenges stand out:

1. **Spatial Heterogeneity:** QPUs exhibit spatial heterogeneity, wherein the same quantum job may encounter varying waiting times and fidelities when executed on different QPUs. This spatial variability poses a unique challenge, demanding scheduling mechanisms that account for and adapt to the diverse characteristics of distinct QPUs within the quantum cloud environment.

2. **Temporal Heterogeneity:** Temporal heterogeneity stems from the dynamic nature of quantum backends. Recalibrations, updates, and other temporal variations can lead to fluctuations in waiting times and fidelities for the same quantum job executed on the same QPU across different time intervals. Effectively addressing temporal heterogeneity requires scheduling algorithms that dynamically adapt to the evolving conditions of quantum backends.

3. **Scalability:** The inherent scalability challenge arises with the increasing number of qubits and gates within a quantum circuit, compounded by the rising number of circuits within a job. As the complexity of quantum computations grows, the fidelity of executions tends to diminish, leading to suboptimal results. Simultaneously, the waiting time for job execution experiences significant escalation, underscoring the need for robust scheduling strategies to account for these detrimental effects.

3.3 Design Goals

While designing the system, we have formulated the following goals to guide the development process:

1. **Many-to-Many Scheduling:** The system aims to facilitate many-to-many scheduling, simultaneously allowing multiple quantum jobs to be scheduled across various quantum backends. This approach optimizes not only for individual users but also for the overall system efficiency. By balancing the load and strategically assigning jobs, the system maximizes resource utilization, contributing to a more efficient quantum cloud environment.

2. **Optimization for Conflicting Objectives:** Recognizing the inherent trade-off between fidelity and waiting time in quantum computing, the system aims to optimize for conflicting objectives. Striking a balance between achieving high-fidelity quantum computations and minimizing waiting times is crucial. The goal is to employ intelligent scheduling strategies that navigate this trade-off, delivering a reliable and efficient quantum computing experience.

3. **Execution Time Estimation:** Accurate estimation of the execution time for quantum jobs is crucial for effective task scheduling. The system incorporates robust algorithms for estimating the execution time of quantum circuits on different backends. Reliable predictions of job durations aid the system in strategic decision-making when selecting quantum backends and can also be beneficial for users by gaining insights into the expected duration of their computations.

4. **Customizable Objective Priorities:** Recognizing the dynamic nature of the quantum computing requirements, the system supports the customization of objective priorities. The system should have the flexibility to assign different priorities to fidelity and waiting time based on the current needs. This customization ensures adaptability and aligns the system's optimization strategies with the evolving priorities of the quantum provider.

5. **Scalability with the Growing Quantum Cloud:** Recognizing the rapid growth of the quantum computing field and the expanding user base, the system prioritizes scalability. It is designed to efficiently handle an increasing number of quantum jobs and QPUs. This scalability ensures that the system can seamlessly accommodate the evolving landscape of quantum cloud computing, meeting the demands of a growing user community and an expanding quantum infrastructure.

3.4 System Workflow

The system workflow encompasses a series of interconnected stages, orchestrated to efficiently schedule quantum jobs on available QPUs. The following figure illustrates the flow of the system's operations:

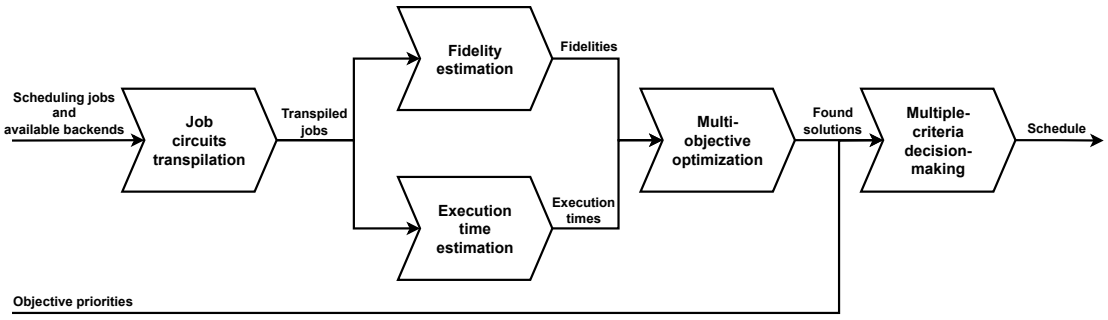


Figure 3.2: System Workflow

The process commences with the input of scheduling jobs and available backends. These inputs enter the "Job Circuit Transpilation" stage, where every circuit in each job undergoes transpilation for every available backend. This stage ensures compatibility between quantum jobs and diverse QPUs.

Subsequently, the transpiled jobs proceed to the "Fidelity Estimation" and "Execution Time Estimation" blocks. Here, each job-to-QPU pair receives fidelity and execution time estimates, crucial metrics for optimizing quantum scheduling.

The system then enters the "Multi-Objective Optimization" block, employing an optimization algorithm to identify solutions on the Pareto front. This block navigates the conflicting objectives, such as fidelity and waiting time, producing a range of Pareto-optimal solutions.

Following optimization, the Multiple-Criteria Decision Making block comes into play, utilizing the currently specified objective priorities. In this stage, the system utilizes the specified priorities to select a single solution that best aligns with the provided preferences.

Ultimately, the output of the system is the finalized quantum computing schedule, comprising optimal assignments of jobs to specific QPUs.

4 Design

This section delves into the intricacies of the quantum computing scheduling system, offering a detailed exploration of each stage illustrated in the Figure 3.2

4.1 Job Transpilation

The initial stage in our quantum cloud scheduling system, Job Transpilation, takes the scheduling jobs and currently available QPUs as input. The primary objective is to prepare quantum jobs for execution by adapting them to the specific characteristics of each QPU.

To streamline the transpilation process, the system first filters out any jobs that cannot be accommodated by the available QPUs. This involves identifying jobs with quantum circuits exceeding the size constraints of all currently accessible QPUs. By eliminating such impractical scenarios early on, the system avoids unnecessary transpilation attempts that would inevitably result in failure.

The Job Transpilation stage offers two distinct transpilation modes:

1. **Transpilation of Each Job to Each QPU:** In this mode, the system iterates over each job, selecting QPUs that can collectively accommodate all the quantum circuits within the job. Subsequently, for each identified QPU, the system transpiles all the quantum circuits present in the job. This approach ensures a tailored adaptation of each job to the specific capabilities of the chosen QPU.

2. **Transpilation of Each Job to Each Processor Type:** Alternatively, this mode explores transpiling each job for each distinct processor type among the available QPUs. Processor types signify QPUs with similar topologies and gate sets. While potentially saving computational time by avoiding redundant transpilations, this approach comes with a trade-off. The transpiled jobs do not consider variations in error properties among QPUs of the same processor type, potentially impacting job fidelity.

Regardless of the chosen transpilation mode, the culmination of this stage results in all jobs being transpiled for all suitable backends. This prepares the quantum jobs for subsequent stages, ensuring compatibility with the diverse characteristics of the available QPUs.

4.2 Fidelity Estimation

Following the transpilation stage, Fidelity Estimation plays a pivotal role in assessing the anticipated execution fidelity of quantum circuits on the selected QPUs. Recognizing that true fidelity can only be definitively calculated post-execution, as was described in Equation 2.1.4, our algorithm employs a forward-looking approach, leveraging available data from the backend to estimate execution fidelity before the actual execution takes place.

The fidelity estimation algorithm traverses each gate and measurement operation in the transpiled circuit. For each operation, the algorithm identifies the physical qubits affected by the operation and extracts the execution error associated with these qubits and the specific operation type from the quantum backend properties. The execution fidelity of an operation $F_{\text{operation}}$ is then calculated as

$$F_{\text{operation}} = 1 - \epsilon_{\text{qubits, operation}} \tag{4.1}$$

where $\epsilon_{qubits,operation}$ is an execution error of the operation on qubits.

For the entire transpiled circuit, the execution fidelity F_{circuit} is determined by taking the product of individual operation fidelities:

$$F_{\text{circuit}} = \prod_{\text{operations}} F_{\text{operation}} \tag{4.2}$$

To obtain the execution fidelity of the entire job F_{job}, the arithmetic mean of the execution fidelities of all circuits within the job is calculated:

$$F_{\text{job}} = \frac{\sum_{\text{circuits}} F_{\text{circuit}}}{\text{Number of circuits}} \tag{4.3}$$

As a result of the Fidelity Estimation stage, the system generates execution fidelity scores for all job-to-QPU combinations. These scores serve as indicators of the anticipated reliability of quantum computations on specific QPUs, guiding subsequent stages in the scheduling process.

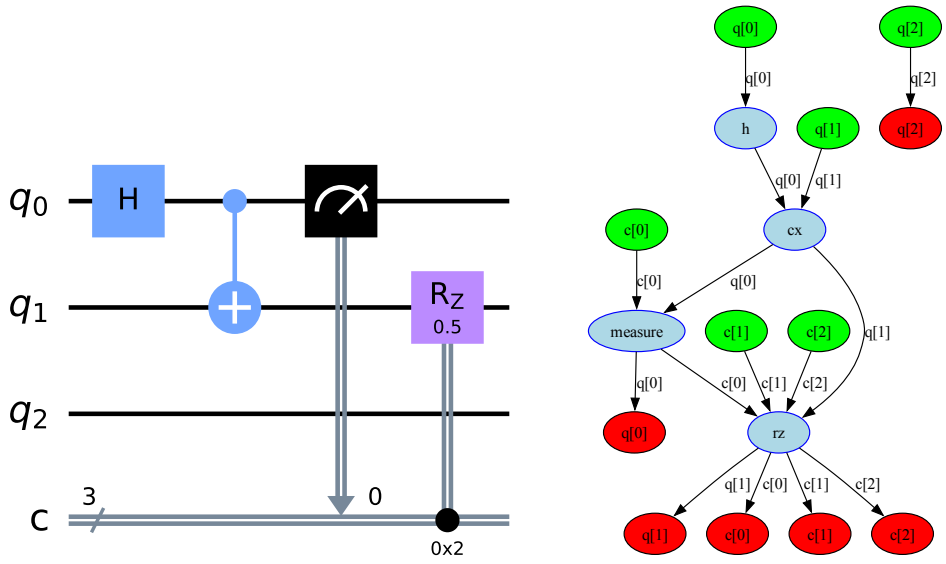
4.3 Execution Time Estimation

In the Execution Time Estimation stage, the focus shifts towards determining the approximate execution time for each transpiled job on every available QPU. Accurate execution time estimates are fundamental for subsequent optimization processes within the scheduling system. To address this requirement, two distinct approaches for execution time estimation are presented, each leveraging unique methodologies. The first

approach relies on graph traversal and utilizes available backend properties, while the second approach employs a regression model, incorporating various features extracted from the transpiled job. These estimation techniques prepare the foundation for the optimization algorithm to select efficient job-to-QPU assignments in the subsequent stages of the scheduling process.

4.3.1 Graph Traversal

The Graph Traversal method is grounded in the concept that each quantum circuit can be effectively represented as a Directed Acyclic Graph (DAG). In this representation, each node of the DAG corresponds to a quantum operation, be it a gate or measurement, while edges symbolize the data flow (qubits and classical bits) between operations. The structure of the circuit DAG begins with input nodes, proceeds with operation nodes representing gates or measurements, and concludes with output nodes. An example of a quantum circuit (a) and its graph representation (b) can be found in the figure below:



(a) Circuit (b) DAG

Figure 4.1: Quantum Circuit and Graph Representation

A fundamental metric in quantum circuit analysis is the circuit depth, which refers to the length of the longest path from input to output nodes in the DAG. In essence, circuit depth quantifies the temporal extent of quantum computations within a circuit.

The Graph Traversal method transforms each quantum circuit in a job into its corresponding DAG and iterates through its layers. Each layer of the DAG encapsulates gates that act on disjoint qubits, constituting a subcircuit with a depth of 1. The total number of layers mirrors the circuit depth, providing a hierarchical view of quantum operations.

For each layer of the DAG, the algorithm iterates through its operation nodes, extracting the durations for each operation and its associated qubits from the backend properties. The maximum duration across all operations within the layer is then considered the duration of the entire layer. This assumption is made feasible by the disjoint nature of operations within a layer, allowing them to be executed in parallel.

The algorithm proceeds by summing up the durations of all layers, providing an estimation of the execution time for the quantum circuit. To determine the execution time of the entire quantum job, the algorithm sums up the execution times of all individual circuits within the job.

