Towards Cloud Agnostic Quantum-Classical Hybrid Computing

Dang Hai Luong

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.
Espoo 29.5.2023

Supervisor

Assist. Prof. Dr. Paler Alexandru

Advisors

Dr. Valtteri Lahtinen

Asser Lähdemäki

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Dang Hai Luong	
Title Towards Cloud Agnostic Quantum-Classical Hybrid Computing	
Degree programme Computer, Communication and Information Sciences	
Major Computer Science	**Code of major** SCI3042
Supervisor Assist. Prof. Dr. Paler Alexandru	
Advisors Dr. Valtteri Lahtinen, Asser Lähdemäki	
Date 29.5.2023	**Number of pages** 57+1 **Language** English

Abstract

Quantum classical hybrid computing is a paradigm which describes systems of classical and quantum computers that enable running quantum algorithms and hybrid algorithms consisting of classical and quantum parts in a programmable interface. Such quantum classical hybrid systems are provided by popular cloud computing providers such as Amazon Web Services (AWS), Azure, Google Cloud and IBM, to name a few. Using these services, researchers can access either real quantum computers or high-performance quantum simulators without owning the expensive hardwares. However, each service comes a different set of available features, capabilities and, for example, supported set of basic quantum gates. Thus it is important to understand the differences between each service's provided capabilities when choosing a suitable cloud provider to run a quantum circuit.

Managing cloud-provisioned infrastructure is an issue that should also be solved. One possible solution to such issue is to deterministically manage the cloud infrastructure using Infrastructure as Code (IaC) methodology. The methodology enables declarative and deterministic approach to manage and provision cloud infrastructure via human-readable definition files. Abstraction is one additional benefit of using IaC, which means IaC definitions can be designed to be independent of cloud service providers, or cloud-agnostic.

The purpose of this thesis is to designs an Infrastructure-as-code definition in Terraform language to manage cloud infrastructure from AWS and Azure. The definition is designed to enable an easy switch between such services. The thesis also explores and compares 2 above mentioned cloud providers. In particular, the thesis focuses on each provider's capabilities and efficiency of designing and running different quantum circuits.

Keywords quantum computing, hybrid computing, terraform, infrastructure as code

Preface

I want to thank Asst. Prof. Dr. Alexandru Paler and my advisors Dr. Valtteri Lahtinen and Asser Lähdemäki for their guidance. I also want to thank my friend Aashish Sah for supporting me and giving me hints and tips during the research of this thesis.

Otaniemi, 29.5.2023

Dang Hai Luong

Contents

Abbreviations

Abbreviations

SDK	Software Development Kit
SaaS	Software-as-a-service
PaaS	Platform-as-a-service
IaaS	Infrastructure-as-a-service
AWS	Amazon Web Services
API	Application Programming Interface
IaC	Infrastructure as Code

1 Introduction

Quantum computing is an emerging and swiftly developing discipline that seeks to make use of the characteristics of quantum mechanics to conduct calculations beyond what's possible of conventional computers. Quantum computers conduct certain types of calculations more quickly than classical computers. In addition, quantum computers may employ the quantum mechanical property of superposition to conduct certain forms of parallel calculations [15, 47, 14]. As a result of these characteristics, quantum computers have the potential to become a useful tool for the solution of certain sorts of problems, such as the factoring of big numbers and the searching of enormous databases, which are now either difficult or impossible to solve with conventional computers [53].

Quantum computers have the capability for resolving particular challenges significantly quicker than classical computers, but their development is still in its infancy. Consequently, hybrid computing combines the capabilities of classical and quantum computers to resolve problems that are challenging or impossible for either form of computer to solve on its own [28].

1.1 Quantum gates

Quantum gates are essential to quantum computing since their function is to permit the systematic and predetermined transformation and processing of quantum information. Single qubit gates and multiple qubit gates are the two sorts of quantum gates used for influencing quantum states in quantum computing. Single qubit gates operate on a single qubit and serve the purpose to modify the qubit's state or to carry out quantum state space rotations. The Pauli X gate, the Pauli Y gate, and the Pauli Z gate are common single-qubit gates. These gates are utilized for operations such as inverting the qubit state, rotating the qubit state in the X-Y plane, and altering the qubit's phase, respectively. Multiple qubit gates, on the contrary, operate simultaneously on multiple qubits and are utilized to entangle several qubits or conduct complex quantum state operations. The CNOT gate, the 16 SWAP gate, and the Toffoli gate are examples of common multiple-qubit gates. These gates are utilized for entangling qubits, exchanging the states of two qubits, and performing conditional operations on the state of multiple qubits, respectively [14].

