Prioritization of Compiled Quantum Circuits for Different Quantum Computers

Marie Salm, Johanna Barzen, Frank Leymann, Benjamin Weder

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{salm, barzen, leymann, weder}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings{Salm2022_PrioritizationCompiledQuantumCircuits,
  Author    = {Marie Salm and Johanna Barzen and Frank Leymann and Benjamin
               Weder},
  Title     = {{Prioritization of Compiled Quantum Circuits for Different
               Quantum Computers}},
  Booktitle = {Proceedings of the 2022 IEEE International Conference on
               Software Analysis, Evolution and Reengineering (SANER 2022)},
  Pages     = {1258--1265},
  Publisher = {IEEE},
  Year      = 2022,
  Doi       = {10.1109/SANER53432.2022.00150}
}
```

Prioritization of Compiled Quantum Circuits for Different Quantum Computers

Marie Salm, Johanna Barzen, Frank Leymann, Benjamin Weder
Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart, Germany
{salm, barzen, leymann, weder}@iaas.uni-stuttgart.de

***Abstract*—Quantum computers can solve certain problems faster than classical computers. The number of quantum computers offered via the cloud increases such that a variety is accessible for the public. However, current quantum computers still suffer from a small number of qubits and high error rates. Selecting a quantum computer providing enough qubits and low error rates is tremendously important to receive stable execution results for a given quantum circuit. The execution results are also influenced by the compilation result of the selected quantum compiler for hardware mapping and optimization. For computing time, the access is regulated by vendors of quantum computers resulting in waiting times for the user. Thus, a deep and manifold analysis of quantum computers and compiled circuits of different quantum compilers is required to estimate in advance which compiled circuit will return stable execution results also in consideration of non-functional requirements such as waiting times. To address this, we present a framework that analyzes and prioritizes compiled circuits for different quantum computers using several compilers based on the requirements of the user. We show the practical feasibility of our framework by a prototype and two use cases.**

***Index Terms*—quantum computing, compilation, prioritization, MCDA, decision support, NISQ Analyzer**

I. Introduction

Quantum computers are known to solve specific problems faster than classical computers [1], [2]. However, applying quantum algorithms with up to an exponential speedup, such as the Shor algorithm [3], on real complex problems are not yet feasible: Also, the next years will be dominated by the *Noisy Intermediate-Scale Quantum (NISQ)* era [4]. It represents current gate-based quantum computers with high error rates and few qubits limiting the size of executable quantum circuits, i.e., implemented quantum algorithms [4], [5].

Nevertheless, the number of quantum computers grows continuously such that already a variety is publicly accessible via the cloud [6], [7]. They differ in their hardware characteristics, such as the number of qubits, qubit connectivity, gate set, and error rates. These characteristics influence whether a circuit is executable and, if this is the case, directly affect the precision of the execution result [8], [9]. To execute a given circuit, the selection of a quantum computer offering a sufficient number of qubits and low error rates is hampered by the NISQ era [9]. But it is therefore tremendously important to obtain stable execution results in terms of executing contained error-prone gates within the decoherence times of the qubits.

However, always selecting the same quantum computer that has led to successful executions in the past is not advisable: (i) Quantum computers are regularly re-calibrated causing varying error rates [10], [11]. (ii) Before executing a circuit, it has to be mapped, i.e., compiled to the quantum computer [5], [12]. Based, e.g., on qubit connectivity and implemented gate set, the compilation may result in a circuit requiring more qubits and gates and, thus, causing a higher overall error rate [8]. (iii) Furthermore, existing quantum compilers differ in their mapping algorithms resulting in different compilation results for the same circuit and quantum computer as the mapping is NP-hard and can only be solved approximately [12], [13]. Multiple compilers should therefore be considered. (iv) Vendors offering quantum computers via the cloud regulate the access for computing time. Thereby, several access methods exist, e.g., job queues or time slices [6], [14]. These may cause varying waiting times for different quantum computers until executing a circuit. Overall, a deep and versatile analysis of different quantum computers, their characteristics, and their access regulation via the cloud is required. Related compilation results from several compilers of the circuit to be executed should also be considered to estimate in advance which compiled circuits will result in stable execution results within short waiting times and prioritize them according to the needs of the user. Our first research question (RQ) is therefore as follows:

RQ 1: What are relevant metrics for the analysis and prioritization of compilation results of a quantum circuit for different quantum computers?

To address **RQ 1**, several quantum software development kits (SDKs) and vendors of quantum computers are analyzed to identify metrics describing properties of compiled circuits, up-to-date quantum computer characteristics, and non-functional requirements such as waiting times influencing the execution and to serve as a basis for prioritization.

However, retrieving all these metrics requires a lot of manual effort. Additionally, interpreting them to prioritize compilation results based on prospects of stable and fast available execution results requires immense knowledge about quantum computing. The individual needs of the user must thereby be the focus. Our second RQ is accordingly the following:

RQ 2: How can compiled circuits of a given quantum circuit for different quantum computers be automatically prioritized based on requirements of the user?

To tackle **RQ 2**, we present a framework that automatically prioritizes compilation results of a given quantum circuit for different quantum computers based on a set of metrics whose importance is stated individually by the user. Besides automated compilation with multiple compilers [12], the framework enables to further *(i) analyze* compiled circuits and quantum computers and *(ii) prioritize* them according to metrics identified by answering **RQ 1**.

For prioritization, well-known *(i) multi-criteria decision analysis (MCDA) methods* are applied. *(ii) Use cases* are presented performing experiments to retrieve initial weights for the metrics accordingly. Additionally, the *(iii) user is enabled* to adjust weights according to own needs. We present a *(iv) prototype* to demonstrate the feasibility of this framework. The prototype is a plug-in-based system; thus, further metrics and prioritization methods can be included.

