An Exploratory Study on the Challenges of Engineering Quantum Applications in the Cloud

Daniel Vietz, Johanna Barzen, Frank Leymann, Benjamin Weder and Vladimir Yussupov

University of Stuttgart, Institute of Architecture of Applications Systems, Universitätsstr. 38, 70569 Stuttgart, Germany

Abstract

The rapid evolution of quantum computation in the cloud creates considerable opportunities for multiple real-world application scenarios, including chemical simulation, optimization, and machine learning. Typical quantum applications are hybrid as they consist of both classical and quantum components. The latter require quantum computers for execution, which are often offered as cloud services. Thus, to implement quantum applications, developers need to have expertise in integration of quantum and classical components of the application, as well as understanding the relevant cloud-specific challenges and limitations. In this work, we explore the challenges which can be encountered when designing and implementing hybrid quantum applications in the cloud and identify which limitations of current quantum cloud services make such integration complex. To achieve this, we (i) implemented four quantum applications highlighting different scenarios of using quantum software components in cloud applications and (ii) analyzed the challenges and limitations encountered during the implementation process and documented the key observations. In addition, we discuss open research questions and ways to address them to improve the process of developing quantum applications in the cloud.

Keywords

Cloud Computing, Quantum Computing, Hybrid Quantum Applications, Quantum-Classic Integration

1. Introduction

Quantum computing is an emerging field, which promises to solve many problems from different domains more efficiently or with better precision compared to classical computers [1, 2, 3], e.g., optimization, machine learning, or simulation of chemical molecules [4]. Quantum computers are often provided as cloud services, making them available to a broader audience and allowing their usage in real application scenarios [5]. However, existing quantum computers are limited in available resources and prone to errors, e.g., due to the instability of the generated quantum states [6, 7]. In addition, operations on quantum computers are often imprecise. Thus, current quantum computers are referred to as *Noisy Intermediate-Scale Quantum (NISQ)* [6] computers. Due to these limitations and given the fact that quantum computers are only superior in certain cases, they cannot fully replace their classical counterparts [1, 6]. Instead, quantum computers

2nd Quantum Software Engineering and Technology Workshop, co-located with IEEE International Conference on Quantum Computing and Engineering (QCE21) (IEEE Quantum Week 2021), October 18–22, 2021

vietz@iaas.uni-stuttgart.de (D. Vietz); barzen@iaas.uni-stuttgart.de (J. Barzen); leymann@iaas.uni-stuttgart.de (F. Leymann); weder@iaas.uni-stuttgart.de (B. Weder); yussupov@iaas.uni-stuttgart.de (V. Yussupov)

0000-0003-1366-5805 (D. Vietz); 0000-0001-8397-7973 (J. Barzen); 0000-0002-9123-259X (F. Leymann); 0000-0002-6761-6243 (B. Weder); 0000-0002-6498-637X (V. Yussupov)

can be employed for solving *only specific tasks they are best suited for*, whereas the remaining tasks are performed using classical computers, e.g., preparing inputs for quantum algorithms, storing data, or processing user interactions [8]. Therefore, such hybrid quantum applications, or hQApps for short, inherently combine both worlds: classical and quantum computers.

Consequently, to implement hQApps in the cloud, software engineers need to deal with the implementation and integration of quantum software components that rely on specialized services such as IBM Quantum (IBMQ) [9], AWS Braket [10], or Azure Quantum [11], and classical software implemented using traditional cloud offerings. As a result, the development of hQApps requires a team that combines not only quantum-specific expertise but also expertise in more traditional domains such as cloud computing, software integration, service-oriented architectures, and workflow technology [12]. Therefore, it is crucial to understand the challenges such interdisciplinary teams of hQApps developers are facing.

In this work, we explore common challenges from different domains which can be encountered when designing and implementing hQApps that combine traditional and quantum cloud offerings from commercial providers. Therefore, the main research question in this work can be formulated as follows: "*Which design and implementation challenges are commonly encountered when engineering hQApps in the cloud?*"

To address this question, we (i) design and implement four quantum application scenarios based on the existing literature focusing on engineering hQApps [13, 14]. As a result, we present scenarios using different types of interaction and of varying complexity: from a "simple" application generating random bit strings to composite applications that involve integrating different quantum tasks and hybrid algorithms that rely on optimization loops. Next, we (ii) analyze the challenges encountered during the design and implementation of each application scenario, and (iii) discuss key observations and open research questions.