4.3.2 Regression Model

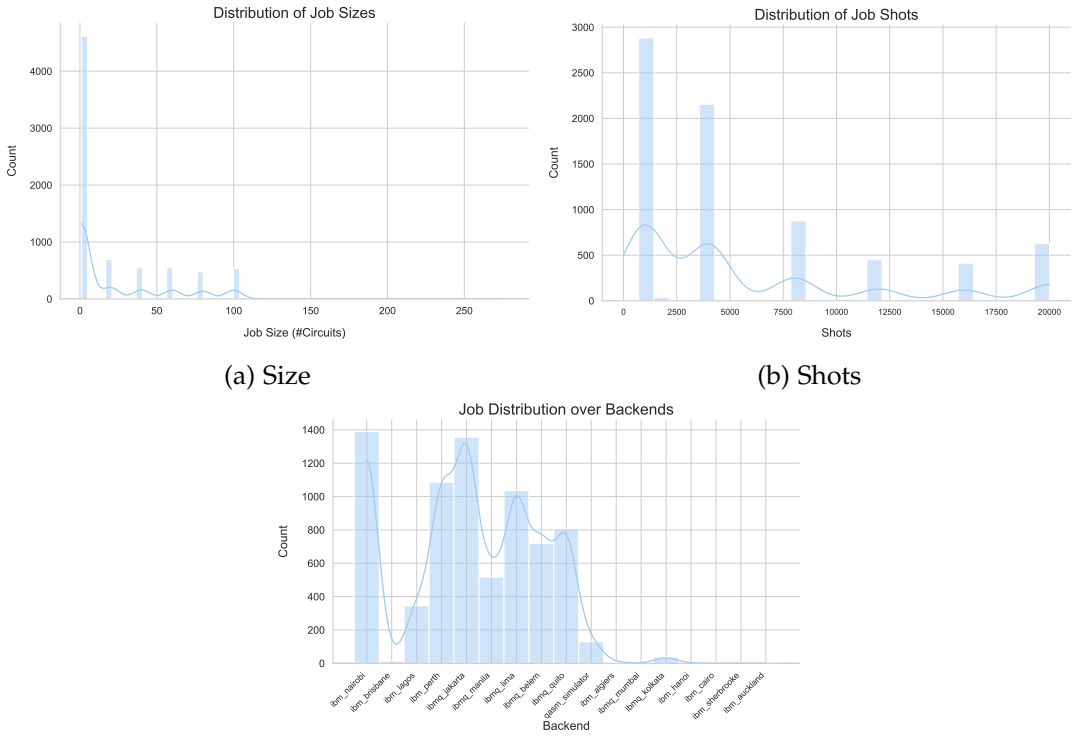
The foundation of this approach lies in a comprehensive dataset resulting from our experimentation within the quantum cloud. With over 7,000 job executions across diverse quantum backends, encompassing varying shot counts and the number of circuits within each job, our dataset serves as a starting point for developing regression models. The Figure 4.3.2 offers insights into the distribution of key parameters within the dataset.

Given that we can retrieve detailed information post-execution, including the precise execution time, regression analysis becomes a potent tool for extrapolating insights and patterns within the dataset. However, raw circuits and their execution times alone do not suffice to train a regression model. The model necessitates the transformation of raw data into meaningful features.

Beginning with fundamental features, we incorporate key aspects of a job, such as the number of circuits within the job, the number of shots, and the circuit depth. These features serve as initial contributors to the model's understanding of the dataset.

Building on these initial features, we enrich our feature set by incorporating advanced metrics, drawing insights from [17] and [23]. The first paper recommends expanding the feature set by including gate counts for all standard gates specified in the QASM 3.0 specification, emphasizing the potential benefits of such granularity for the model. Delving deeper into circuit analysis, the second paper introduces a suite of metrics designed to capture nuanced aspects of quantum circuits:

1. **Program Communication:** This metric gauges the average degree of interaction among all qubits within a quantum circuit. A value of 1 signifies that each qubit



(a) Size

(b) Shots

(c) Backends

Figure 4.2: Job Distribution in Dataset

interacts at least once with every other qubit, providing insights into the overall communication dynamics of the quantum program.

2. **Critical Depth:** This metric measures the proportion of multi-qubit gates situated on the longest path, defining the depth of a quantum circuit. A value of 1 indicates that all multi-qubit gates are strategically placed along the longest path.

3. **Entanglement Ratio:** Reflecting the proportion of multi-qubit gates in a quantum circuit, this metric holds a value of 1 when the circuit comprises exclusively multi-qubit gates. It provides a measure of the entanglement density within the quantum program.

4. **Parallelism:** This metric quantifies the potential for parallelization within the circuit due to simultaneous gate execution. A value of 1 denotes substantial parallelization, offering insights into the circuit's capacity for concurrent gate operations.

5. **Liveness:** Assessing how frequently qubits remain idle, awaiting their next gate execution, this metric holds a value of 1 in circuits where each qubit experiences gate execution at every time step. Liveness provides a metric of the temporal dynamics and efficiency of gate utilization within the quantum circuit.

In order to obtain features for the entire job, we aggregate the features for individual circuits. For all counting features, such as gate counts, we sum up the values across all circuits in the job. Meanwhile, for metric features that have a range from 0 to 1, such as Program Communication, Critical Depth, Entanglement Ratio, Parallelism, and Liveness, we calculate the average across the circuits.

Following the feature engineering process, the next step is selecting a model capable of capturing dependencies between independent variables and the dependent variable. To make an informed choice, we conduct a model selection process, training multiple models and evaluating their performance through K-fold cross-validation. In this process, the dataset is divided into K subsets, and the model is trained and validated K times, each time using a different subset as the validation data and the remaining data for training. This ensures a comprehensive assessment of the model's generalization performance.

The scoring function employed for model evaluation is the R^2 score, also known as the coefficient of determination. R^2 score measures the proportion of the variance in the dependent variable that is predictable from the independent variables. It ranges from 0 to 1, with 1 indicating perfect prediction.

The formula for the R^2 score is given by:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{4.4}$$

where y_i represents the observed values, $\hat{y}_i$ represents the predicted values, $\bar{y}$ is the mean of the observed values, and n is the number of observations.

Among the models considered, the Extra Trees model emerges as the most accurate, achieving a R^2 score of 0.998. The Extra Trees model, or Extremely Randomized Trees, belongs to the ensemble learning category and operates by constructing a multitude of decision trees during training. Unlike traditional decision trees, Extra Trees introduces additional randomness in the selection of feature splits, leading to a diverse set of trees. The final prediction is obtained by averaging the predictions of individual trees, resulting in a robust regression model for our execution time estimation task.

4.4 Multi-Objective Optimization

At the core of the proposed scheduling system lies the Multi-Objective Optimization stage, representing a key component in achieving efficient resource allocation for quantum computing. Building on the estimated fidelities and execution times obtained in Section 4.2 and Section 4.3, this stage further integrates essential parameters such as the current waiting times of the backends, reflecting the anticipated duration for executing already scheduled jobs. Additionally, the sizes of the jobs and the capacities of the backends are considered. This diverse set of input data enables the formulation of a scheduling problem.

4.4.1 Problem Formulation

Addressing the imperative trade-off between fidelity and job waiting time, coupled with the inherent size constraints in quantum computing, we formulate the following optimization problem:

$$\begin{aligned}
\min \quad & f_1(x) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{Q} x_{ij} \left(\sum_{k=1}^{N} x_{kj} t_{kj} + w_j \right) \\
\min \quad & f_2(x) = \frac{1}{N} \sum_{i=1}^{N} \left(1 - x_{ij} f_{ij}\right) \\
\text{s.t.} \quad & x_{ij} q_i - s_j \leq 0 \quad \forall i = 1, .., N; \quad \forall j = 1, .., Q \\
& \sum_{j=1}^{Q} x_{ij} - 1 = 0 \quad \forall i = 1, .., N \\
& 0 \leq x_{ij} \leq 1 \quad \forall i = 1, .., N; \quad \forall j = 1, .., Q
\end{aligned} \tag{4.5}$$

where x_{ij} is a binary variable signifying the assignment of quantum job i to quantum computer j; N is the number of scheduled quantum jobs; Q is the number of available QPUs; w_j represents the approximate waiting time of the job queue to quantum backend j; t_{ij} denotes the estimated execution time of quantum job i on QPU j; f_{ij} is the estimated fidelity of quantum job i on quantum computer j; q_i stands for the maximum number of qubits in quantum job i; s_j signifies the size of quantum computer j.

The problem formulation presented here addresses a binary variable problem that optimizes two key objectives: mean waiting time and mean error. In our context, we consider error as the inverse of fidelity, aligning with the minimization nature of the problem. Calculating job waiting time involves summing up the estimated execution times of all jobs scheduled on the same backend as the current job. This sum is then

augmented by the approximate waiting time of the current job queue for the specific backend.

Employing the arithmetic mean in both objectives not only facilitates the aggregation of individual objectives but also contributes to a more balanced distribution of the computational load across multiple QPUs.

The initial constraint ensures that job assignments are restricted to backends with sufficient qubits to accommodate all circuits within a given job. The subsequent constraint dictates that each job can be scheduled on only a single QPU. The final constraint restricts variables to binary values, reinforcing the discrete nature of the assignment decisions.

As the number of backends and jobs grows, the dimensionality of the previously presented problem escalates as $\mathcal{O}(NC)$, posing challenges for optimization algorithms in terms of finding feasible solutions. To address this issue, we propose an alternative problem formulation:

$$\begin{aligned} \min \quad & f_1(x) = \frac{1}{N} \sum_{i=1}^{N} \left(w_{x_i} + \sum_{k=1}^{N} t_{kx_k} [x_i = x_k] \right) \\ \min \quad & f_2(x) = \frac{1}{N} \sum_{i=1}^{N} (1 - f_{ix_i}) \\ \text{s.t.} \quad & q_i - s_{x_i} \leq 0 \quad \forall i = 1, .., N \\ & 1 \leq x_i \leq Q \quad \forall i = 1, .., N \end{aligned} \tag{4.6}$$

In this formulation, x_i represents a discrete variable indicating the assignment of quantum job i to quantum computer x_i, with N being the number of scheduled quantum jobs and Q denoting the available QPUs. The objective function $f_1(x)$ seeks to minimize the waiting time, incorporating the approximate waiting time of the job queue to the assigned backend and the sum of estimated execution times for all jobs on that backend. On the other hand, $f_2(x)$ minimizes the error (inverse fidelity) across all assigned quantum computers.

The problem includes constraints ensuring that the qubit size of each job is compatible with the assigned quantum computer and that the assignment variable x_i remains within the available QPU range.

This discrete variable formulation maintains the same objectives as the binary counterpart in Equation 4.4.1 but eliminates the equality constraint, which has proven challenging for multi-objective algorithms. Notably, this formulation significantly reduces the problem's dimensionality growth to just $\mathcal{O}(N)$, making it independent of the number of backends and enhancing the tractability of the optimization process.

4.4.2 Optimization Algorithm

We employ the NSGAII algorithm, as introduced in Subsection 2.3.3, to tackle the formulated multi-objective optimization problem. Tailoring genetic operators to the specific problem types enhances the algorithm's effectiveness.

For binary variable problem, we utilize the following operators:

- **Binary Random Sampling:** This operator involves the random generation of binary strings to initialize the population.
- **Two Point Crossover:** This crossover method exchanges substrings between two parent solutions at two randomly chosen points.
- **Bitflip Mutation:** Mutation is performed by flipping individual bits in the binary representation of a solution.

For discrete variable problem, the following operators are employed:

- **Integer Random Sampling:** This operator initializes the population by randomly sampling integers within the valid range for discrete variables.
- **Simulated Binary Crossover (SBX)**: SBX simulates the crossover operation for real values represented in binary notation. It uses a probability distribution, following an exponential distribution, to perform the simulated crossover.
- **Polynomial Mutation:** This mutation operator perturbs a solution within a parent's vicinity using a polynomial probability distribution.

These operators are selected to ensure the diversity and exploration of the solution space for both binary and discrete variable problem formulations. The choice of specific operators aligns with the nature of each variable type, optimizing the algorithm's performance for the given problem constraints.

To determine when to conclude the optimization process, we employ a comprehensive set of termination criteria. Commonly considered indicators include the movement within the design space and the convergence observed in both the constraint and objective spaces. To set an upper limit on the algorithm's iterations, we define maximum thresholds for the number of generations and function evaluations, preventing the algorithm from prolonged execution. It is noteworthy that our tolerance termination strategy relies on a sliding window approach. Instead of evaluating only the latest generation, a sequence covering a defined period of generations is considered. This extended perspective allows for a more robust comparison of tolerances with predefined bounds. This holistic termination approach ensures that the optimization

process concludes effectively while taking into account both convergence behavior and computational limits.