The paper is structured as follows: Sect. II introduces fundamentals about MCDA. In Sect. III, SDKs and vendors are analyzed to identify metrics. Sect. IV presents our prioritization approach. Sect. V shows the system architecture and prototype. Sect. VI presents two use cases. Sect. VII discusses current limitations. Sect. VIII presents related work. Sect. IX concludes the paper and shows future work.

II. Multi-Criteria Decision Analysis

In several domains such as Cloud Computing [15], finance [16], and logistics [17], MCDA methods support the decision-maker to select an alternative out of a set of given alternatives based on individual needs [18]. Multiple criteria thereby describe properties of alternatives targeting one of the defined goals. The significance of single criteria can be regulated by defining individual weights [18]. When mapping the concepts to our approach, the compiled circuits jointly with associated quantum computers can be understood as the alternatives and our metrics describe the criteria. A goal of the user might be the fast execution of a circuit described by metrics about waiting time. The potentially conflicting goals are stable execution results considered by the metrics about hardware characteristics and circuit properties [19].

Today, a huge number of MCDA methods exist targeting different aspects in the multi-criteria analysis field, e.g., by returning a single alternative or a ranked list [18]. We therefore use the decision-support system *MCDA Method Selection Tool* [18], [20] to select suitable methods for our approach. The system relies on different criteria, e.g., consideration of weights, the scale of metric values, and the type of the given decision problem such as single choice or ranking. As the impact of the different metrics is to be examined and the user should be able to reflect their needs, weights should be supported by the used MCDA methods. The values of our metrics are quantitatively compared, not, e.g., mapped to a common scale. Our decision problem is a ranking of compiled circuits and not a single choice or classification problem. The considered MCDA methods can either provide a complete or a partial ranking. Based on the given criteria, 30 MCDA methods are returned. As a first attempt, we support three of them: *Technique for Order Preference by Similarity to Ideal Solution (TOPSIS)* [21], *ELimination Et Choice Translating REality III (ELECTRE III)* [22], and *Preference Ranking Organization METHod for Enrichment of Evaluations II (PROMETHEE II)* [23]. All three support quantitative weights simplifying their handling and are among the most common MCDA methods [18], [24]. Due to the expandability of our framework, further MCDA methods can be added.

TOPSIS ranks alternatives according to their distance to the optimal alternative providing the best metric values regarding all alternatives and to the anti-optimal alternative providing the worst values [25]. All metric values and weights are normalized to enable a comparison of metrics. The optimal and anti-optimal alternatives are determined and the Euclidean distances between them and given alternatives is calculated considering defined weights. Regarding both distances, the relative distance of an alternative to the optimum is determined. The ranking is the descending order of relative values.

PROMETHEE, in general, compares alternatives pairwise for each metric [26]. Thresholds determine which of two regarded alternatives is preferred by defining the minimum distance for each metric. The *outranking* over all metrics between two alternatives is calculated by summing the products of previously determined preference distances and related metric weights over all metrics and normalizing it [19]. Eventually, it is calculated how far an alternative is more preferred than the others, called *positive flow* [26]. On the opposite, it is calculated how far the other alternatives are more preferred than the regarded one, called *negative flow*. In PROMETHEE II, the negative flow is subtracted from the positive flow resulting in a *net flow*, i.e., a total order of alternatives [19].

ELECTRE III also compares alternatives pairwise and uses thresholds to determine indifference and preference of two alternatives [27]. The *concordance index* defines how much an alternative outranks another for each metric based on given thresholds. Veto thresholds enable the rejection of outrankings if metric values of weaker alternatives are better than the related veto thresholds. The *discordance index* defines the opposite using the veto thresholds and is compared with the concordance index. To get a partial ranking, alternatives are ordered ascending and descending based on their outranking degree and both orderings are intersected [22], [27].

III. Identification of Quantum Computer and Quantum Circuit Metrics

Objectives in executing a quantum circuit on a quantum computer are, e.g., to obtain stable execution results and, additionally, to have only a short waiting time until execution. In this case, we need to consider the compilation result of a circuit with its properties as well as the related quantum computer with its hardware characteristics and access regulations [5]. Existing SDKs and vendors already provide methods to analyze circuits and supported quantum computers. To address **RQ 1**, we summarize metrics introduced by [6], [11], [14] and extend them by analyzing four SDKs and three vendors to describe properties of compiled circuits and quantum computers influencing executions and waiting times: Qiskit [28] with the vendor IBMQ, Cirq [29] with Google, Forest [30] with Rigetti, and the vendor-independent SDK pytket [31] supporting all mentioned vendors. The identified set of metrics is expandable. Further SDK analysis is presented in [32]. While most circuit metrics can be defined programmatically if not natively supported as methods, metrics for quantum computers are limited by the information provided by the vendor. Defined metrics about hardware characteristics are therefore offered by all quantum computers of considered vendors. We describe the identified metrics for circuits and quantum computers hereafter.

A. *Quantum Circuit Metrics*

A metric describing the number of required qubits of a circuit is the *(i) width* [5], [11]. The target quantum computer must provide at least as many qubits as the compiled circuit [9]. If the compiler can decrease the number of qubits, the mapping to the topology of the quantum computer may be optimized reducing errors caused by qubits, qubit connections, and SWAP gates [8], [10], [31]. *(ii) Depth* defines the number of sequentially executable gates [5]. All gates should be applied before the qubits of the quantum computer decohere [5], [8]. Furthermore, errors of sequentially applied gates accumulate, amplifying the resulting error. As multi-qubit gates have the highest gate errors, *(iii) multi-qubit gate depth* describes the number of sequentially executable multi-qubit gates in a circuit [31]. Each operation introduces errors, thus, *(iv) total number of operations* represents the overall size of a circuit considering gates and measurement operations [11]. *(v) Number of single-qubit gates* counts the number of single-qubit gates whereas *(vi) number of multi-qubit gates* counts the error-prone multi-qubit gates in a circuit [31]. *(vii) Number of measurement operations* counts all contained measurement operations.