1.2 Quantum circuits

Quantum circuits represent the manipulation and processing of quantum information. At a high level, a quantum circuit consists of a series of quantum gates applied to a set of quantum bits (qubits). These gates are unitary transformations that alter the state of the qubits in a well-defined way [47].

Quantum circuit is the ability to entangle qubits, which allows them to interact and perform operations that are not possible on classical computers. For instance, a quantum circuit may entangle two qubits in such a manner that a measurement of one qubit relies on the state of the other qubit, even though the two qubits are physically separated by a significant distance. This is possible even if the two qubits are at distinct locations. One of the primary distinctions between quantum and classical computing may be found in this phenomena, which is referred to as quantum non-locality [47].

Quantum circuits can also be used to conduct other quantum operations, such as quantum teleportation, which enables the transmission of quantum states between distant qubits, and quantum error correction, which protects against errors and decoherence caused by environmental factors [14].

1.3 Hybrid computing

Quantum hybrid computing is a type of computing that combines the capabilities of classical computing with those of quantum computing. It involves using both classical computers and quantum computers to solve problems that are too complex or too time-consuming for classical computers to solve on their own [28].

Due to their ability to utilize the principles of quantum mechanics, quantum computers can conduct certain types of calculations significantly faster than classical computers. However, they are still in the developmental phases. Using classical computers for certain tasks and quantum computers for others, quantum hybrid computing enables us to take advantage of the strengths of both types of computers. Using classical computers to process and analyze data generated by quantum computers is an example of quantum hybrid computing. This can help surmount the limitations of quantum computers and enable the solution of complex problems that were previously intractable [28].

1.3.1 Applications of quantum hybrid computing

There are a few examples of quantum hybrid computing:

1. Quantum optimization: Optimization problems are a form of mathematical problem in which, given a set of possible solutions and a set of constraints, the objective is to identify the optimal solution. Using quantum algorithms, it is possible to simultaneously explore a vast space of potential solutions [4, 18].

2. Quantum chemistry: Quantum computers can be used to perform quantum chemistry calculations, which can help to better understand the properties of chemical compounds and predict their behavior [4, 18].

3. Quantum simulations: Quantum computers can be used to simulate complex systems, such as the behavior of materials or the behavior of biological molecules. Quantum simulation helps to revolutionize the field of computational fluid dynamics (CFD) by providing new computational capabilities beyond classical computing, by utilizing quantum parallelism and quantum superposition to perform multiple computations simultaneously [4, 18, 15].

4. Quantum encryption: Quantum computers can be used to enhance the security of classical encryption systems by generating and distributing keys that are much more difficult to break [4, 18, 15].

Quantum hybrid computing has the potential to improve many different areas of science, engineering, and technology by providing more powerful and efficient ways to solve complex problems.

1.4 Structure of the thesis

The thesis is organized in the following way: Chapter 2 contains the theoretical background and high-level description of quantum computing, the background on cloud infrastructure, as well as Infrastructure as Code (IaC) and its advantages, as well as introduce popular IaC tools. Chapter 3 describes the research methodology. Chapter 4 describes the experiment execution of running the circuit in parallels in different clouds using IaC tool. Chapter 5 shows the benchmark result of previous experiment, concludes the thesis and shows possible future work.

2 Background: Quantum computing

This chapter contains a brief introduction to quantum circuits and quantum gates. The mathematics of quantum computing involves the use of linear algebra and matrix operations to manipulate and measure quantum states. We show a brief mathematical representation of quantum resources, namely, qubits, entanglement, and superposition. We discuss about quantum linear solver and the Qiskit quantum computing software development kit.