Ultimately, the output of this optimization stage yields a Pareto front, showcasing a range of scheduling solutions with associated objective values. Each solution on the Pareto front represents a distinct trade-off between fidelity and execution time, providing a comprehensive view of the scheduling possibilities under consideration.

4.5 Multiple-Criteria Decision-Making

Multi-Criteria Decision-Making (MCDM) is a systematic process employed to evaluate and select the most suitable solution from a set of alternatives, considering multiple conflicting criteria. The application of the MCDM method is exemplified in the figure below:

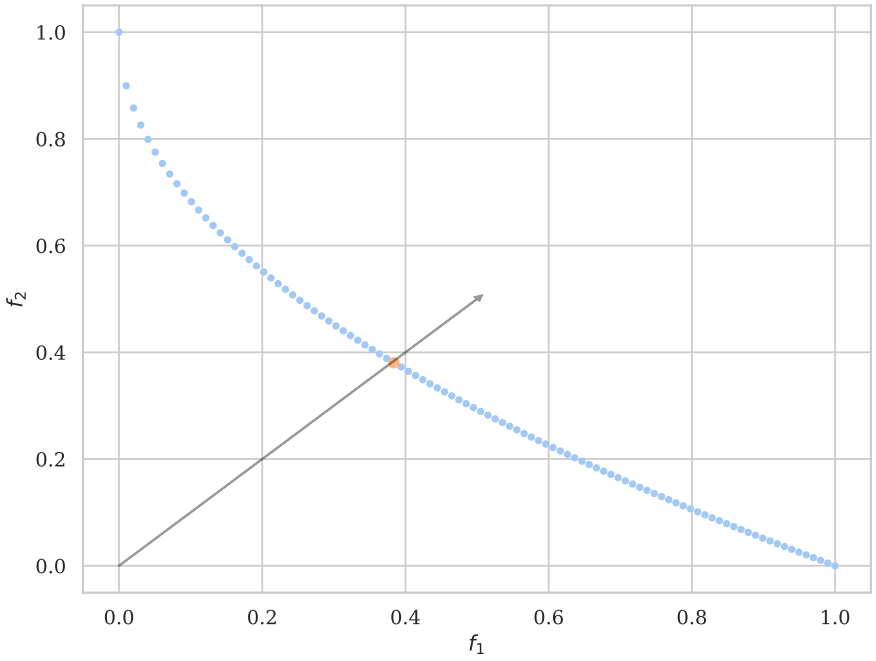


Figure 4.3: Application of MCDM Method on Pareto Front

In the context of our scheduling system, MCDM serves as a crucial step in determining a single solution from the Pareto front generated by the optimization algorithm. This selection is guided by the priorities assigned to the objectives, ensuring that the chosen solution aligns optimally with the desired emphasis on fidelity and execution

time.

In the realm of MCDM, various methods exist, each catering to specific scenarios. For our scheduling task, we opt for the Pseudo-Weights method, deemed suitable for convex Pareto fronts, which aligns with our optimization context. The calculation of pseudo weights denoted as $w_i(x)$ for the i-th objective function out of M objective functions for solution x, involves normalizing the distance to the worst solution concerning each objective. This provides an indicator of the solution's location in the objective space.

The pseudo-weight equation is expressed as:

$$w_i(x) = \frac{(f_i^{max} - f_i(x)) \,/\, (f_i^{max} - f_i^{min})}{\sum_{m=1}^{M} (f_m^{max} - f_m(x)) \,/\, (f_m^{max} - f_m^{min})} \tag{4.7}$$

Here, f_i^{min} and f_i^{max} denote the minimum and maximum objective values of objective i over all solutions in the Pareto front. The pseudo-weight $w_i(x)$ signifies the relative importance of the i-th objective for solution x within the entire Pareto front.

Subsequently, the solution x with a pseudo-weight vector closest to a desired preference vector $P = (p_1, p_2, ..., p_M)$ is selected. Each value p_i in the preference vector P indicates the importance assigned to objective i. The preference vector P should be defined such that $\sum_{i=1}^{M} p_i = 1$.

As a result of this stage, we obtain the solution that best aligns with the specified priorities for fidelity and execution time. This chosen solution represents the final schedule, encompassing job assignments to QPUs. This stage marks the conclusion of our scheduling process, and the jobs can be dispatched to the designated QPUs, given that they have already been transpiled.

5 Implementation

In this chapter, we delve into the technological stack and provide a detailed exploration of the implementation aspects of the proposed system. This section offers a comprehensive overview of the system's implementation from the underlying technologies to specific components.

5.1 Technological Stack

The project implementation employs Python [24] as the primary programming language due to its versatility, efficiency, and extensive ecosystem of libraries and frameworks. These features facilitate rapid prototyping and iterative development, well-suited for the dynamic and exploratory nature of research.

As our quantum cloud provider, we select IBM Quantum [6] due to its open-access model and robust platform, including access to its own quantum hardware. Furthermore, its quantum computing toolkit, Qiskit [15], offers seamless integration with both the platform and Python, aligning with our language choice. Qiskit supports the entire lifecycle of quantum jobs from creation to transpilation and execution on both real and simulated backends. Its modular structure covers various aspects of quantum computing, providing modules tailored for different applications and domains.

In our regression analysis, employed for estimating job execution times, we leverage the capabilities of scikit-learn [13], a widely-used open-source machine learning library. Python-based like the rest of our technological stack, scikit-learn seamlessly integrates into our workflow, providing support for supervised learning tasks. The library encompasses a rich set of tools for model fitting, data preprocessing, model selection, and evaluation.

For multi-objective optimization, we employ pymoo [2], a Python framework offering an extensive library of various single- and multi-objective optimization algorithms. Pymoo provides a suite of features tailored to multi-objective optimization, including visualization tools and support for MCDM. Notably, the framework enhances computational efficiency by leveraging compiled modules with computationally intensive functions implemented in Cython. In tandem with Numpy [5], an open-source numerical computing library for Python, our implementation benefits from efficient

matrix calculations, leveraging hardware acceleration to ensure optimal performance in tackling the optimization problem at hand.

5.2 Circuit Transpilation

One of the crucial challenges in executing quantum circuits on actual hardware is finding the optimal qubit mapping. Traditionally, attempts involve choosing an initial layout for the target system and then performing SWAP operations to map the circuit onto those qubits. However, this approach suffers from limitations. An initial layout chosen for noise characteristics might not be optimal for SWAP mapping, leading to either low-noise layouts with excessive SWAP gates or optimally mapped circuits on suboptimal qubits.

To overcome this challenge, we adopt a transpilation approach proposed by the mapomatic library [10]. This approach prioritizes finding the best low-noise sub-graph within the target system to execute the circuit. After transpiling the circuit, a post-processing step rewrites it to match the two-qubit gate structure of the chosen sub-graph. The VF2 mapper [3] identifies suitable sub-graphs, and a heuristic algorithm ranks them based on error rates derived from the current calibration data. This approach ensures that, for any given target system, mapomatic identifies the optimal set of qubits for executing the compiled circuit.

Integrating this transpilation strategy offers several benefits for our system:

- **Flexibility for Different QPU Types:** Our system provides the option to save computational resources by transpiling each circuit to each processor type instead of every individual QPU. However, finding the optimal mapping and scoring for each QPU remains necessary.
- **Custom Scoring Function:** Performing our own mapping allows us to implement custom scoring functions beyond the fidelity estimator described in Section 4.2. This enables incorporating more sophisticated scoring criteria in the future.

The transpilation process involves the following steps:

1. **Qiskit Transpilation:** We initially transpile the circuit using the native Qiskit transpiler. Although noise-unaware, this step allows efficient gate decomposition and optimization.
2. **Multiple Transpilation Runs:** Due to the stochastic nature of the SWAP mappers in Qiskit, the number of inserted SWAP gates can vary with each run. To mitigate this variability, we perform the transpilation process multiple times and select the instance with the minimum number of SWAP gates.

3. **Circuit Deflation:** Since transpilation inflates the circuit to the number of qubits in the target QPU, it hinders finding optimal mappings for small circuits. Therefore, we deflate the circuit after the initial transpilation to retain only the active qubits.

4. **Sub-Graph Matching and Scoring:** We employ the VF2 mapper to identify all matching sub-graphs within the target backend that can accommodate the deflated circuit. Each sub-graph is then evaluated using a fidelity score estimation, and the one with the highest score is chosen.

5. **Final Transpilation with Layout:** The final step involves transpiling the circuit once again using the Qiskit transpiler, but this time specifying the previously identified optimal layout for the target system.

This approach ensures efficient and accurate transpilation for executing quantum circuits on actual hardware. It balances the need for optimal qubit mapping with efficient processing and flexibility for different score functions.

5.3 Dataset Preparation

Building a robust regression model for execution time estimation requires a comprehensive and high-quality dataset. Our approach to dataset preparation involves three key stages:

1. **Data Collection**

   We begin by collecting raw data on job execution statistics from the IBM Quantum platform. This involves extracting detailed information about each job, including execution time, circuit details, and backend configuration. We leverage the IBM Quantum API to access this information and store it in a central SQLite database for efficient management and querying.

   Additionally, we extract the actual quantum circuits associated with each job and save them in the QASM 2.0 format. This format provides a standardized representation of the quantum circuits, facilitating further analysis and processing.

2. **Feature Extraction**

   Next, we delve into feature extraction, an essential step in preparing the data for model training. We iterate through each job in the database and calculate the relevant features described in Subsection 4.3.2. Our implementation draws inspiration from the work presented in [17] while incorporating modifications to enhance computational efficiency.

3. **Data Organization**

   Finally, we organize the extracted data into a format suitable for model training. We save the calculated features as independent variables and the corresponding execution times from the job metadata as dependent variables. Both sets of data are stored as NumPy arrays for efficient processing and manipulation by the machine learning algorithms.

5.4 Model Selection and Training

Selecting the suitable model and optimizing its hyperparameters are essential for achieving accurate execution time estimates. Our system utilizes an approach that leverages popular regression models from scikit-learn and employs best practice techniques to ensure robust performance.

We begin by evaluating several prominent regression models, including Extra Trees, Random Forest, Gradient Boosting, Adaboost, and Polynomial Regression. Each model possesses distinct strengths and weaknesses, making it crucial to identify the one that best captures the relationship between job characteristics and execution time.

However, simply training each model and comparing their performance doesn't provide a comprehensive evaluation. Each model comes equipped with hyperparameters that significantly influence its predictive power. To optimize these parameters and unlock the full potential of each model, we employ grid search. This technique systematically explores a predefined grid of hyperparameter values, evaluating the model's performance on each combination. By analyzing this performance landscape, we identify the optimal hyperparameter configuration that maximizes predictive accuracy.

To assess the effectiveness of each model and its chosen hyperparameters, we utilize the R^2 score, a metric described in 4.3.2. We employ K-fold cross-validation to estimate the R^2 score and ensure statistically reliable results. To further eliminate potential data leaks and ensure a truly unbiased evaluation, we utilize nested K-fold cross-validation. This nested approach involves an inner and an outer loop. The inner loop performs grid search and hyperparameter tuning within each K-fold of the outer cross-validation. This effectively prevents data leakage from hyperparameter tuning to the final performance estimation, leading to a more accurate and reliable assessment of the model's performance.

Once the model with the best performance and optimal hyperparameters is identified, we save it as a pickle file. This serialized representation allows for easy loading and deployment, enabling the model to be readily utilized for future execution time prediction tasks.

5.5 Optimization Performance

At the core of our scheduling system lies a robust optimization algorithm tasked with finding the optimal job-to-backend assignments. To achieve this, we leverage the flexibility of the pymoo library, which allows us to formulate both discrete and binary variable problems.

We represent the scheduling problem using the `ElementwiseProblem` class provided by pymoo. This class forms the backbone of our optimization process, defining the objective function and any constraints we wish to enforce. Within this class, we implement the `__evaluate()` method, the key component that calculates the objective values and constraint violations for each solution within the current population. To ensure efficient calculation and minimize computational time, our implementation utilizes matrix operations provided by the NumPy library.

Furthermore, we leverage the parallelization capabilities of pymoo to achieve even more significant performance gains. We employ the pymoo functionality that integrates seamlessly with the standard Python multithreading library. By creating a thread pool with the number of threads equal to the available CPU cores and passing it to pymoo, we enable the concurrent evaluation of solutions during the optimization process. This distributes the computational workload efficiently across available CPU cores, further accelerating the search for the optimal solution.

6 Evaluation

This chapter delves into the evaluation of our quantum scheduling system. Here, we assess the core components of the system in isolation, ensuring their individual performance meets the design goals. Subsequently, we conduct an end-to-end evaluation, examining the system's overall functionality and effectiveness in orchestrating quantum jobs across diverse QPUs.