B. *Quantum Computer Metrics*

The success of circuit executions relies on the hardware characteristics of quantum computers. *(viii) Average single-qubit gate error* defines the average error rates of all single-qubit gates as well as *(ix) average multi-qubit gate error* for all multi-qubit gates of a given quantum computer [31]. Besides gate errors, also gate times must be considered since they use the decoherence times of targeting qubits [8], [11]. Thus, *(x) average single-qubit gate time* and *(xi) average multi-qubit gate time* are introduced to represent the mean value of gate times of single-qubit and multi-qubit gates. The *(xii) average readout error* defines the average error of all measurement operations of a quantum computer causing long delays and introducing further errors [5], [8]. The decoherence time consists of T1 measuring the time it takes for a qubit to relax from state $|1\rangle$ to $|0\rangle$ and T2 measuring the time it takes for a qubit to exhibit smaller phase errors [10], [11]. The longer these times are, the higher are the chances of stable execution results. *(xiii) Average T1* and *(xiv) average T2* define the decoherence times of all qubits of a quantum computer. To target short waiting times until accessing a quantum computer for execution, *(xv) waiting time* is introduced. The access methods [33] defined by vendors of quantum computers are merged. It represents, e.g., queue sizes, wait time until free time slices, or scheduled reservations.

IV. Prioritization of Compiled Quantum Circuits

We address **RQ 2** by presenting a framework to prioritize compiled circuits of a given circuit for different quantum computers using several quantum compilers. Fig. 1 gives an overview of our approach. Light components are from previous work [9], [11], [12], shaded components are extended, and dark components are new components introduced in this work.

A. *Translation*

Present SDKs providing quantum compilers to enable the compilation of quantum circuits differ in their supported programming languages and gate sets [6], [12]. Thus, in the *(A) Translation* phase, if the given circuit of the user is written in a programming language not supported by the SDK of one of the supported compilers, it is automatically translated into the required language and gate set [12].

B. *Compilation*

The circuits in the appropriate languages for the different SDKs are passed to the *(B) Compilation* phase. Supported compilers, e.g., t$|$ket$\rangle$ [31] and Qiskit Transpiler [28], are selected for compilation if their SDKs natively support access to the individual available quantum computers [12]. The provenance system QProv [11] is therefore used to provide uniform up-to-date information such as availabilities, error rates, and gate sets about the quantum computers of supported vendors. The selected compilers compile the given circuits on all supported quantum computers and simulators [12].

C. *Circuit and QPU Analysis*

In the *(C) Circuit and QPU Analysis* phase, compiled circuits and related quantum computers are analyzed to determine values of metrics identified in Sect. III. Recent values for quantum computer metrics, e.g., average gate and readout errors are retrieved [11]. The executability of compiled circuits on related quantum computers is examined and circuits are filtered accordingly [9], [12]: The depth of a compiled circuit is compared with the quotient of average decoherence times of all qubits divided by the maximum gate time of the quantum computer [34]. If quantum computers do not provide enough qubits, compilers automatically cause compilations to fail, reducing the set of compiled circuits for the next phase [12].

D. *Prioritization*

The determined metric values of executable compilation results and quantum computers are handed over to the *(D) Prioritization* phase where the compiled circuits are prioritized. To calculate a ranking, the user selects one of the supported MCDA methods and adjust the weights of the metrics if needed according to their requirements. The method SMART [35] is provided to simplify the appropriate selection of weighting values: The user assigns between 0 and 100 points to each metric, with 0 being the lowest and 100 being the highest

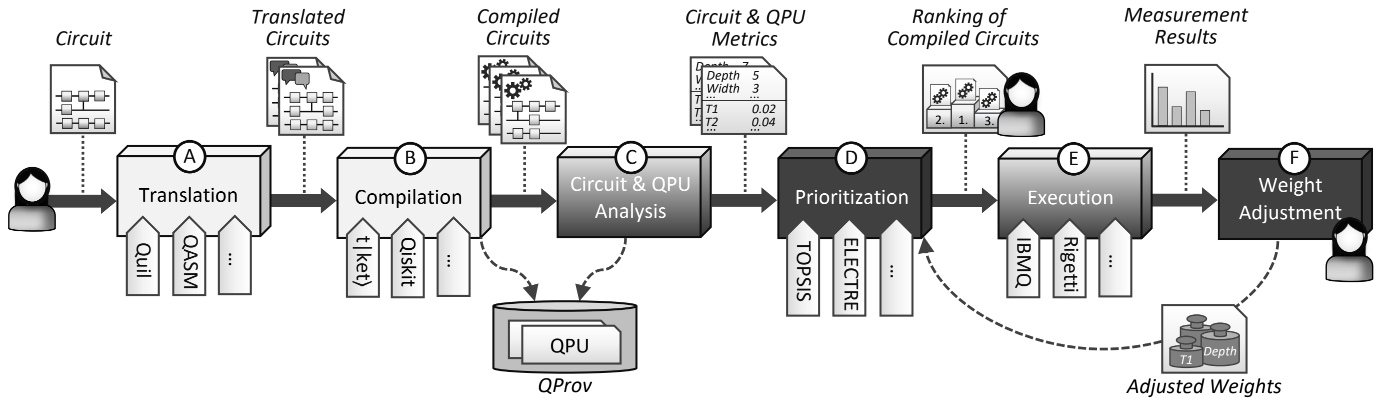

Fig. 1. Automated compilation, analysis, and prioritization of compiled circuits for available quantum computers using multiple quantum compilers.

number of points [19]. Metrics with 0 points are not considered for ranking, thus, different metric sets and MCDA methods can be combined. For example, if fast execution is the most important aspect, *(xv) waiting time* receives 100 points, all other metrics have accordingly fewer points. Its weight is calculated by dividing the 100 points by the sum of all points. Alternatively, TOPSIS [21] with initial values for metric weights is applied. A ranked list of compiled circuits and their related quantum computers is finally presented to the user.