After having covered the background and fundamentals in Section 2, Section 3 presents the implemented scenarios. Section 4 presents and discusses the identified challenges, and Section 5 outlines related work. Finally, Section 6 gives a summary and an outlook on future work.

2. Background and Fundamentals

In general, there are different quantum computation models such as the gate-based, measurement-based, or quantum annealing computation model. These computation models influence the implementation of quantum algorithms [15, 16]. In this work, we focus on the gate-based quantum computation model using so-called quantum circuits to formulate computation steps executed on a quantum computer. Within a quantum circuit, multiple qubits form a quantum register and specific operations, so-called quantum gates, are applied to that register [5].

Quantum computers are often provided as services in the cloud and, hence, can be used on demand by a broader audience [5]. These quantum cloud services typically offer application programming interfaces (APIs), enabling the execution of quantum circuits on real quantum processing units (QPUs) or quantum simulators running on classical hardware. Since QPUs and also simulators enable computing quantum circuits, we use the term "quantum computer" for both. To facilitate and support the implementation of quantum circuits and their execution on quantum computers, providers usually offer software development kits (SDKs) [16, 17].

Since today's quantum computers are limited, most algorithms already applicable during the NISQ era are hybrid, i.e., they combine computations on quantum and classical computers [7]. Variational algorithms, such as the Variational Quantum Eigensolver (VQE) [18] for determining eigenvalues, and the Quantum Approximation Optimization Algorithm (QAOA) [19] for approximating the solution of an optimization problem, are common types of hybrid algorithms [20]. They use a parametrized quantum circuit (called ansatz) and optimize the measuring results classically by varying the input parameters of the ansatz in each iteration [20].

3. Integration Scenarios

In this section, we present four scenarios that highlight various quantum-classical integration aspects including different kinds of interaction, composition of multiple quantum tasks in one application, and implementation of hybrid algorithms with the control flow encompassing classical and quantum tasks. The first two scenarios show how quantum and classical components can be composed into simple sequences of actions. The last two scenarios show more complex compositions in which the intended control flow spans classical and quantum components and has loops and conditions. To explain these more complex scenarios, we model them using the Business Process Model and Notation (BPMN) [21], a well-known standard which provides a visual notation for modeling business processes.

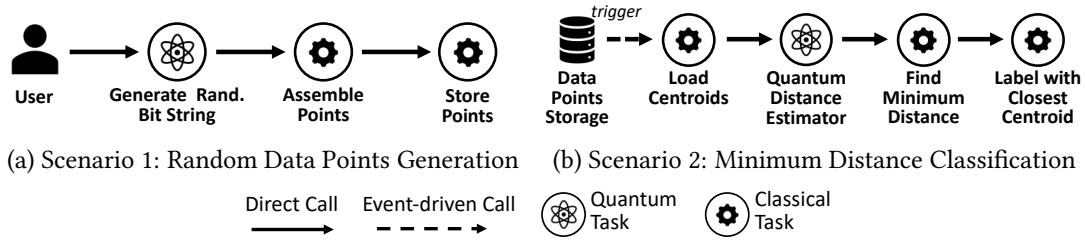
We implemented the scenarios using the cloud offerings of two providers, namely Amazon and IBM. The quantum components are implemented using Python, based on Qiskit and the AWS Braket SDK, respectively. Additional libraries, such as Boto3 and Pennylane, were also used to implement individual scenarios. Developed workflow models are specified and executed using the Camunda workflow system [22], which comprises a graphical modeling tool and a state-of-the-art BPMN workflow engine. The implemented scenarios are open-source and available via GitHub [23].