6.1 Evaluation Setup

To conduct a comprehensive evaluation of our proposed system, we leverage a robust infrastructure that guarantees reliable and reproducible results. The evaluation environment employs machines featuring AMD EPYC 7713P 64-Core processors and 0.5 TB of RAM. Additionally, the utilization of NixOS as the operating system establishes the reproducibility of experiments.

On the quantum front, we utilize the IBM Quantum open access plan, providing access to a wide range of QPUs. Additionally, we leverage Qiskit's fake backends, which realistically mimic the characteristics of real IBM hardware. These simulated backends provide the foundation for controlled and scalable testing, enabling us to explore a broader range of scenarios without incurring the limitations of real hardware access.

To generate the quantum jobs used for evaluation, we rely on the MQT Benchmark library [16]. This benchmark suite offers an extensive array of over 70,000 benchmark circuits, encompassing circuits from 2 to 130 qubits in size. MQT Benchmark covers most of the standard quantum algorithms, including Variational Quantum Eigensolver (VEQ) [14], Grover's [4], and Shor's [21] algorithms. Moreover, it provides dedicated application benchmarks tailored specifically for variational quantum algorithms. These benchmarks span various domains, including optimization, machine learning, finance, and natural sciences.

6.2 Execution Time Estimation

6.2.1 Graph Traversal

We begin our evaluation of the execution time estimation module by focusing on the Graph Traversal method presented in 4.3.1. To assess its accuracy, we generate jobs containing random benchmark circuits, varying the number of circuits within the job. We start with a single circuit and progressively increase the number until we reach the maximum of 100 circuits per job, as stipulated by the IBM Open Plan quota.

These jobs are then submitted for execution on the QPU with the shortest queue (ibm_perth in our case), ensuring rapid results. We fix the number of shots for all jobs on 4000 (default IBM value) and extract the actual execution time from the job metadata upon completion. This data is then plotted alongside the estimates provided by the Graph Traversal method, as presented in the figure below:

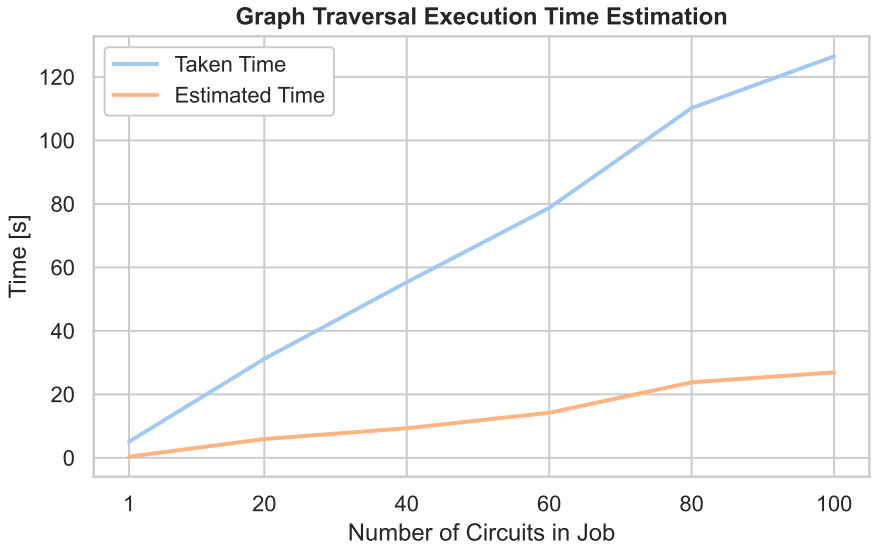


Figure 6.1: Graph Traversal Execution Time Estimation

As the figure clearly demonstrates, the initial implementation of the Graph Traversal method significantly underestimates the actual execution time. This discrepancy becomes increasingly pronounced as the size of the job grows. We identify two primary factors contributing to this underestimation:

1. **Cold Start Time:** The current method does not account for the cold start time associated with the initiation of each job. This initial delay can significantly impact the overall execution time, particularly for smaller jobs.

2. **Repetition Delay:** We discover that IBM Quantum adds an additional repetition delay between each shot of each circuit. This delay can be extracted from the backend properties and is not factored into our initial calculations.

After analyzing our execution data, we estimate the cold start time to be approximately 3 seconds. By incorporating these newly discovered factors into our Graph Traversal method, we obtain the following improved results:

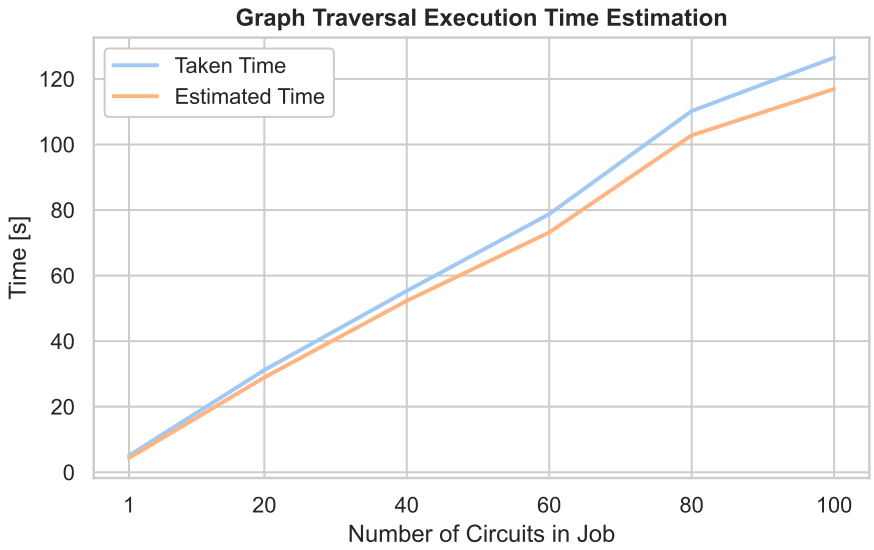


Figure 6.2: Improved Graph Traversal Execution Time Estimation

Compared to the initial results, we observe a marked improvement in the accuracy of our predictions. However, there still persists a noticeable underestimation for larger jobs. While this does not necessarily hinder our optimization process, as our method remains effective for comparing the execution times of different job-to-QPU assignments, it renders the estimates unreliable for direct user consumption. This is particularly relevant for users who rely on these estimates to predict their job execution costs, as quantum providers like IBM charge based on execution time.

The source of these discrepancies could lie in several factors, including:

- **Hidden delays:** There might be additional, undocumented delays introduced by IBM that our model does not currently account for.
- **Gate exertion underestimation:** The reported gate execution delays might be underestimated by IBM, leading to inaccurate predictions.

Further investigation into these factors is necessary to refine this execution time estimation method and ensure its accuracy for larger jobs.

6.2.2 Regression Model

After our experimentation with the Graph Traversal method, we accumulate a substantial dataset of job executions. This dataset inspires us to tackle the execution time estimation problem from a different perspective, namely regression analysis. To this end, we propose an approach utilizing a regression model, described in 4.3.2.

Feature Selection

To assess the performance of this method, we begin by analyzing the suggested features used by the model to make predictions. We leverage the scikit-learn library, which offers a basic set of tools for feature selection.

First, we utilize a variance threshold selection, a simple baseline approach to feature selection. This method removes all features whose variance does not meet a certain threshold. By default, all zero-variance features, i.e., features with the same value in all samples, are removed. Our analysis reveals that out of 51 initial features, only 19 have non-zero variance. The remaining features have zero variance, rendering them useless for a regression model. All of these features are derived from the QASM 3.0 gate counts, suggesting that current IBM QPUs only operate on a minority of the gate types listed in the QASM 3.0 specification.

Next, we apply univariate feature selection, which selects the best features based on univariate statistical tests. We begin with an F-test, calculating the F-score for all features. The F-test is a statistical test that compares the variance of two groups to determine whether they are significantly different. In the context of feature selection, the F-test is used to assess whether the variance of a feature's values between different samples is significantly greater than the variance of the target variable's values. It is performed in two steps:

1. The cross-correlation between each regressor and the target is computed using the Pearson correlation coefficient:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{6.1}$$

   where x_i and y_i are the i-th values of the regressor and target variables, respectively, and $\bar{x}$ and $\bar{y}$ are the respective means. The Pearson correlation coefficient ranges from -1 to 1, with 1 indicating a perfect positive correlation, -1 indicating a perfect negative correlation, and 0 indicating no correlation.

2. Next, this correlation is converted into an F-score and to a p-value, with the features ranked accordingly based on their positive correlation with the target.

A higher F-score indicates a greater difference in variance between the two groups, suggesting that the feature is more predictive of the target variable. We sort the features by the resulting score and show the results in the figure below:

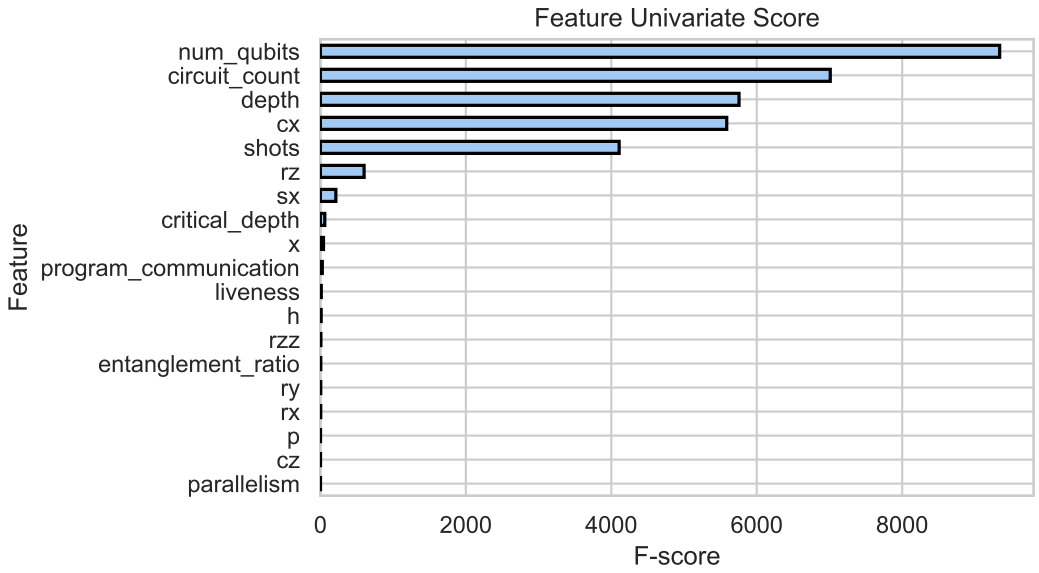


Figure 6.3: Feature F-Score

As can be seen, only 5 out of 19 features had significant scores. This suggests that a majority of the initial features are not informative for execution time estimation.

The F-test estimates the degree of linear dependency between two random variables. However, it is limited in its ability to capture more complex forms of statistical dependency. Mutual information, on the other hand, is a metric that can capture any kind of statistical dependency between two random variables. It is defined as the amount of information that one variable tells us about the other. Mutual information is always non-negative, with a value of zero indicating that the two variables are independent and higher values indicating stronger dependency.

Mutual information between two random variables X and Y is calculated as follows:

$$I(X;Y) = H(X) - H(X|Y) \tag{6.2}$$

where $H(X)$ is the entropy of X, and $H(X|Y)$ is the conditional entropy of X given Y.

The entropy of a random variable is a measure of its uncertainty. The conditional entropy of a random variable X given another random variable Y is the amount of

uncertainty remaining in X after observing Y. Therefore, mutual information measures how much the uncertainty in X is reduced by observing Y. If X and Y are independent, then observing Y will not reduce the uncertainty in X, and hence the mutual information will be zero. On the other hand, if X and Y are perfectly dependent, then observing Y will completely remove the uncertainty in X, and the mutual information will be equal to the entropy of X.