E. Execution

The user selects which of the prioritized executable compilation results to execute. In the *(E) Execution* phase, the selected circuit is automatically delivered to, e.g., IBMQ or Rigetti providing the target quantum computer by using the SDK of the related quantum compiler [12]. Besides the execution on the quantum computer, a proper compiled circuit is executed on an available simulator if it fits the resource requirements of the circuit. Afterward, the returned measurement results of the execution are shown to the user. To evaluate the quality of the execution results via a single value, histogram intersection [36] is applied to the execution results as they are commonly represented as histograms. The histograms of the simulator and the quantum computer are therefore overlaid and compared to determine the deviation due to errors caused by the quantum computer. The resulting deviation value is shown to the user.

F. Weight Adjustment

The results are input for the optional *(F) Weight Adjustment* phase. The user can compare the results and histogram intersection values of the compiled circuits. Based on their requirements and analysis of the results, the user can adjust the metric weights from Sect. IV-D. With the adjusted weights, the user can initiate a re-prioritization of the list of compiled circuits (Sect. IV-D) but can also start the compilation and prioritization of another circuit based on these weights (Sect. IV-A).

V. System Architecture and Prototype

In this section, we describe the system architecture of the prioritization approach, shown in Sect. IV. Further, its prototypical implementation is presented.

A. System Architecture

Fig. 2 presents the overall system architecture of our framework. We therefore extended the *NISQ Analyzer* [9], [12] by additional components and external services to prioritize compiled quantum circuits. Light components are from previous work, middle gray components are expanded, and dark components are new. Further, external components which are integrated are marked with dashed lines. The *Translator* and its *Translator UI* translates given quantum circuits in required languages based on stored *Gate Mappings* [12]. To prioritize, external *MCDA Services* are invoked providing the execution of MCDA methods described in Sect. II offered via the cloud. To invoke required MCDA Services, the *Connector* of the NISQ Analyzer is extended. The *NISQ Analyzer UI* and the HTTP REST API of the NISQ Analyzer are extended to prioritize and analyze compilation results in Sect. III. With the additional metrics and the histogram intersection values for compiled quantum circuits, the model of stored compilation and execution *Results* contain further attributes. The metrics as well as their weights are stored in a repository. The *Performance Extractor* collects and extracts the metric values of compilation results and supported quantum computers required for the application of MCDA Services. The *Prioritizer* invokes the MCDA Services, interprets the returned results, and stores the rankings of compiled circuits. Furthermore, it supports the calculation of histogram intersections as described in Sect. IV. On the right side of Fig. 2, the SDK Services *Forest Service*, *pytket Service*, and *Qiskit Service* as well as the provenance system QProv [11] are extended to support the analysis of compiled circuits and quantum computers based on the additional metrics [12].

To prioritize compiled circuits of a given quantum circuit for different quantum computers using multiple compilers, the user starts the compilation via the NISQ Analyzer UI, as presented in [12]. All quantum computers and at least one simulator provided via QProv are therefore considered. The compilation on a simulator is required for the later histogram intersection. A detailed description of the translation and compilation process is presented in [12]. After compilation, the SDK Services analyze the compiled circuits based on the defined set of

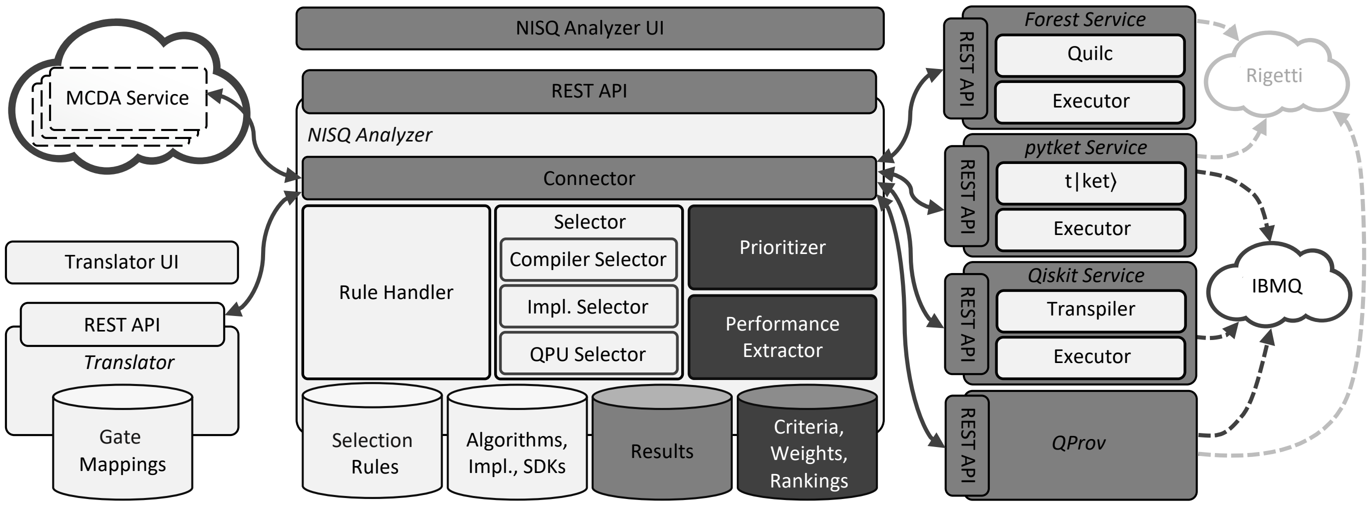

Fig. 2. System architecture to prioritize compiled quantum circuits for different quantum computers using multiple quantum compilers. Adapted from [12].