Scenario 1: Random Data Points Generation

Due to their inherent quantum-mechanical properties, quantum computers facilitate generating true random bits [24]. The first scenario shown in subfigure (a) of Fig. 1 focuses on generating random bit strings using quantum computers and post-processing the generated random bits using classical computers. For this, the quantum task *Generate Random Bit String* needs to be integrated with the classical task *Assemble Points* which interprets the random bits and stores them in the desired format, e.g., as two-dimensional data points. Finally, another classical task needs to persist the generated random data.

Scenario 2: Minimum Distance Classification

In the second scenario, data points are classified w.r.t. a given set of classes using the distance to their centroids. The distance is calculated on a quantum computer using a quantum distance estimator [25, 26]. Subfigure (b) in Figure 1 shows the classical task *Load Centroids* getting invoked by a trigger-event in the *Data Points Storage* (e.g., an insert event). After the centroids have been loaded, the distances between the data point and all centroids are computed using



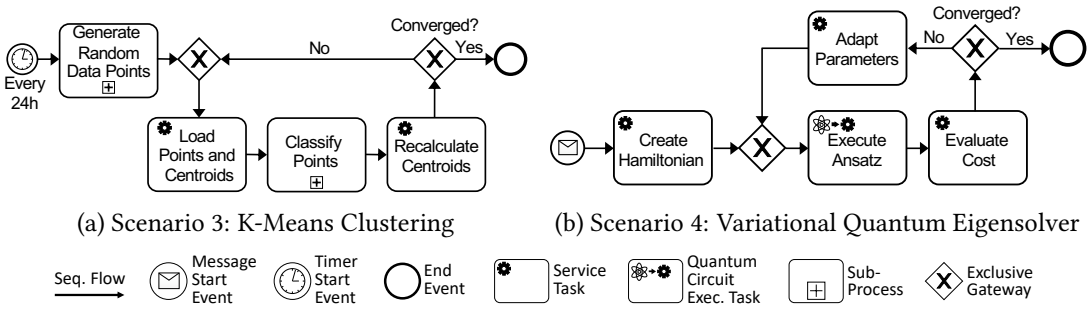
(a) Scenario 1: Random Data Points Generation (b) Scenario 2: Minimum Distance Classification

Figure 1: Example scenarios of sequential quantum and classical tasks integration

the quantum task called *Quantum Distance Estimator*. This quantum task normalizes the data points, encodes them as angles on the unit circle [27], and uses a SWAP-Test [28] to check to what extent the prepared quantum states differ. The results are used by two classical tasks: the first finds the minimum distance and the second labels that information to the data point.

Scenario 3: K-Means Clustering

This scenario demonstrates the composition of the components from the first two scenarios to implement the quantum k-means clustering algorithm [26]. Subfigure (a) of Figure 2 depicts the composition of required components as a *BPMN Process*. First, the execution is scheduled by a timer start event which triggers the execution of this scenario once every 24 hours. The first task called *Generate Random Data Points* is depicted as a *BPMN Sub-Process*: this task generates random data points on a quantum computer, i.e., it executes the random data points generator (see Scenario 1) as a first activity. These random data points are used as initial values for the centroids, which will be recalculated later. After the data points and centroids are loaded using a classical service task, the data points are assigned to the centroids using another sub-process, which is the *Minimum Distance Classification* of Scenario 2. The last task recalculates the centroids using the mean of all data points assigned to them. The *BPMN Exclusive Gateways* are used to model a loop in which the aforementioned three tasks are repeated until the algorithm converges, i.e., by reaching a maximum number of iterations or if centroids no longer need to be updated.



(a) Scenario 3: K-Means Clustering (b) Scenario 4: Variational Quantum Eigensolver

Figure 2: Example scenarios of quantum and classical tasks integration using workflows

Scenario 4: Variational Quantum Eigensolver

This scenario represents a hybrid quantum-classical algorithm called Variational Quantum Eigensolver (VQE) [18] for approximating the smallest eigenvalue of a hermitian matrix. Subfigure (b) in Fig. 2 shows this scenario modeled in BPMN. Based on the input matrix, the first task creates a Hamiltonian (or a combination of several Pauli strings). Following the structure of variational quantum algorithms [20], an ansatz is then executed on a quantum computer. Afterwards, a cost function is evaluated, which is defined as the expectation value of the Hamiltonian, i.e., the sum of the expectation values of the Pauli strings. The workflow minimizes this cost function by "tweaking" the parameters of the ansatz. Due to the variational principle, the expectation value is always greater or equal to the smallest eigenvalue, thus, the expectation value gets minimized in order to approximate the minimum eigenvalue. The algorithm stops if the changes of the expectation value are below a certain threshold.