We calculate the mutual information score for each of the features and sort the features by their scores. The resulting plot is shown below:

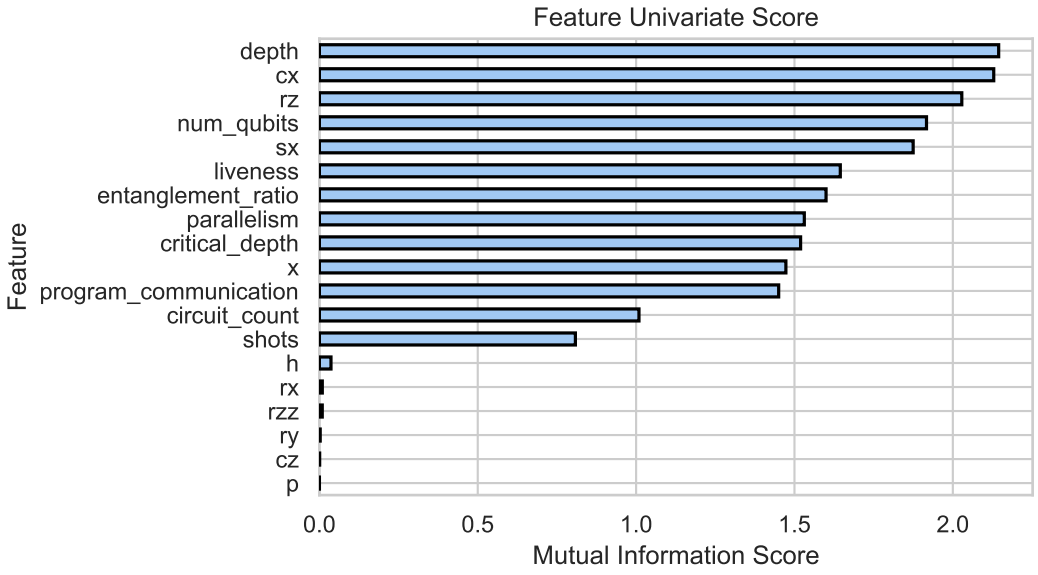


Figure 6.4: Feature Mutual Information Score

Unlike the F-test results, we can see that 13 out of 19 features have significant mutual information scores. This suggests that many of the proposed features are informative for execution time estimation, even though they may not have a linear relationship with execution time.

The previous analysis of the dataset features provides valuable insights into the potential informativeness of each feature for execution time estimation. To further explore the importance of each feature for our Extra Trees regressor, we can leverage the Mean Decrease in Impurity (MDI) metric, which is provided by all the tree-based models.

Impurity is quantified by the splitting criterion of the decision trees (Gini, Log Loss, or Mean Squared Error). MDI measures the average decrease in impurity over all trees in the model when a given feature is used to split nodes. A higher MDI score indicates

that the feature is more important for predicting the target variable. MDI is calculated as follows:

$$MDI(x_i) = \frac{1}{N_t} \sum_{t=1}^{N_t} \left[\text{Impurity}(t) - \text{Impurity}(t_{\text{split}}) \right] \tag{6.3}$$

where x_i, is the feature of interest; N_t is the number of trees in the forest; t is a tree in the forest; Impurity(t) is the impurity of the tree before splitting on feature x_i; Impurity$(t_{\text{split}}$ is the impurity of the tree after splitting on feature x_i.

We extract the feature importances from the trained Extra Trees model and plot them in the following figure:

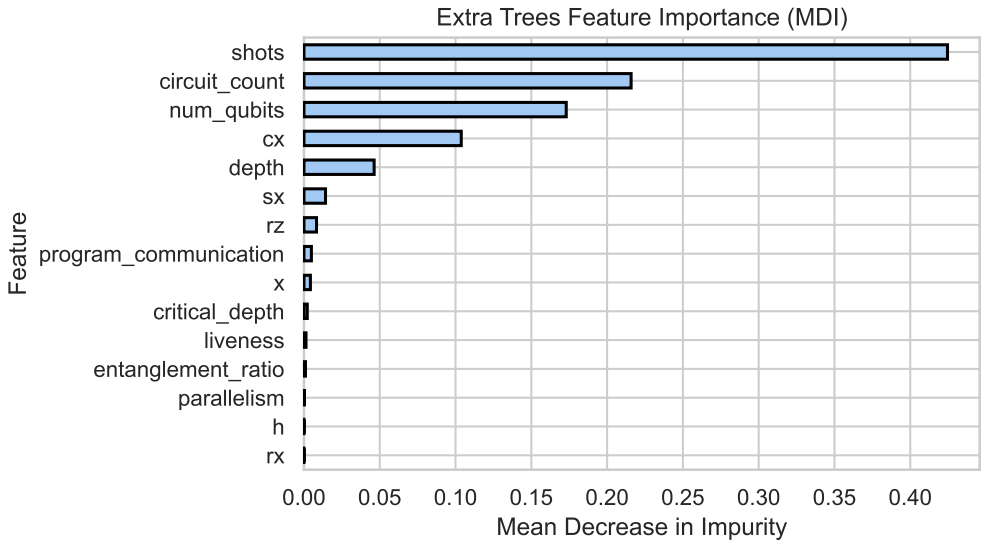


Figure 6.5: Mean Decrease in Impurity

As the plot suggests, only five features seem to be important for our model:

1. **shots:** The number of times a job is executed;
2. **circuit_count**: The number of circuits in a job;
3. **num_qubits**: The total number of qubits used in all of the circuits in a job;
4. **cx**: The number of CX gates in the job. CX gates are typically used to implement SWAP operations, which are considered to be one of the most computationally expensive operations on quantum computers;
5. **depth**: The total depth of all the circuits in a job.

These findings are consistent with our intuition about the factors that influence execution time. For example, a higher number of shots, circuits or CX gates will generally result in a longer execution time of a job. Additionally, circuits with a larger number of qubits or a deeper depth are more complex and will also take longer to execute.

While the MDI is a valuable metric for assessing feature importance, it can be susceptible to overfitting. Overfitting occurs when a model learns the training data too well and is unable to generalize to new data. In this case, the MDI scores may assign high importance to features that are not predictive on unseen data.

Permutation-based Feature Importance (PFI) is a more robust metric that avoids this issue. PFI measures the decrease in model performance when a single feature value is randomly shuffled. This breaks the relationship between the feature and the target variable, so the drop in model performance indicates how much the model depends on that feature. To calculate PFI, we randomly shuffle the values of each feature in the held-out testing set and evaluate the model performance on the shuffled data. We repeat this process multiple times and average the results.

The Figure 6.6 shows that the features ranked by PFI are the same as the features ranked by MDI. This suggests that our model is not overfitting and that the features identified as important by MDI are also predictive on unseen data.

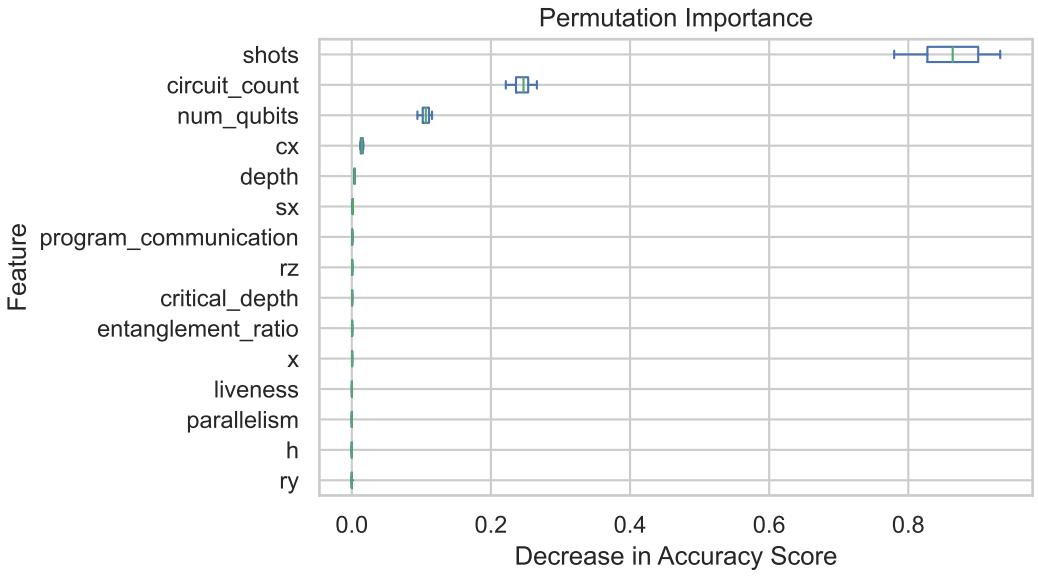


Figure 6.6: Permutation-based Feature Importance

Prediction Accuracy

Having identified informative features and ensured our model is not overfitted on the training data, we can now assess its performance in predicting actual execution times. As mentioned in Section 4.3.2, our Extra Trees model achieves R^2 score of 0.998 on our dataset, calculated using nested K-fold cross-validation.

To further evaluate the model's prediction accuracy, we replicate the approach used for the Graph Traversal method. We generate jobs with varying circuit counts, ranging from 1 to 100, and execute them with a fixed number of shots (4000) on the same backend with the shortest waiting time (ibm_nairobi in our case). We then compare the actual job execution times to the predictions provided by our regression model. The results are presented in Figure 6.7.

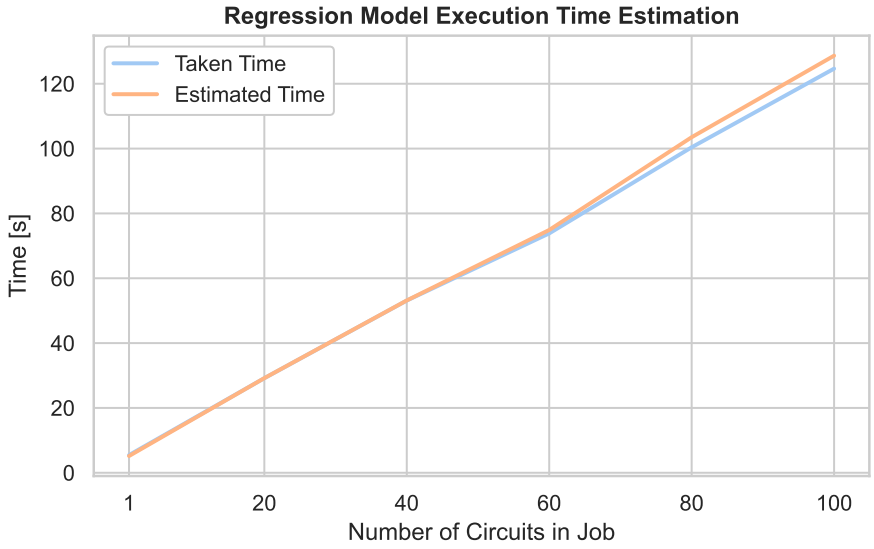


Figure 6.7: Regression Model Execution Time Estimation

As the figure illustrates, our model exhibits near-optimal accuracy in predicting execution times for small to medium-sized jobs. However, for larger jobs, it tends to overestimate execution time, potentially due to the skew in our dataset towards smaller jobs.

Despite this minor overestimation, the Regression Model method significantly outperforms the Graph Traversal method in terms of accuracy. Consequently, we choose it as the default execution time estimation method within our scheduling system.

Furthermore, the regression model's estimations are more suitable for user information display as they avoid underestimation and provide a reliable upper bound for

job execution cost prediction. This can provide users with valuable information for planning their workflows and managing their resources effectively.

6.3 Optimization and MCDM

The multi-objective optimization module lies at the core of our scheduling system. Its key function is to identify the Pareto front of solutions, offering various trade-offs between waiting time and execution fidelity. To assess its effectiveness, we simulate a scenario scheduling 100 randomly generated jobs. These jobs follow a normal distribution with a mean size of 50 and a standard deviation of 20.