metrics and return the results to the NISQ Analyzer. The Selector proofs their executability [9], stores them into the repository, and they are presented to the user. With the NISQ Analyzer UI, the user can select which MCDA method to apply on the non-ranked list of compiled circuits. The user can adjust given weights of enabled metrics stored in the repository based on the SMART method [35], see Sect. IV. The Prioritizer then calls the Performance Extractor to extract metric values of compilation results for prioritization. The Performance Extractor also invokes QProv via the Connector to gain metric values of the considered quantum computers. Metrics, their values, weights, and the compilation results are returned to the Prioritizer and are transferred to the MCDA Service of the selected MCDA method over the Connector. Eventually, the resulted ranking of compiled circuits is returned to the NISQ Analyzer. The Prioritizer interprets the ranking, orders the list of compilation results, and stores the ranking. The ranked list of compiled circuits is presented via the NISQ Analyzer UI. The user can select compiled circuits to execute them [9]. The SDK Service that previously compiled the selected circuit is therefore invoked [12]. Besides the selected compilation result of the quantum computer, also the compilation result matching an available simulator is passed to the Executor. The execution results of the quantum computer and simulator are returned to the NISQ Analyzer. Their histogram intersection value is calculated, stored, and presented to the user if the simulator offered enough resources to compute its compilation result. The user can analyze the results and, again, adjust the weights of defined metrics for further prioritization.

The framework is plug-in based, thus, further MCDA Services, i.e., MCDA methods, SDK Services, programming languages, and metrics can be added.

B. Prototype

The NISQ Analyzer and QProv are implemented in Java using the framework Spring Boot. The UIs are written in TypeScript using Angular. For the Translator and the SDK Services, the Python framework Flask is used. For further details about their prototypical implementations, see [9], [12]. The framework is available open-source [37].

To enable the prioritization of compiled quantum circuits based on MCDA methods, Decision Deck [38] is integrated. It is a project to develop and provide open-source software in the context of MCDA [38]. Combinable parts or entire MCDA methods are thereby offered as web services that are callable by the NISQ Analyzer via SOAP. Supported MCDA methods so far are based on sample workflows of several web services, see [39]–[41]. After each web service, the NISQ Analyzer retrieves the returned results and, if the call of another web service is foreseen in the workflow, forwards them to the next web service. Processed data is described in XMCDA, an XML standard tailored for MCDA [38]. The NISQ Analyzer has to transform metric values, weights, and IDs of compiled quantum circuits into the required format. Returned final ranking results of web services in XMCDA have to be parsed to prioritize compiled quantum circuits accordingly.

VI. Case Study

In this section, we present the usage of our framework introduced in Sect. V. We therefore present two use cases based on three circuits of different quantum algorithms provided in our GitHub repository [37]: The circuit *Grover-SAT* computes the *Boolean satisfiability problem* of the Boolean formula $(A \wedge B)$ based on the Grover algorithm [42]. *Shor-15* defines the circuit to calculate the prime factors of the number 15 based on the Shor algorithm [3]. *BV-00110* computes the secret string 00110 based on the Bernstein-Vazirani algorithm [43]. All three circuits are compiled and executed on the IBMQ quantum computers *ibmq_lima* [44], *ibmq_manila* [45], and *ibmq_quito* [46] with 8192 shots using the Qiskit Transpiler [28] via Qiskit Service and the t$|$ket$\rangle$ compiler [31] via pytket Service. *(xv) Waiting time* is defined by the queue size of a given quantum computer. The three circuits are also compiled and executed on the *ibmq_qasm_simulator* [47] for

TABLE I
RANKINGS OF COMPILED CIRCUITS BASED ON IDENTIFIED METRICS

	QPU	SDK	Depth	Multi-Q Depth	Num. Ops	Num. Multi-Q	Multi-Q Error	Multi-Q Time (ns)	Readout Error	T1 (ms)	Queue	Histo. Inter.	T-A	P-A	T-B	P-B
Grover-SAT	Sim.	Qiskit	14	6	21	6	0	0	0	∞	2	1	1	1	1	1
	lima	pytket	27	9	45	9	0.010	387	0.0279	94	2	0.874	3	3	2	4
	manila	pytket	27	9	45	9	0.008	350	0.0276	196	5	0.847	2	2	6	3
	quito	pytket	27	9	45	9	0.025	277	0.0563	101	1	0.838	6	4	3	2
	lima	Qiskit	44	13	64	13	0.010	387	0.0279	94	2	0.824	5	7	4	7
	manila	Qiskit	30	13	48	13	0.008	350	0.0276	196	5	0.811	4	5	7	5
	quito	Qiskit	42	13	61	13	0.025	277	0.0563	101	1	0.757	7	6	5	6
W.-A			0.10	0.10	0.10	0.11	0.06	0.05	0.06	0.04	0					
W.-B			0.08	0.09	0.08	0.10	0.05	0.04	0.05	0.03	0.12					

the histogram intersection using the Qiskit Service. In total, 21 compiled circuits are provided for the first approach to define initial weights. For the use cases, the MCDA methods TOPSIS, PROMETHEE II, and ELECTRE III from Sect. II are applied on the set of compiled circuits of each quantum algorithm circuit. Because of space constraints, in Table I, we only present a subset of all metrics and rankings of TOPSIS and PROMETHEE II for *Grover-SAT*. The compiled circuits are ordered by the respective histogram intersection values. The entire table containing all compiled circuits, metrics, weights, and rankings based on ELECTRE III is shown in [48].