4. Engineering Challenges of hQApps in the Cloud

The design and implementation of hQApps require expertise from different domains, such as quantum computing, cloud computing, workflow technology, etc. In this section, we discuss various challenges from different domains we encountered when designing and implementing the scenarios introduced in Section 3.

Challenge 1: Identify Quantum-Classical Split

One of the first encountered challenges is the task of splitting a problem into classical and quantum components [29, 12]. This challenge of quantum-classical split is twofold. Firstly, it is necessary to identify which problem parts are suitable for computation on quantum computers [3]. This requires identifying suitable quantum algorithms and assessing relevant functional and non-functional characteristics, e.g., their performance in the context of a given problem [30]. Here, one needs also to decide which computation model will be used to implement the quantum parts [12]. In particular, this concerns the decision between universal computation models, such as gate-based and measurement-based quantum computing, or restricted computation models, such as quantum annealing [16]. Since quantum circuits are wrapped by classical source code handling the execution, further classical logic can be combined into this wrapper. Thus, the second aspect is to decide which classical logic should be implemented tightly coupled with the quantum circuit. For example, the loading of the centroids in Scenario 2 can be combined with the execution of the quantum distance estimator.

Key Observations and Open Research: Turning requirements into an architecture design is difficult for classical applications as well as for quantum applications. One way to facilitate this process is to model hQApps using workflow languages as they provide a good overview of the intended control flow w.r.t. involved activities, which can serve as a basis for identifying better component boundaries, e.g., based on the observed data locality and interaction patterns. Although a separation of classical and quantum problem parts into separate components makes the implementation more concise and maintainable, strict separation is not always preferable due to performance reasons. For example, variational algorithms, as described in Scenario 4,

comprise quantum and classical computations, and splitting them introduces an additional communication overhead. Thus, it can be preferable in some cases to combine quantum and classical parts into one component. We also used this strategy in our implementation of Scenario 4. A "good" quantum-classic split via properly-defined component boundaries improves maintainability and reusablility of application components while at the same time ensures efficient execution. However, finding a suitable quantum-classical split is highly problem-dependent and requires further research.

Challenge 2: Identify a Suitable Quantum Service Type

Another challenge encountered when designing hQApps is to choose a cloud service type suiting the defined application requirements. For instance, one important factor to consider is whether the selected quantum cloud service supports the required interaction type, e.g., implementation of direct API calls.

Key Observations and Open Research: In general, available quantum service offerings fall into one of the following four categories:

(i) *Circuit Composers* offer a combination of graphical and textual quantum circuit editors. Examples are the IBMQ Composer and the QI Editor in Quantum Inspire [31]

(ii) *Jupyter Notebook Services* enable the development and direct execution of quantum components in a document-style manner where source code is accompanied by documentation and console instructions. IBM Quantum Lab and AWS Braket Notebooks are offerings which fall into this category.

(iii) *Quantum Computation as a Service (QCaaS)* offerings receive computation requests over an API. Thus, they require to construct a quantum circuit first to be sent to this interface.

(iv) *Hybrid Cloud Services* (e.g., [32]) allow transmitting hybrid components consisting of both classical and quantum parts which are managed and executed by service providers.

Although *Circuit Composers* allow to create quantum circuits which are stored in the cloud and can be executed multiple times on various quantum computers, they do not allow external input variables to be used and cannot establish a connection to other services. Currently, *Jupyter Notebook Services* do not provide endpoints for implementations, so they cannot be triggered externally. In the future, however, providers could add such functionality, making Jupyter Notebook Services a sufficient option for hosting invocable quantum components. The structure of current *QCaaS* offerings implies hosting on classical cloud components which connect to QCaaS APIs to perform quantum computations. First, some classical component creates a quantum circuit, which is then sent to the API of the QCaaS where it gets executed on a quantum computer. The response is interpreted by the classical component that further defines all external interfaces needed for invocation. *Hybrid Cloud Services* enable to create hybrid quantum components hosted in the cloud that can be invoked over an HTTP endpoint.