To eliminate the influence of uneven queue sizes on real backends, we utilize 8 simulated QPUs, setting their initial queue sizes to zero. This ensures that the optimization solely relies on the execution times of scheduled jobs. We feed these jobs and backends to our optimization module, obtaining the following results:

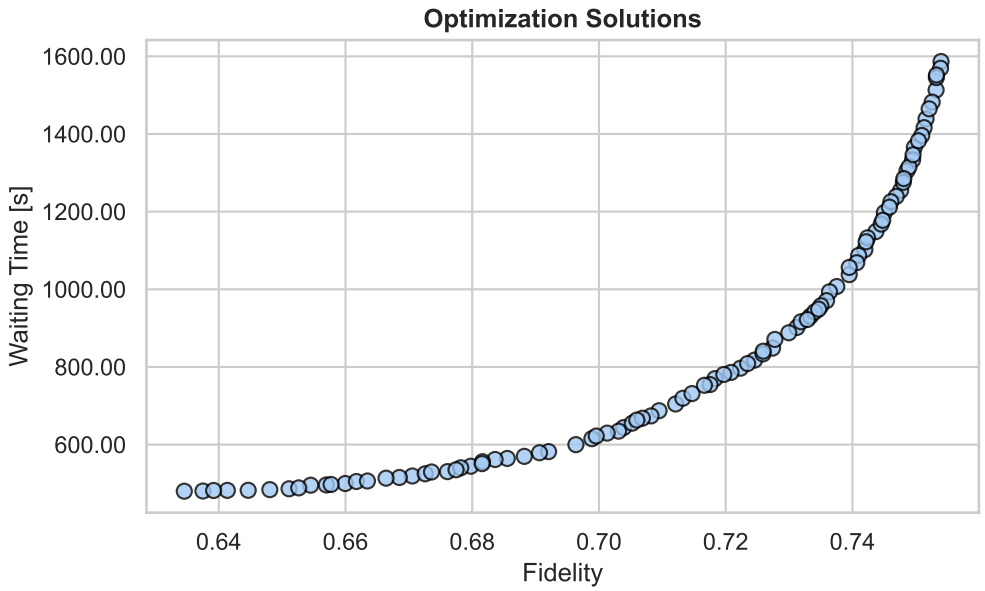


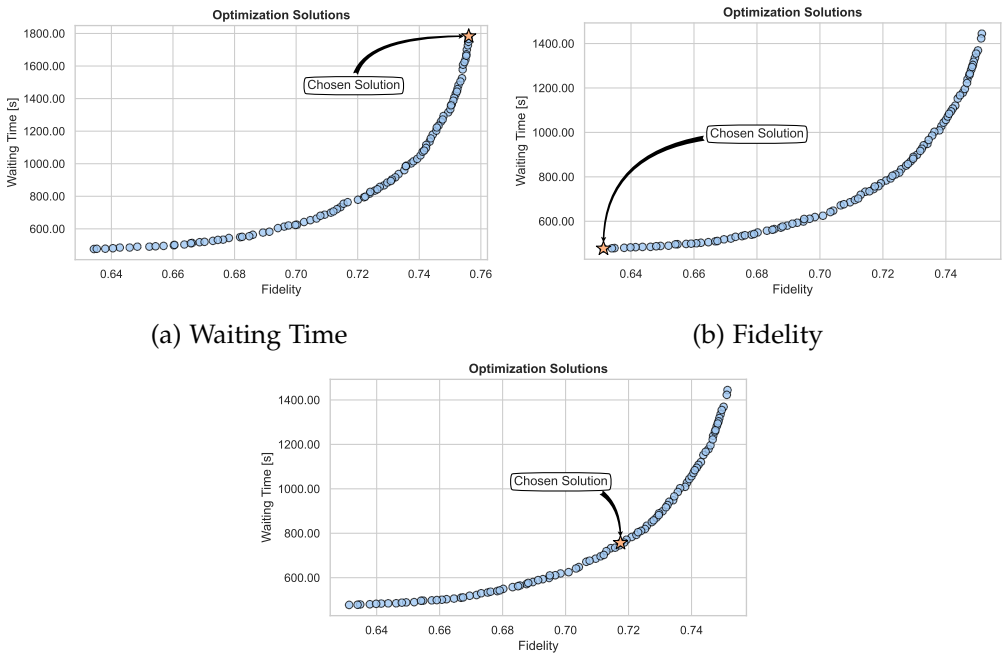
Figure 6.8: Pareto Front of Optimization Solutions

The figure reveals that our algorithm effectively generates a convex Pareto front of solutions. Notably, the results suggest that a mere 3% sacrifice in fidelity could lead to a reduction in waiting time by half. This highlights the importance of our scheduling system, as users often lack the information to identify such optimal trade-offs between waiting time and fidelity.

Furthermore, we evaluate the ability of our MCDM module to select the solution that best aligns with the system priorities for the two objectives. We test it in three scenarios:

1. **Priority on Waiting Time:** Assigning zero weight to fidelity and a weight of 1 to waiting time, the MCDM module should choose the solution guaranteeing the fastest job execution.
2. **Priority on Fidelity:** Applying a weight of zero to execution time and a weight of 1 to fidelity, the MCDM module should favor the solution with the highest fidelity.
3. **Balanced Trade-Off:** Setting equal weights of 0.5 to both objectives, the MCDM module should find a solution that balances fidelity and waiting time.

The results of the MCDM module are presented in the following figure:



(a) Waiting Time

(b) Fidelity

(c) Trade-off

Figure 6.9: MCDM Results with Different Priorities

The figure clearly illustrates that our MCDM module performs as expected across all three scenarios. It consistently identifies solutions that best correspond to the specified

priorities, further demonstrating its ability to tailor scheduling decisions to specific system needs.

6.4 Scheduling Stages Performance

While the previous sections establish that the system components satisfy the functional requirements, we also need to get insights into their performance characteristics. To achieve this, we generate batches of jobs, ranging from 1 to 100 jobs, and submit each batch to our scheduling system alongside 8 quantum backends. During every scheduling cycle, we measure the runtime of each component and present the collected results in Figure 6.10.

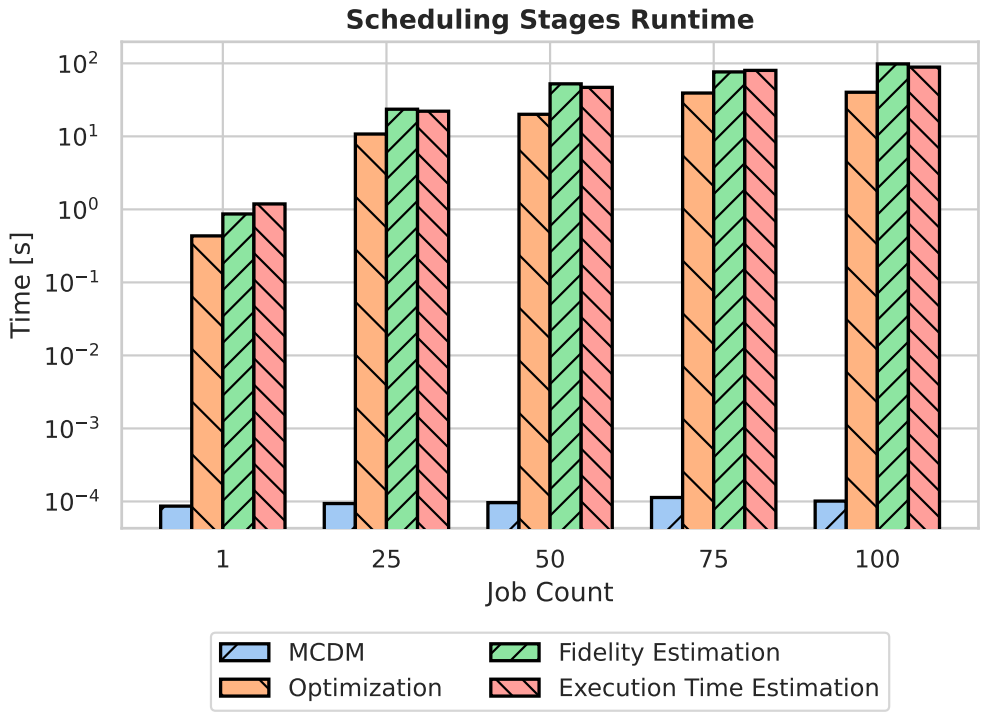


Figure 6.10: Scheduling Stages Runtime Comparison

The plot reveals that the MCDM component incurs a negligible amount of runtime. On the other hand, the preparatory stages - fidelity estimation and execution time estimation, account for the most significant portion of the system runtime, with each stage having approximately the same duration. The optimization stage, which is the core of the scheduling process, consumes only a quarter of the time required

for the preparatory stages. This fact suggests that the system can achieve significant performance gains if we either improve the performance of the estimation stages or perform these estimations in advance.

6.5 Quantum Cloud Simulation

Having assessed the individual components of our scheduling system and confirmed their effectiveness, we complete this chapter with a comprehensive end-to-end evaluation. This final stage replicates the real conditions of a quantum cloud environment, allowing us to observe the system's performance in its entirety and gain insights into its overall functionality and effectiveness.

6.5.1 Workload Analysis

Mimicking real quantum cloud conditions requires an understanding of its workloads. We continuously monitor all available quantum resources on the IBM Quantum platform to gather this information. Since we cannot access the jobs in the backend queues, we cannot analyze their characteristics. The only statistical information we can obtain is the backend's status, including the queue length. Therefore, we limit our workload analysis to analyzing the number of incoming jobs.

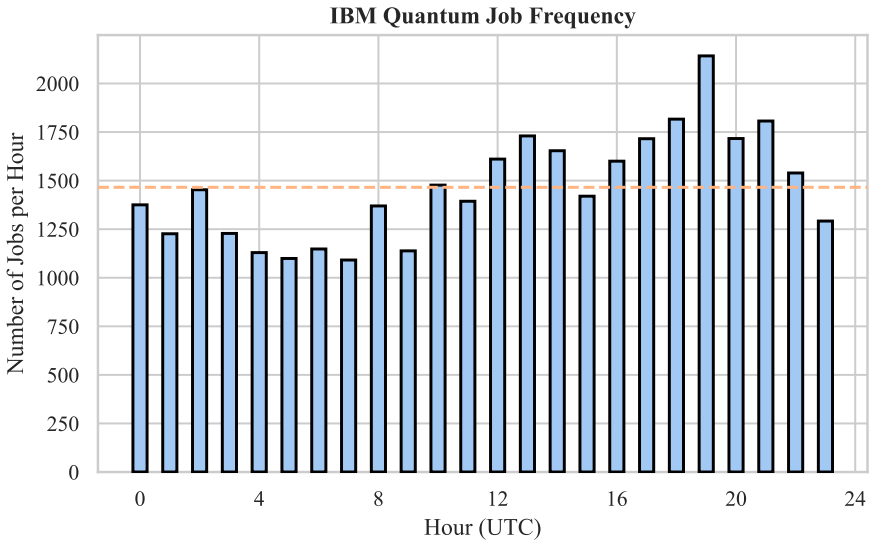


Figure 6.11: IBM Quantum Workload

We collect the status information every 10 seconds to avoid being rate-limited by the higher frequency requests. After tracking this information for 10 days, we aggregate it and analyze the differences in queue sizes for each backend to determine the number of incoming jobs. We then aggregate the data for all days and backends and plot the arithmetic means for the number of new jobs for each hour. The resulting distribution is shown in Figure 6.11.

The figure reveals that the number of incoming jobs exhibits a significant variance across different hours of the day, ranging from 1100 to 2050 jobs per hour. This suggests that user activity is not uniformly distributed throughout the day, with some hours experiencing higher demand than others. The most indicative statistic for our simulation is the total average of all hours, displayed as an orange dotted line in the figure. This value, approximately 1500 jobs per hour, provides us with a baseline for our evaluation.

6.5.2 Simulation Environment

In order to assess the performance of our quantum scheduling system under realistic conditions, we now construct a simulation environment. This environment consists of two key modules: the scheduling manager and the load generator. These modules operate as separate processes, communicating via a multiprocess queue.

Load Generator

The load generator takes on the responsibility of creating a synthetic workload that closely mirrors the observed real-world job arrival patterns. It accomplishes this by randomly generating jobs with a fixed number of shots (4000 in our case) and varying circuit sizes following a normal distribution with a mean of 50 and a standard deviation of 20. The rationale behind these numbers stems from the lack of publicly available statistics about job sizes. We, therefore, rely on estimates based on the fact that the IBM Open Plan allows scheduling up to 100 circuits per job. As circuits, we use benchmarks provided by the MQT Bench library.

These jobs are then submitted to the multiprocess queue with a fixed frequency, simulating the arrival rate of jobs in the real world. Once a predetermined number of jobs has been submitted, the load generator gracefully exits.

Scheduling Manager

The scheduling manager plays a key role in orchestrating the execution of jobs. It continuously monitors the state of the queue, waiting for the defined moment to invoke our scheduling system. We employ two mechanisms to trigger scheduling:

- **Queue size threshold:** If the queue size reaches a specified limit (100 in our evaluation), the scheduling manager initiates the scheduling process;
- **Time-based trigger:** If a pre-defined time interval elapses (120 seconds in our evaluation), the scheduling manager triggers scheduling regardless of the queue size.

To schedule the jobs effectively, the scheduling manager needs information about the available backends. We achieve this by utilizing fake QPUs provided by Qiskit, supplemented with additional functionality. We patch each backend with the ability to maintain its own queue of scheduled jobs and accurately estimate current waiting times. These enhancements are possible because we have full access to the internal workings of the simulated QPUs, unlike real backends where access is limited. Additionally, our fake QPUs are patched with a notion of time, ensuring that jobs progressively exit the queue as time advances, reflecting the real-world flow of jobs through the system.

After each scheduling cycle, the scheduling manager receives the results and assigns jobs to the queues of the chosen backends. This process continues iteratively, simulating the ongoing arrival and execution of jobs in a real quantum cloud environment.

6.5.3 Scheduling Results

Building upon the insights obtained from our workload analysis, we utilize our simulation environment to evaluate the performance of our scheduling system. As a baseline workload for our load generator module, we adopt the previously determined job arrival rate of 1500 jobs per hour, translating to 1 job submitted every 2.4 seconds.