Our first use case focuses on finding an initial weighting based on the quality of execution results. For this use case, *(xv) waiting time* is not considered. For each circuit, we focus on the histogram intersection results shown in Table I and compare the metric values to identify why specific compilation results produce better execution results than others. Based on detected correlations between histogram intersection values and metric values, weights are determined. First, for each quantum algorithm circuit, we propose to determine pre-weighting values between 0 and 100 for each metric: If a matching ordering of distances to the optimal metric value can clearly be identified, the pre-weight 100 is set for this metric. For example, considering histogram intersection values in Table I, the compiled circuits of *Grover-SAT* show an ascending order for *(vi) number of multi-qubit gates* which should be minimal. Note that *(vi) number of multi-qubit gates* and *(iii) multi-qubit gate depth* may also differ for other circuits. If no total ordering is presented for a metric, but more optimal metric values are mostly part of more optimal compilation results and vice versa, 80 is defined, e.g., for *(ii) depth* at *Grover-SAT*. If no relation between a metric and the histogram intersection values can be clearly detected, 50 is set as pre-weight, e.g., at *(ix) average multi-qubit gate error*. At *Grover-SAT*, *(xiii) average T1* is set to 20 as the best metric values are not part of the compiled circuits with the highest histogram intersection values, however, they are also not part of the worst once. Finally, if the worst metric values are related to compilation results with high ranks, 0 is set. For example, for *Grover-SAT* the highest *(xi) average multi-qubit-gate time* values are part of compiled circuits with the highest histogram intersection value. If pre-weights for each metric for all three quantum algorithm circuits are set, they are divided by three and then, the SMART method presented in Sect. IV is applied to retrieve initial weights. Other procedures regarding pre-weighting can also be applied. The resulted weights (W.-A) for the first use case are presented in Table I. These are passed to the MCDA methods to execute them on the compilation results for each circuit. The rankings of TOPSIS (T-A) and PROMETHEE II (P-A) for *Grover-SAT* and *Shor-15* reflect the histogram intersection values better than for *BV-00110*. In Table I, the rankings T-A and P-A present a rough trend compared to the histogram intersection values. The rankings between the MCDA methods differ due to their different approaches [17]. Regarding *Grover-SAT* and *Shor-15*, the t$|$ket$\rangle$ compiler results in higher histogram intersection values than the Qiskit Transpiler. The characteristics of quantum computers do not seem to have much influence. Instead, for *BV-00110* the properties of the different compilation results are similar such that the characteristics of the quantum computers seem to influence the execution results.

In the second use case, also the *waiting time* is considered by setting its pre-weighting value to 100. Previously defined pre-weights remain unchanged. The resulting weights (W.-B) are shown in Table I. The ranks of TOPSIS (T-B) and PROMETHEE II (P-B) in Table I show that *ibmq_manila* gets, in general, a lower rank than in the previous use case as it has the highest queue size. On the other hand, the rankings considering *ibmq_quito* improved because this quantum computer has the smallest queue size.

VII. Discussion and Limitations

With the introduced framework, the user is enabled to further analyze and prioritize compiled circuits of a given quantum circuit for different quantum computers using multiple quantum compilers. With initial weights defined in Sect. VI, the histogram intersection results are roughly reflected but are not enabling a precise ranking of compilation results of other quantum circuits. However, the framework enables the user to prioritize compiled circuits based on their needs. In the future, we want to consider a larger set of quantum circuits and, especially, want to include an automated approach to learn weights based on past executions. Often, to further evaluate the stability of resulting rankings of MCDA methods, sensitivity analyzes are applied [19]. As the focus of this paper is the collection of metrics and the support of different MCDA methods, we will provide such sensitivity analyzes in the future. The applicability of histogram intersection based on simulators as an indicator of stable execution results is only feasible as long as the quantum calculation can be simulated [49]. However, also other methods for evaluation can be added, e.g., *Probability of Successful Trials (PST)* [10], as the framework is plug-in-based. Monetary metrics, e.g., execution costs, are currently not covered in our set of metrics as we accessed free accessible quantum computers. Nevertheless, the framework can be extended to support further metrics. We only consider gate-based quantum computers, but we also want to analyze metrics for, e.g., photonic quantum computers in the future [2].

VIII. Related Work

MCDA is applied in a variety of areas, such as finance [16] and logistics [17]. In the area of Cloud Computing, e.g., Garg et al. [15] present a framework that applies an MCDA method to rank different cloud services based on several quality-of-service attributes. Nevertheless, cloud offerings for quantum computers are not considered.

The work of Ravi et al. [50] proposes a framework to schedule execution jobs to multiple quantum computers in the cloud. Metrics regarding the fidelities of quantum computers, compiled circuit properties, and waiting times in their related queues are therefore considered. They introduce prediction models to predict waiting times and learn correlations between different metrics. To select the optimal quantum computer regarding waiting time and fidelity, a utility function is used. However, the user is not enabled to prioritize supported metrics based on their needs. Furthermore, the considered quantum computers and their queue sizes are simulated, and no prototypical implementation is provided.

In [51], Ravi et al. present a study similar to the previously discussed paper [50]. They make theoretical recommendations considering, e.g., the importance of two-qubit gates, prediction of waiting times in queues, and optimized compilation strategies observing past circuit executions [51]. A model to predict execution times on quantum computers is introduced based on metrics such as width, depth, number of gates, and number of qubits. The model is then applied on several quantum computers. Their work gives an overview of current and future issues regarding quantum computers and their usability in the cloud, but no prototypical implementation is presented.

Grossi et al. [52] introduce an architecture that integrates the API of quantum computer vendors into a classical enterprise architecture. To execute quantum circuits on quantum hardware, the quantum computer offering the required number of qubits and the shortest queue length is selected. However, the requirements of the user are not considered by, e.g., prioritization, and no hardware characteristics such as decoherence times are taken into account. Besides the number of qubits, no further properties of quantum circuits are observed.

Cruz-Lemus et al. [53] propose a set of different metrics to analyze quantum circuits. They consider counts and ratios of single-qubit gates, multi-qubit gates, measurement gates, ancillae, and oracles as well as depth, width, and the number of parallel gates. The proposed metrics still need to be implemented and validated by experiments. As our framework is expandable, further metrics can be added.

The well-known Quantum Volume [54] considering the size of a circuit and the error rate of a quantum computer enables the comparison of performances between different quantum computers [8]. However, not all vendors calculate and provide the resulting performance values of available quantum computers and Quantum Volume only enables the ranking of quantum computers, not compiled circuits which are considered by the approach of Wack et al. [55].