Some of our scenarios also use event-driven interaction between components. However, none of the currently available quantum cloud services we are aware of allow this type of interaction.

Thus, suitable offerings from the classical cloud must be selected. For example, the quantum component can be implemented as a function hosted on a FaaS offering, such as AWS Lambda or IBM Cloud Functions, establishing a connection to a QCaaS offering. The function can be bound to specific events so that it gets executed when they occur.

Challenge 3: Decide on Quantum Computer Utilization Strategy

Quantum computers are often shared by multiple users, with quantum circuits typically being queued before execution. Compared to the actual computation time on the quantum computer, this can result in a rather long total execution duration. Therefore, before implementing the hQApp it is important to choose the desired quantum computer utilization strategy.

Key Observations and Open Research: Since the execution of hQApps depends on the selected quantum computer, a suitable utilization strategy must be selected. In multi-circuit scenarios, e.g., it has to be decided whether to use several quantum computers or execute all quantum circuits on the same one. Furthermore, it must be decided how to select those quantum computers. Since quantum computers provide varying capabilities, such as qubit count and computation accuracy, one strategy is to find all suitable instances for a given problem [30] and use the instance with the fewest amount of requests waiting in the queue. Since the utilization of available quantum computers can be high, many quantum-specific services offer to book time-slots for exclusive access to certain quantum computers. Since it incurs additional costs, it is especially useful for scenarios that perform many quantum computations such as Scenario 3 and Scenario 4. Booking a time slot for single circuit executions rises the question of how to collect multiple execution requests to fit in a time slot, i.e., how to estimate execution times for quantum components.

Additionally, it must be considered whether quantum circuits should be executed sequentially or in parallel. For example, in Scenario 2, the distance estimations of multiple data points are independent from each other and can be computed in parallel. In contrast, Scenario 4 requires iterative execution of the quantum circuits. When parallel computing is possible, bulk processing features should be used to transmit multiple quantum circuits simultaneously rather than sequentially. To avoid sending each circuit individually to the interface, hybrid cloud services can be used. These upcoming services combine the execution of quantum computations with classical computations and allow iterative execution of multiple circuits behind the queue. Thus, they are especially useful for iterative scenarios such as Scenario 3 and 4.

To determine an appropriate quantum computer utilization strategy, it is important to first analyze the overall structure of the hQApp. Although the analysis is currently a manual task, it could be automated in certain ways. For example, the iterative nature of applications, such as in Scenario 3 and Scenario 4 can be identified by detecting loops in the workflow model [8]. Additional actions to reserve time-slots for exclusive access can then be integrated by extending the workflow. However, availability of quantum computers might not be assured anytime and time slots might be postponed. Another approach could merge existing implementations of quantum and classical tasks as hybrid components, e.g., using the Qiskit Runtime. However, an automated realization may pose further challenges, e.g., if quantum and classical components are implemented in different programming languages.

Challenge 4: Select Components Integration Style

The individual components need to be integrated to form the application system, raising the question of how to integrate quantum components with classical components.

Key Observations and Open Research: In general, different approaches can be used to integrate different components. For example, tightly-coupled components can interact with each other by means of hard-coded endpoint calls. However, tight coupling conflicts with the separation of concerns principle, hence, hindering the maintainability. To avoid this, more loosely-coupled approaches can be used, e.g., required components can be composed using the workflow technology [33]. Quantum-specific extensions of conventional workflow technologies have also been proposed [34]. Another option is to use messaging-based integration, e.g., to decouple the preparation of quantum circuits from the processing of the results. Here, the QCaaS offerings are used as an external component integrated using the *Service Activator Pattern* [35]. Fig. 3 shows this pattern – quantum circuits are no longer sent directly to the QCaaS, but to a *Request Queue* that the *Service Activator* listens to. The Service Activator, e.g., hosted on a PaaS offering, communicates with the QCaaS API to submit the quantum circuits and poll for computation results. It then forwards the results to the *Response Queue*.