One of the primary challenges we aim to address with our proposed system is the issue of unevenly distributed load across backends in the current quantum cloud environment. To assess how effectively our system addresses this issue, we collect data throughout the simulation, tracking the total execution time allocated to each backend over a one-hour period.

The resulting distribution for the eight simulated backends is presented in Figure 6.12.

As the figure demonstrates, the distribution of resource utilization across backends is nearly uniform. This can be attributed to our mean waiting time objective function, which penalizes scheduling jobs on the same backends multiple times as subsequent jobs are assigned increasingly longer waiting times. This result provides evidence that our system effectively addresses the issue of uneven workload distribution, promoting a fairer and more balanced utilization of available resources.

To further validate our system's effectiveness, we explore its scalability under increasing workloads. As quantum computing is a rapidly evolving field, it is crucial that our

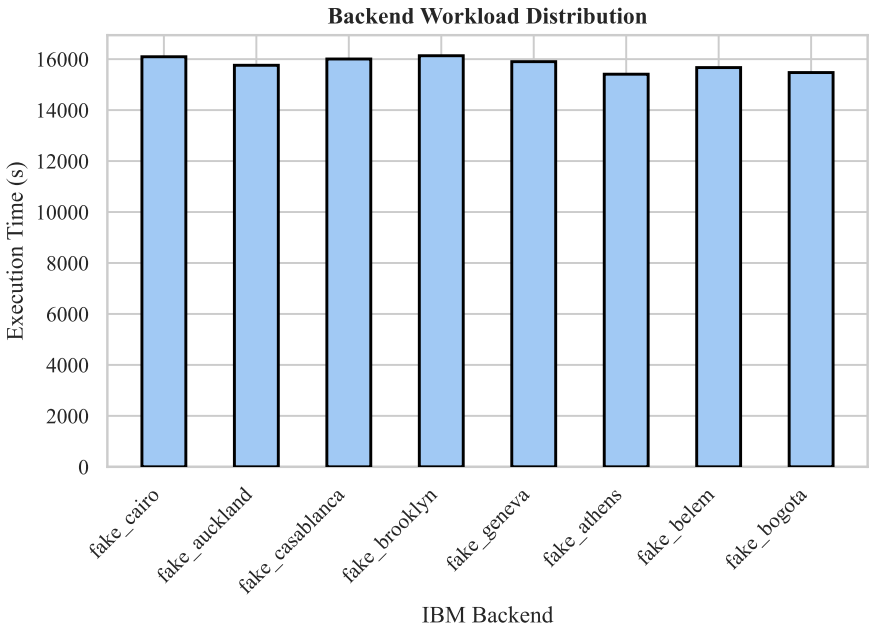


Figure 6.12: Backend Workload Distribution

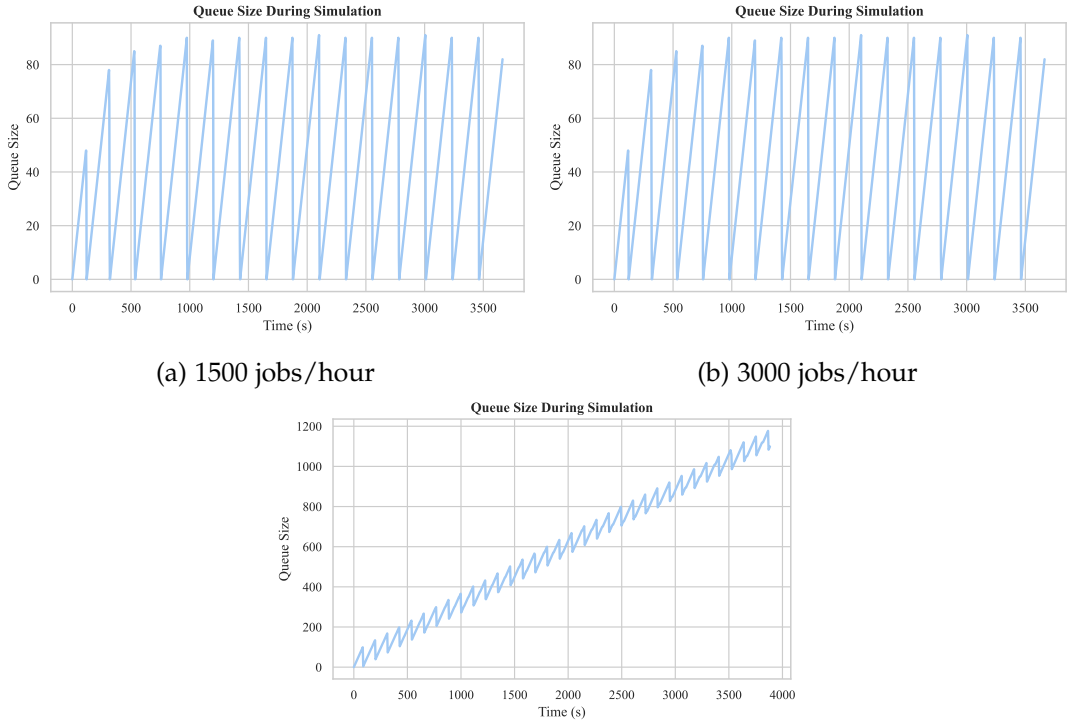
system not only handles the current quantum cloud workload but also has sufficient buffer for future growth.

To evaluate this, we monitor the length of the job queue between the scheduling manager and the load generator. If the queue size remains constant over time, it indicates that the scheduling system can keep pace with incoming jobs. Conversely, a growing queue signifies that the system is falling behind, and jobs are arriving faster than they can be processed.

We plot the results of this evaluation considering three scenarios: the current quantum cloud workload (1500 jobs per hour), a doubled workload (3000 jobs per hour), and a tripled workload (4500 jobs per hour). The results are presented in Figure 6.13.

As the plot illustrates, our system successfully handles the current workload and remains stable even when the workload is doubled. However, when faced with a tripled workload, the system begins to lag behind, indicating its limitations. While these results demonstrate that our system possesses a substantial buffer for future quantum cloud growth, we believe further optimization is possible to enhance its scalability and ensure its continued effectiveness as the field progresses.

To ensure that our system maintains its ability to identify the Pareto front of scheduling solutions and strike a balance between fidelity and waiting time throughout the



(a) 1500 jobs/hour

(b) 3000 jobs/hour

(c) 4500 jobs/hours

Figure 6.13: Job Queue Size with Increasing Workloads

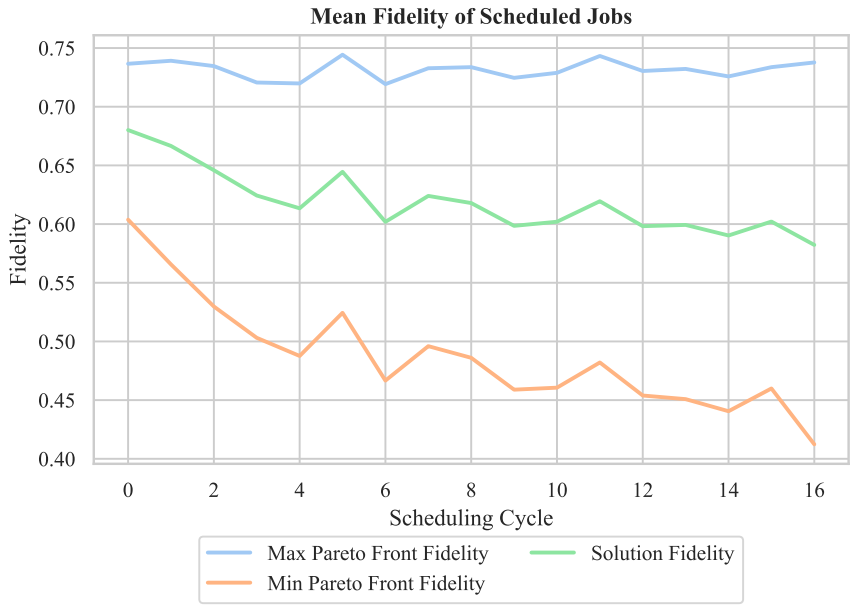
ongoing simulation, we collect the objective values for all found solutions during each scheduling cycle. This allows us to analyze the evolution of the optimization process over time.

We plot the minimum and maximum values of the objective functions for each cycle, alongside the objective values of the ultimately chosen solution. The resulting plots for fidelity and waiting time are presented in 6.14.

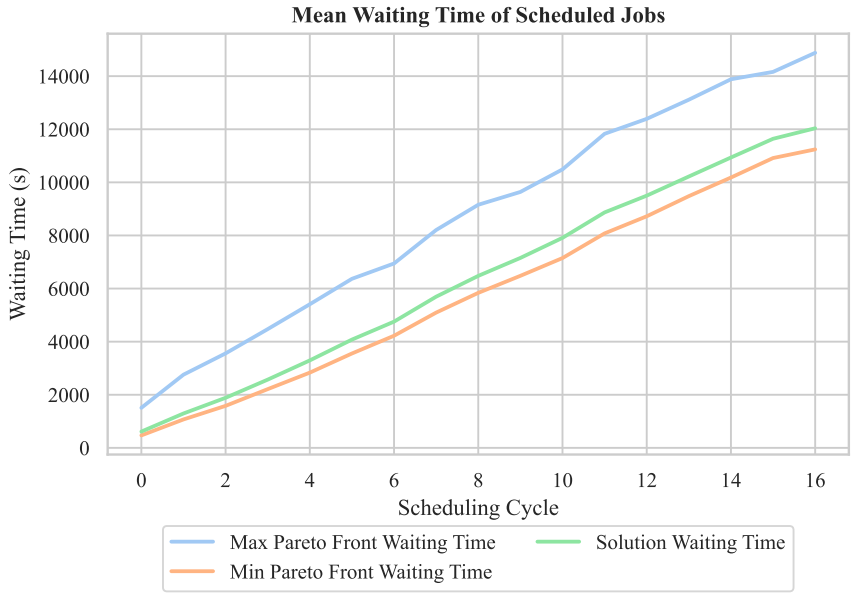
As the plots demonstrate, our algorithm shows stable performance throughout the simulation. It consistently identifies diverse solutions, even as the backends' waiting times increase. This implies that our system effectively avoids falling into suboptimal local optima and maintains its ability to explore the entire Pareto front.

Furthermore, the plots reveal a consistent preference for solutions that strike a balance between fidelity and waiting time. The chosen solutions consistently gravitate towards the minimum waiting time and average fidelity, showcasing our system's ability to prioritize efficient job execution while ensuring acceptable fidelity levels.

While the waiting time plot suggests that the system effectively identifies solutions



(a) Fidelity



(b) Waiting Time

Figure 6.14: Optimization Solutions Objectives throughout Simulation

throughout the simulation, it also reveals a continuous growth in waiting time despite the simulated backends' notion of time. This can be attributed to the high volume of incoming jobs exceeding the processing capacity of the available resources.

To better understand the impact of available quantum resources on job waiting times, we analyze the performance of our system with varying numbers of QPUs: 4, 8, and 16. Figure 6.15 depicts the resulting mean waiting times of the chosen solutions for each scenario.

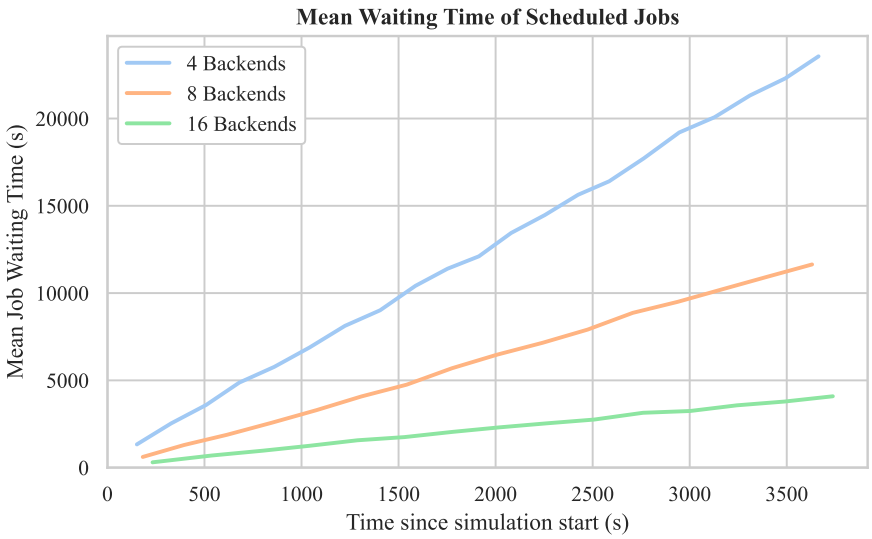


Figure 6.15: Mean Waiting Time with Varying Number of Backends

As anticipated, the increase in available QPUs leads to a progressively shallower slope in the mean waiting time plot. This observation indicates that our proposed system effectively adapts to the growing number of quantum resources by distributing the workload evenly across them, facilitating a significant reduction in waiting times. This analysis demonstrates the system's scalability not only in terms of the job workload but also in terms of the growing quantum providers' capacities. This scalability ensures that the system will remain relevant even as the quantum cloud evolves and more resources become available.