IX. Conclusion and Future Work

In this work, we presented a framework to prioritize compilation results of a given quantum circuit for different quantum computers based on estimations of execution results and non-functional requirements. Especially, compilation results of several quantum compilers are considered. Besides automated compilation, the framework supports further (i) analysis of metrics of quantum computers and compiled circuits, as well as (ii) prioritization of compiled circuits according to the needs of the user answering **RQ 2**. The user is enabled to determine the weights of individual metrics and select the MCDA method to prioritize compiled circuits. Initial metric weights were determined by two presented use cases to further support the user. Answering **RQ 1**, SDKs and vendors of quantum computers were analyzed to identify metrics describing the properties of quantum computers and compiled circuits, building the basis of our prioritization framework.

In the future, we plan to support further metrics describing the hardware characteristics of quantum computers and the properties of compiled quantum circuits. For example, we plan to support PST [10] to further improve our analysis. To support additional SDKs, compilers, and vendors for prioritization, we want to add more SDK Services. We also want to improve the initial metric weights and further support the user selecting compiled quantum circuits promising stable execution results by automatically learning weights and choosing suitable MCDA methods based on past executions using Machine Learning [56]. We thereby plan to provide sensitivity analyzes [19] to evaluate the stability of resulted rankings.

Acknowledgement

This work was partially funded by the BMWi project *PlanQK* (01MK20005N).

References

[1] F. Arute *et al.*, “Quantum supremacy using a programmable superconducting processor,” *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.
[2] H.-S. Zhong *et al.*, “Quantum computational advantage using photons,” *Science*, 2020.
[3] P. W. Shor, “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer,” *SIAM Journal on Computing*, vol. 26, no. 5, p. 1484–1509, 1997.
[4] J. Preskill, “Quantum Computing in the NISQ era and beyond,” *Quantum*, vol. 2, p. 79, 2018.
[5] F. Leymann and J. Barzen, “The bitter truth about gate-based quantum algorithms in the NISQ era,” *Quantum Science and Technology*, vol. 5, no. 4, pp. 1–28, 2020.
[6] R. LaRose, “Overview and Comparison of Gate Level Quantum Software Platforms,” *Quantum*, vol. 3, p. 130, 2019.

[7] F. Leymann *et al.*, "Quantum in the Cloud: Application Potentials and Research Opportunities," in *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020)*. SciTePress, 2020, pp. 9–24.

[8] M. Salm, J. Barzen, F. Leymann, and B. Weder, "About a Criterion of Successfully Executing a Circuit in the NISQ Era: What $wd \ll 1/\epsilon_{\mathrm{eff}}$ Really Means," in *Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software (APEQS 2020)*. ACM, 2020, Workshop, pp. 10–13.

[9] M. Salm *et al.*, "The NISQ Analyzer: Automating the Selection of Quantum Computers for Quantum Algorithms," in *Proceedings of the 14th Symposium and Summer School on Service-Oriented Computing (SummerSOC 2020)*. Springer International Publishing, 2020, pp. 66–85.

[10] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: A case for variability-aware policies for nisq-era quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. ACM, 2019, p. 987–999.

[11] B. Weder, J. Barzen, F. Leymann, M. Salm, and K. Wild, "QProv: A provenance system for quantum computing," *IET QuantumCommunication*, vol. 2, no. 4, pp. 171–181, Dec. 2021.

[12] M. Salm, J. Barzen, F. Leymann, B. Weder, and K. Wild, "Automating the Comparison of Quantum Compilers for Quantum Circuits," in *Proceedings of the 15th Symposium and Summer School on Service-Oriented Computing (SummerSOC 2021)*. Springer International Publishing, Sep. 2021, pp. 64–80.

[13] A. Cowtan *et al.*, "On the Qubit Routing Problem," in *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 135. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, pp. 5:1–5:32.

[14] D. Vietz, J. Barzen, F. Leymann, and K. Wild, "On Decision Support for Quantum Application Developers: Categorization, Comparison, and Analysis of Existing Technologies," in *Computational Science – ICCS 2021*. Springer International Publishing, 2021, Workshop, pp. 127–141.

[15] S. K. Garg, S. Versteeg, and R. Buyya, "SMICloud: A Framework for Comparing and Ranking Cloud Services," in *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, 2011, pp. 210–218.

[16] D. Alper and C. Başdar, "A Comparison of TOPSIS and ELECTRE Methods: an Application on the Factoring Industry," *Business and Economics Research Journal*, vol. 8, no. 3, p. 627, 2017.

[17] M. Velasquez and P. T. Hester, "An analysis of multi-criteria decision making methods," *International journal of operations research*, vol. 10, no. 2, pp. 56–66, 2013.

[18] J. Wątróbski, J. Jankowski, P. Ziemba, A. Karczmarczyk, and M. Zioło, "Generalised framework for multi-criteria method selection," *Omega*, vol. 86, pp. 107–124, 2019.

[19] J. Geldermann and N. Lerche, "Leitfaden zur Anwendung von Methoden der multikriteriellen Entscheidungsunterstützung," *Methode: Promethee*, 2014.

[20] J. Wątróbski, J. Jankowski, P. Ziemba, A. Karczmarczyk, and M. Zioło. (2021) MCDA Method Selection Tool. [Online]. Available: http://mcda.it

[21] C.-L. Hwang and K. Yoon, *Methods for Multiple Attribute Decision Making*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1981, pp. 58–191.

[22] J. R. Figueira, V. Mousseau, and B. Roy, *ELECTRE Methods*. New York, NY: Springer New York, 2016, pp. 155–185.

[23] J.-P. Brans and B. Mareschal, *Promethee Methods*. New York, NY: Springer New York, 2005, pp. 163–186.