Challenge 5: Implement for Specific Hardware

When implementing hQApps, one issue that must be taken into account is that the choice for certain quantum hardware (one quantum service offering might provide several quantum hardware options) influences the actual implementation.

Key Observations and Open Research: In contrast to classical cloud service offerings, quantum cloud services do not incorporate virtualization techniques; instead users must manually select quantum computers and align their implementations with them. Selecting the instance with the least number of jobs in the queue, as mentioned in Challenge 3, is one approach to avoid manual selection. Since it does not incorporate information about the number of required qubits, it might pick an unsuitable quantum computer to execute a certain quantum circuit. Thus, the approach is useful for circuits requiring only few qubits, because in these cases "small" quantum computers can also be chosen. Another possible solution is to make hardware selection configurable within the application, e.g., by implementing the *Content Enricher* pattern [35]. Figure 4 shows an example implementation of the Content Enricher, in which the application still creates quantum circuits, however, without referencing concrete quantum computers. This

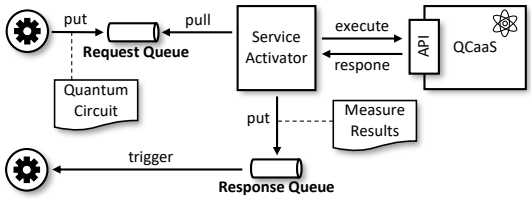


Figure 3: Message-based Integration of QCaaS Using a Service Activator [35]

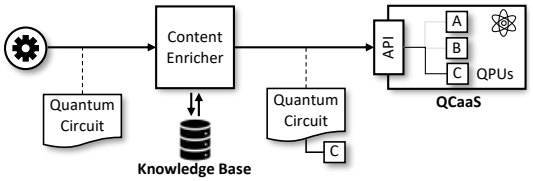


Figure 4: Configurable Hardware Selection Using the Content Enricher Pattern [35]

information is injected into the quantum circuits as they pass through. However, multiple hardware-specific criteria [36] must be considered to enable such configurable behavior, e.g., the connectivity of qubits and the average error rates of different operations. Additionally, it is important to check whether the given quantum circuit can be executed successfully on the selected quantum computer, e.g., using existing tools [30] that help automating the selection.

In the NISQ-era, however, the selection of suitable quantum computers for running an existing implementation is only one aspect. In addition, the implementation itself must be aware of the hardware limitations in the NISQ era, e.g., to avoid creating too large quantum circuits. As hardware continues to improve, the question arises of how to implement applications that are as flexible as possible to grow with progress.

5. Related Work

There are already various publications that discuss challenges of developing hQApps. Rahaman et al. [37], e.g., present different problems of QCaaS offerings in general and focus on reliability and security issues. Leymann and Barzen [7] point out several pitfalls for the successful and efficient implementation of quantum algorithms in the NISQ era: algorithms are often presented without considering crucial steps, such as state preparation, oracle expansion, connectivity, etc. Rojo et al. [38] present an empirical study describing the tribulations of quantum-classical microservice systems. However, they focus on technical properties, such as the *number of qubits* and *response times*, of current quantum cloud services. Hevia Olivera [39] presents requirements for quantum service providers, discussing several challenges related to technical limitations and the diversity of current offerings. In previous work [40, 12] we present the development lifecycle of quantum applications showing that expertise from different areas is required. While the aforementioned works discuss challenges only related to technical conditions or the pure implementation of quantum algorithms, in this work we explore the engineering challenges faced by interdisciplinary teams of hQApps developers.