7 Related Work

While the field of scheduling has received significant attention in the realm of classical computing, its application to quantum computing remains in its early stages due to the relative infancy of the technology. As a consequence, the area of quantum scheduling is still relatively underdeveloped.

One notable group of works, including [7] and [11], proposes various quantum resource allocation schemes for the dynamic quantum cloud. These schemes, similar to our system, optimize for multiple objectives, often tackling conflicting metrics like the cost of spinning up additional resources for quantum circuit execution and the waiting time of the circuit. However, a crucial distinction lies in the scope of their work. While these studies focus on resource allocation, our proposed system tackles a different aspect of the problem: task scheduling, as elaborated upon in Subsection 2.2.2.

Beyond the realm of resource allocation, another prominent group of methods addresses the same challenge of task scheduling in the quantum cloud, sharing our goal of optimizing for conflicting metrics. [19] proposes a system that optimizes for the same metrics as our system: fidelity and waiting time. They achieve this optimization through a utility function, seeking a solution that offers a balanced trade-off between these competing objectives. Similarly, [20] presents a system that also optimizes for fidelity and waiting time. They achieve this by introducing metrics that influence these objectives and assigning weights to each metric. These weighted metrics are then used to rank all backends utilizing MCDM methods. However, the first system lacks the capability to put different priorities on the objectives, requiring a redesign of the optimization module to achieve this functionality. While the second system provides a ranked list of backends based on objectives, it leaves the selection of the best backend for each job to the user. Ultimately, compared to our scheduling system, the critical drawback shared by both works lies in their focus on finding the best backend for a single job at a time. This approach proves impractical and inefficient under the heavy workloads of frequent job submissions, rendering it non-scalable for large-scale quantum cloud scenarios. Furthermore, these works solely consider the user perspective during optimization, neglecting the issue of uneven workload distribution faced by cloud providers. In contrast, our scheduling system addresses the needs of both users and providers. It efficiently schedules jobs under diverse workload conditions and optimizes for both fidelity and waiting time while effectively balancing

the load across available backends.

Beyond the realm of quantum scheduling, another noteworthy contribution lies in the area of quantum cloud analysis. [18] presents a statistical analysis of a dataset exceeding 6,000 job executions. This data, collected by their research group, serves as the foundation for their proposed execution time estimation model. However, compared to our analysis, their approach focuses on their own executions, neglecting the actual workload of the IBM quantum platform. This narrow scope limits the generalizability of their findings. Furthermore, their estimation model utilizes a limited set of features, only analyzing the first circuit in each job. This approach proves insufficient for jobs comprised of diverse circuits, as the initial circuit may not be representative of the entire job. In our attempt to apply the proposed model to predict jobs in the current quantum cloud environment, the results closely resemble the performance of the initial version of our Gate Traversal method presented earlier in Figure 6.1. These results further highlight the limitations of their model compared to the permormance of our proposed regression model.

8 Summary and Conclusion

This project breaks new ground in the rapidly evolving field of quantum computing by presenting, to the best of our knowledge, the first quantum scheduling system capable of true many-to-many scheduling. This approach simultaneously schedules multiple quantum jobs across multiple QPUs, paving the way for a more efficient and user-friendly quantum cloud experience.

Our evaluation demonstrates the multifaceted benefits of this approach. For cloud providers, our system tackles the critical issue of uneven load distribution, ensuring a fairer and more balanced utilization of available resources. Users, on the other hand, benefit from the system's ability to find the best-suited backends for their jobs. This results in significantly reduced waiting times and improved fidelity, enhancing the overall user experience.

Furthermore, our system shows potential for scalability, offering a buffer for future growth in the quantum computing arena. This scalability extends to both the workload volume handled and the growing computational capacities of quantum resources. This adaptability ensures our system's continued relevance and effectiveness as the quantum landscape evolves.

Beyond its immediate practical applications, our work also contributes to the theoretical foundations of quantum scheduling. We present the first formulation of the quantum scheduling problem within the multi-objective optimization domain. This framework opens the path for future exploration of different objective functions and the implementation of more advanced optimization algorithms, further enhancing the efficiency and flexibility of quantum scheduling solutions.

Our work further enriches the quantum cloud domain by providing a dataset of over 7,000 executions on the IBM Quantum platform. This dataset extends beyond execution metadata, encompassing the complete collection of quantum circuits, offering the flexibility to apply it for different tasks.

Finally, we propose two distinct methods for execution time estimation. The Gate Traversal method leverages the latest information from the quantum backend, providing real-time insights, while the Regression Model method utilizes the aforementioned dataset to generate predictions based on past job executions. Our conducted feature analysis sheds light on the key factors influencing execution time estimation. This knowledge may serve as a foundation for developing more advanced and accurate

models in the future.

The full source code of our system, along with the collected dataset, is readily available on GitHub at `https://github.com/Gr1dlock/ScalableQuantumCloudScheduling`.

We predict that the field of quantum scheduling will experience explosive growth in the coming years, mirroring the trajectory of its classical computing counterpart. As the quantum computing landscape continues to evolve, we anticipate automated scheduling solutions to become crucial for both cloud providers and end users, and that more and more quantum providers incorporate them in their access models. We are confident that our system lays the groundwork for a future of efficient and optimized quantum scheduling, paving the way for further advancements and contributions in this promising and rapidly developing field.

9 Future Work

Building upon the foundation laid by our system, we envision a multitude of avenues for future development. This section outlines some key directions that could further expand the functionality and performance of our proposed system, paving the way for even more efficient and effective quantum scheduling solutions.

9.1 Performance Improvements

While our system currently boasts a buffer for handling double the current quantum cloud workload, we believe its scalability potential can be further unleashed.

1. **Parallelizing Execution Time Estimation**

   Despite the existing hardware acceleration in the optimization module, the execution time estimation currently relies on a single CPU core, representing a bottleneck for further performance gains. By leveraging multithreading or multiprocessing, we can significantly improve performance. Distributing feature extraction and prediction steps for each job across multiple threads or processes would lead to a substantial speedup, allowing the system to handle even larger workloads with minimal latency.

2. **Exploring Advanced Machine Learning Frameworks**

   Our current regression model relies on scikit-learn, a powerful but hardware-agnostic library. We believe exploring frameworks like PyTorch [12] or TensorFlow [1], which offer built-in hardware acceleration, could significantly enhance prediction speed.

3. **Proactive Job Preparation**

   Currently, jobs enter the scheduler in their initial state and undergo transpilation and estimation within the scheduler. This can be streamlined by pre-processing jobs for scheduling ahead of time, while they reside in a queue awaiting scheduling. By transpiling jobs and estimating their fidelity and execution time beforehand, we can provide the scheduler with all the necessary information upfront. This reduces the workload during the actual scheduling stage, allowing the system

to focus solely on the optimization process, resulting in faster decision-making and increased system throughput.

9.2 Fidelity Estimation

Our initial approach to fidelity estimation leverages the latest information from the backend, mirroring the Gate Traversal method used for execution time prediction. While this approach holds promise, it remains susceptible to potential inaccuracies in reported QPU properties. Drawing upon our experience with execution time estimation, we believe exploring a regression model-based approach for fidelity estimation could offer greater accuracy and reliability.

This shift aligns with the established success of our regression model in predicting execution times. By leveraging the data contained within our dataset, a regression model could learn complex relationships between job characteristics and fidelities observed on real quantum hardware. This data-driven approach has the potential to provide more accurate and robust fidelity estimations, ultimately leading to improved scheduling decisions.

9.3 Distributed Job Scheduling

Another avenue for advancing quantum scheduling lies in the realm of distributed job execution. Inspired by the EQC system in [22], which draws upon ensemble methods and distributed machine learning principles, this approach offers a potential for improved fidelity and load distribution.

By distributing the execution of a single job across multiple QPUs, each running the same circuits, we can leverage the collective power of these resources to achieve several benefits.

- **Improved Fidelity:** By running the same circuits on multiple backends and aggregating the results, we can potentially mitigate the impact of individual QPU noise and errors, leading to a more reliable and accurate final result.
- **Balanced Backend Load:** Distributing workloads across multiple resources can help alleviate pressure on individual QPUs, promoting a more balanced and efficient utilization of available quantum hardware.

Our current scheduling problem formulation can be readily extended to accommodate this distributed execution paradigm by incorporating the ability to schedule the same job on multiple backends with specified shot numbers.

Abbreviations

NISQ Noisy Intermediate-Scale Quantum

QPU Quantum Processing Unit

OS Operating System

CPU Central Processing Unit

SLA Service Level Agreement

GA Genetic Algoridthm

NSGA Non-dominated Sorting Genetic Algorithm

Bagging Boostrap Aggregating

AdaBoost Adaptive Boosting

DAG Directed Acyclic Graph

SBX Simulated Binary Crossover

MCDM Multi-Criteria Decision-Making

VEQ Variational Quantum Eigensolver

MDI Mean Decrease in Impurity

PFI Permutation-based Feature Importance

List of Figures

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. "Tensorflow: A system for large-scale machine learning." In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, pp. 265–283.

[2] J. Blank and K. Deb. "pymoo: Multi-Objective Optimization in Python." In: *IEEE Access* 8 (2020), pp. 89497–89509.

[3] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. "A (sub) graph isomorphism algorithm for matching large graphs." In: *IEEE transactions on pattern analysis and machine intelligence* 26.10 (2004), pp. 1367–1372.

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

[5] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. "Array programming with NumPy." In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[6] *IBM Quantum*. https://quantum.ibm.com. 2023.

[7] R. Kaewpuang, M. Xu, D. Niyato, H. Yu, Z. Xiong, and J. Kang. "Stochastic Qubit Resource Allocation for Quantum Cloud Computing." In: *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*. IEEE. 2023, pp. 1–5.

[8] D. Lei. "Multi-objective production scheduling: a survey." In: *The International Journal of Advanced Manufacturing Technology* 43 (2009), pp. 926–938.

[9] S. A. Murad, A. J. M. Muzahid, Z. R. M. Azmi, M. I. Hoque, and M. Kowsher. "A review on job scheduling technique in cloud computing and priority rule based intelligent framework." In: *Journal of King Saud University - Computer and Information Sciences* 34.6, Part A (2022), pp. 2309–2331. ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2022.03.027.

[10] P. D. Nation and M. Treinish. "Suppressing Quantum Circuit Errors Due to System Variability." In: *PRX Quantum* 4 (1 Mar. 2023), p. 010327. DOI: 10.1103/PRXQuantum.4.010327.

[11] N. Ngoenriang, M. Xu, S. Supittayapornpong, D. Niyato, H. Yu, et al. "Optimal stochastic resource allocation for distributed quantum computing." In: *arXiv preprint arXiv:2210.02886* (2022).

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[14] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'brien. "A variational eigenvalue solver on a photonic quantum processor." In: *Nature communications* 5.1 (2014), p. 4213.

[15] *Qiskit: An Open-source Framework for Quantum Computing*. https://www.ibm.com/quantum/qiskit. 2023.

[16] N. Quetschlich, L. Burgholzer, and R. Wille. "MQT Bench: Benchmarking Software and Design Automation Tools for Quantum Computing." In: *Quantum* (2023). MQT Bench is available at https://www.cda.cit.tum.de/mqtbench/.

[17] N. Quetschlich, L. Burgholzer, and R. Wille. "MQT Predictor: Automatic Device Selection with Device-Specific Circuit Compilation for Quantum Computing." In: *arXiv preprint arXiv:2310.06889* (2023).

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

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

[20] M. Salm, J. Barzen, F. Leymann, and B. Weder. "Prioritization of compiled quantum circuits for different quantum computers." In: *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. 2022, pp. 1258–1265.

[21] P. W. Shor. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer." In: *SIAM review* 41.2 (1999), pp. 303–332.

[22] S. Stein, N. Wiebe, Y. Ding, P. Bo, K. Kowalski, N. Baker, J. Ang, and A. Li. "EQC: ensembled quantum computing for variational quantum algorithms." In: *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 2022, pp. 59–71.

[23] T. Tomesh, P. Gokhale, V. Omole, G. S. Ravi, K. N. Smith, J. Viszlai, X.-C. Wu, N. Hardavellas, M. R. Martonosi, and F. T. Chong. "Supermarq: A scalable quantum benchmark suite." In: *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE. 2022, pp. 587–603.

[24] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.