[24] Sałabun, Wojciech and Wątróbski, Jarosław and Shekhovtsov, Andrii, "Are MCDA Methods Benchmarkable? A Comparative Study of TOPSIS, VIKOR, COPRAS, and PROMETHEE II Methods," *Symmetry*, vol. 12, no. 9, 2020.

[25] A. Bilbao-Terol, M. Arenas-Parra, V. Cañal-Fernández, and J. Antomil-Ibias, "Using TOPSIS for assessing the sustainability of government bond funds," *Omega*, vol. 49, pp. 1–17, 2014.

[26] S. Corrente, S. Greco, and R. Słowiński, "Multiple Criteria Hierarchy Process with ELECTRE and PROMETHEE," *Omega*, vol. 41, no. 5, pp. 820–846, 2013.

[27] S. S. Hashemi, S. H. R. Hajiagha, E. K. Zavadskas, and H. A. Mahdiraji, "Multicriteria group decision making with ELECTRE III method based on interval-valued intuitionistic fuzzy information," *Applied Mathematical Modelling*, vol. 40, no. 2, pp. 1554–1564, 2016.

[28] G. Aleksandrowicz *et al.*, "Qiskit: An Open-source Framework for Quantum Computing," 2019.

[29] Quantum AI team and collaborators, "Cirq," 2020.

[30] Rigetti, "Docs for the Forest SDK," 2021. [Online]. Available: https://pyquil-docs.rigetti.com/

[31] S. Sivarajah *et al.*, "t|ket⟩: A retargetable compiler for NISQ devices," *Quantum Science and Technology*, vol. 6, 2020.

[32] M. Fingerhuth, T. Babej, and P. Wittek, "Open source software in quantum computing," *PLOS ONE*, vol. 13, no. 12, pp. 1–28, 2018.

[33] D. Vietz, J. Barzen, F. Leymann, B. Weder, and V. Yussupov, "An Exploratory Study on the Challenges of Engineering Quantum Applications in the Cloud," in *Proceedings of the 2nd Quantum Software Engineering and Technology Workshop (Q-SET 2021) co-located with IEEE International Conference on Quantum Computing and Engineering (QCE21)*. CEUR Workshop Proceedings, Oct. 2021, pp. 1–12.

[34] E. A. Sete, W. J. Zeng, and C. T. Rigetti, "A functional architecture for scalable quantum computing," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, 2016, pp. 1–6.

[35] W. Edwards, "How to use multiattribute utility measurement for social decisionmaking," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 7, no. 5, pp. 326–340, 1977.

[36] M. J. Swain and D. H. Ballard, "Color indexing," *International Journal of Computer Vision*, vol. 7, no. 1, pp. 11–32, 1991.

[37] University of Stuttgart. (2021) NISQ Analyzer Content Repository. [Online]. Available: https://github.com/UST-QuAntiL/nisq-analyzer-content/tree/paper/prioritization/prioritization

[38] J. C. Ros, *Introduction to Decision Deck-Diviz: Examples User Guide*. Departament d'Enginyeria Informàtica i Matemàtiques, 2011.

[39] (2021) PROMETHEE. [Online]. Available: http://www.diviz.org/workflow.method.PROMETHEE.html

[40] (2021) ELECTRE 3. [Online]. Available: http://www.diviz.org/workflow.method.ELECTRE-3.html

[41] (2021) AHP-TOPSIS. [Online]. Available: http://www.diviz.org/workflow.AHP-TOPSIS.html

[42] 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.

[43] E. Bernstein and U. Vazirani, "Quantum Complexity Theory," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1411–1473, 1997.

[44] IBMQ team. (2021) 5-qubit backend: IBM Q Lima backend specification V1.0.23. [Online]. Available: https://quantum-computing.ibm.com

[45] ——. (2021) 5-qubit backend: IBM Q Manila backend specification V1.0.17. [Online]. Available: https://quantum-computing.ibm.com

[46] ——. (2021) 5-qubit backend: IBM Q Quito backend specification V1.1.16. [Online]. Available: https://quantum-computing.ibm.com

[47] ——. (2021) 32-qubit simulator: IBM Q QASM simulator specification V0.1.547. [Online]. Available: https://quantum-computing.ibm.com

[48] University of Stuttgart. (2021) Ranking of Compiled Circuits Based on Identified Metrics Table. [Online]. Available: https://github.com/UST-QuAntiL/nisq-analyzer-content/blob/paper/prioritization/prioritization/rankings-of-compiled-circuits.csv

[49] Y. Zhou, E. M. Stoudenmire, and X. Waintal, "What Limits the Simulation of Quantum Computers?" *Phys. Rev. X*, vol. 10, p. 041038, Nov 2020.

[50] G. S. Ravi, K. N. Smith, P. Murali, and F. T. Chong, "Adaptive job and resource management for the growing quantum cloud," 2021.

[51] G. S. Ravi, K. N. Smith, P. Gokhale, and F. T. Chong, "Quantum computing in the cloud: Analyzing job and machine characteristics," 2021.

[52] M. Grossi *et al.*, "A serverless cloud integration for quantum computing," 2021.

[53] J. A. Cruz-Lemus, L. A. Marcelo, and M. Piattini, "Towards a set of metrics for quantum circuits understandability," in *Quality of Information and Communications Technology*, A. C. R. Paiva, A. R. Cavalli, P. Ventura Martins, and R. Pérez-Castillo, Eds. Cham: Springer International Publishing, 2021, pp. 239–249.

[54] L. Bishop, S. Bravyi, A. Cross, J. Gambetta, J. Smolin, and March, "Quantum Volume," 2017.

[55] A. Wack *et al.*, "Quality, Speed, and Scale: three key attributes to measure the performance of near-term quantum computers," 2021.

[56] M. Guo, Q. Zhang, X. Liao, F. Y. Chen, and D. D. Zeng, "A hybrid machine learning framework for analyzing human decision-making through learning preferences," *Omega*, vol. 101, p. 102263, 2021.