6. Summary and Future Work

In this work, we analyzed the challenges of engineering hQApps in the cloud by conducting an exploratory study comprising four different integration scenarios. We have shown that it is important to understand how to split a problem into classical and quantum components. On the one hand, fine-granular decomposition leads to better maintainability and reusability. On the other hand, it is crucial to minimize communication overhead between quantum and classical computations in some scenarios, e.g., hybrid algorithms. One strategy, thus, can be to start with a monolith-first implementation and decompose it into classical and quantum components, paying attention to avoid unnecessary communication overhead. Another important step is to decide what type of quantum service to use and how to utilize available quantum computers. This may involve booking exclusive access or result in a quantum computer being shared among multiple applications. It is also important to decide how individual components should be integrated, as the integration style influences the design and implementation of a hQApp and its components. There are several options for this, such as message-based integration

or orchestration based on workflow technologies. In contrast to tightly-coupled and hard-coded integration, the aforementioned approaches are more robust to changes and offer better reusability, but also require additional expertise, e.g., regarding the use of quantum-specific cloud services. Quantum-specific cloud services, unlike classical cloud services, do not incorporate any virtualization techniques. Hence, implementations of hQApps are hardware-specific and cannot run on arbitrary quantum computers.

In future work, we aim to improve tooling support for the design of quantum applications. Thereby, we plan to extend previous work [16] to also incorporate decision support and guide developers in their decision for specific tools and services. Furthermore, we also want to evaluate the feasibility of different virtualization approaches for quantum hardware.

Acknowledgments

This work was partially funded by the BMWi projects *PlanQK* (01MK20005N), the project *SEQUOIA* funded by the Baden-Württem-berg Ministry of Economy, Labour and Housing, and the DFG's Excellence Initiative project *SimTech* (EXC 2075 - 390740016).

References

[1] National Academies of Sciences, Engineering, and Medicine, Quantum Computing: Progress and Prospects, National Academies Press, 2019.

[2] T. Gabor, et al., The Holy Grail of Quantum Artificial Intelligence: Major Challenges in Accelerating the Machine Learning Pipeline, 2020. arXiv:2004.14035.

[3] J. Barzen, F. Leymann, M. Falkenthal, D. Vietz, B. Weder, K. Wild, Relevance of Near-Term Quantum Computing in the Cloud: A Humanities Perspective, Cloud Computing and Services Science 1399 (2021) 25–58.

[4] A. Acín, et al., The quantum technologies roadmap: a european community view, New Journal of Physics 20 (2018) 080201.

[5] F. Leymann, J. Barzen, M. Falkenthal, D. Vietz, B. Weder, K. Wild, 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.

[6] J. Preskill, Quantum Computing in the NISQ era and beyond, Quantum 2 (2018) 79.

[7] F. Leymann, J. Barzen, The bitter truth about gate-based quantum algorithms in the NISQ era, Quantum Science and Technology 5 (2020) 044007.

[8] F. Leymann, J. Barzen, Hybrid Quantum Applications Need Two Orchestrations in Superposition: A Software Architecture Perspective, 2021. arXiv:2103.04320.

[9] IBM, IBM Quantum, 2021. URL: https://quantum-computing.ibm.com.

[10] Amazon.com, Inc, AWS Braket, 2021. URL: https://aws.amazon.com/braket.

[11] Microsoft, Azure Quantum, 2021. URL: https://azure.microsoft.com/services/quantum/.

[12] B. Weder, J. Barzen, F. Leymann, D. Vietz, Quantum Software Development Lifecycle, 2021. arXiv:2106.09323.

[13] J. Zhao, Quantum Software Engineering: Landscapes and Horizons, 2020. arXiv:2007.07047.
[14] C. A. Pérez-Delgado, H. G. Perez-Gonzalez, Towards a Quantum Software Modeling Language, in: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW'20, Association for Computing Machinery, 2020, p. 442–444.
[15] J. A. Miszczak, Models of quantum computation and quantum programming languages, Bulletin of the Polish Academy of Sciences: Technical Sciences 59 (2011) 305–324.
[16] D. Vietz, J. Barzen, F. Leymann, 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, pp. 127–141.
[17] R. LaRose, Overview and Comparison of Gate Level Quantum Software Platforms, Quantum 3 (2019) 130.
[18] A. Peruzzo, et al., A variational eigenvalue solver on a photonic quantum processor, Nature Communications 5 (2014) 4213.
[19] E. Farhi, J. Goldstone, S. Gutmann, A Quantum Approximate Optimization Algorithm, 2014. arXiv:1411.4028.
[20] M. Weigold, J. Barzen, F. Leymann, D. Vietz, Patterns For Hybrid Quantum Algorithms, in: Proceedings of the 15th Symposium and Summer School on Service-Oriented Computing (SummerSOC), Springer International Publishing, 2021, pp. 34–51.
[21] OMG, Business Process Model and Notation (BPMN) Version 2.0, Object Management Group (OMG), 2011.
[22] Camunda, Camunda BPMN Workflow System, 2021. URL: https://camunda.com.
[23] University of Stuttgart, Practical aspects quantum engineering challenges, 2021. URL: https://github.com/vietzd/qc-cloud-challenges.
[24] M. Herrero-Collantes, J. C. Garcia-Escartin, Quantum random number generators, Reviews of Modern Physics 89 (2017) 015004.
[25] M. Schuld, M. Fingerhuth, F. Petruccione, Implementing a distance-based classifier with a quantum interference circuit, EPL (Europhysics Letters) 119 (2017) 60002.
[26] S. U. Khan, A. J. Awan, G. Vall-Llosera, K-Means Clustering on Noisy Intermediate Scale Quantum Computers, 2019. arXiv:1909.12183.
[27] M. Weigold, J. Barzen, M. Salm, F. Leymann, Data Encoding Patterns For Quantum Computing, in: Proceedings of the 27th Conference on Pattern Languages of Programs, The Hillside Group, 2021. Accepted for publication.
[28] M.-S. Kang, J. Heo, S.-G. Choi, S. Moon, S.-W. Han, Implementation of SWAP test for two unknown states in photons via cross-Kerr nonlinearities under decoherence effect, Scientific Reports 9 (2019) 6167.
[29] R. Pérez-Castillo, M. A. Serrano, M. Piattini, Software modernization to embrace quantum technology, Advances in Engineering Software 151 (2021) 102933.
[30] M. Salm, J. Barzen, U. Breitenbücher, F. Leymann, B. Weder, K. Wild, 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.
[31] T. Last, et al., Quantum Inspire: QuTech's platform for co-development and collaboration in

quantum computing, in: Novel Patterning Technologies for Semiconductors, MEMS/NEMS and MOEMS 2020, volume 11324, International Society for Optics and Photonics, SPIE, 2020, pp. 49 – 59.
[32] IBM, Qiskit Runtime, 2021. URL: https://github.com/Qiskit-Partners/qiskit-runtime.
[33] F. Leymann, D. Roller, Production Workflow: Concepts and Techniques, Prentice Hall PTR, 2000.
[34] B. Weder, U. Breitenbücher, F. Leymann, K. Wild, Integrating Quantum Computing into Workflow Modeling and Execution, in: Proc. of the 13th IEEE/ACM Int. Conf. on Utility and Cloud Computing (UCC 2020), IEEE Computer Society, 2020, pp. 279–291.
[35] G. Hohpe, B. Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley, 2004.
[36] M. Salm, J. Barzen, F. Leymann, B. Weder, About a Criterion of Successfully Executing a Circuit in the NISQ Era: What $wd \ll 1/\epsilon_{\text{eff}}$ Really Means, in: Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software (APEQS 2020), ACM, 2020, pp. 10–13.
[37] M. Rahaman, M. M. Islam, A Review on Progress and Problems of Quantum Computing as a Service (QCaaS) in the Perspective of Cloud Computing, Global Journal of Computer Science and Technology 15 (2015).
[38] J. Rojo, D. Valencia, J. Berrocal, E. Moguel, J. García-Alonso, J. M. M. Rodriguez, Trials and Tribulations of Developing Hybrid Quantum-Classical Microservices Systems, 2021. arXiv:2105.04421.
[39] J. L. Hevia Olivera, Requirements for Quantum Software Platforms, in: 1st Quantum Software Engineering and Technology Workshop, 2020, pp. 20–26.
[40] B. Weder, J. Barzen, F. Leymann, M. Salm, D. Vietz, The Quantum Software Lifecycle, in: Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software (APEQS 2020), ACM, 2020, pp. 2–9.