2.1 Qubits and gates

Quantum bits, or qubits, are the fundamental information elements in quantum computing. In contrast to classical bits in conventional computing, which signify information as the binary digits 0 or 1, qubits reflect the concept of superposition, enabling them to simultaneously represent both states [7, 55, 15]. Consequently, qubits provide an exponential growth in computational power as the number of qubits increases, thereby expediting complex problem-solving [39].

Beyond superposition, another property of qubits is entanglement. Entanglement is a fundamental quantum mechanical characteristic in which pairings or groups of qubits may communicate in such a way that the state of each qubit becomes inextricably linked to the state of the others, regardless of the distance between them. Within an entangled state, information about one qubit instantaneously reveals information about the other entangled qubits, enabling parallelism and interconnectivity on a scale unattainable in classical systems [22, 26].

Typically, a qubit is implemented using a quantum system with two levels, such as the spin of an electron or the polarization of a photon [12]. These systems can be manipulated using various quantum operations, or quantum gates, to perform quantum computations.

Quantum gates are the foundational components of quantum computing. Quantum gates possess unique properties that distinguish them from classical gates. These properties arise from the principles of quantum mechanics and the most important ones are unitarity and reversibility. Quantum gates are unitary, meaning their operations preserve the total probability of the quantum state. Every quantum gate is reversible, meaning for every operation, there exists an inverse operation that can undo its effect. Quantum gates can operate on multiple qubits at once, enabling complex, multi-particle interactions [14]. Due to the no-cloning theorem, quantum gates cannot produce an exact copy of an indeterminate undetermined quantum state. This property fundamentally differentiates quantum information processing from classical computing, where information can be copied freely [54].

2.1.1 Superposition

Quantum superposition is a concept in quantum mechanics that explains the simultaneous probabilities of multiple quantum states. It asserts that a quantum particle, such as an electron or photon, can simultaneously exist in multiple locations or possess multiple values of energy, momentum, or spin. In contrast to classical mechanics, in which objects are described as existing in definite states, this is the case. Quantum superposition is fundamental to comprehending the behavior of quantum systems and has numerous applications in disciplines such as quantum computing, cryptography, and quantum simulations. The concept of quantum superposition continues to be one of the most captivating and intriguing aspects of quantum mechanics [47].

In the realm of quantum mechanics, the state of a quantum system can be described mathematically through the use of wave functions. These wave functions are complex-valued functions that characterize the probability of a quantum system being observed in a particular state. These wave functions can be represented as linear combinations of the wave functions of their constituent states [47]. For example, consider a quantum system with two basis states, $|0\rangle$ and $|1\rangle$. Those basis states are the orthonormal basis when the states are visualized in Hilbert space. In the matrix form, those states can be represented as:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

If the quantum system is in a superposition state, its wave function can be written as:

$$\psi(x) = \alpha\,|0\rangle + \beta\,|1\rangle$$

where α and β are complex numbers known as quantum system amplitudes. The amplitudes relate to the probability of measuring the quantum system in the $|0\rangle$ and $|1\rangle$ states, respectively.

The idea of quantum superposition, which is an essential component of quantum mechanics, may be shown with the help of the Bloch sphere, which is named after its inventor. The Bloch sphere is a kind of hypersphere that has three dimensions and depicts all of the states that are conceivable for a quantum system. It consists of the black local Cartesian axis vectors, the red Bloch vector for the subsystem, and the blue dashed scaled correlation axes. The scaled correlation axes of each Bloch sphere are pairwise perpendicular, and they are designated with their index i to denote the correlation pairing that arises between the two Bloch spheres [13]. Figure 1 shows a visual representation of a bloch sphere.

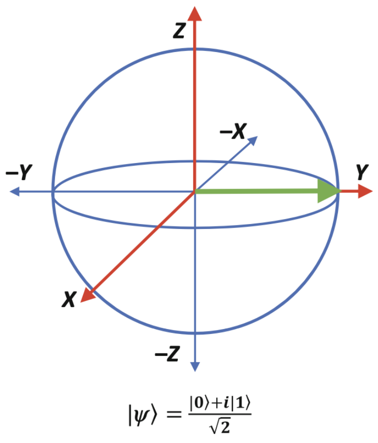


Figure 1: Bloch sphere conveying the initial state vector of a qubit at $\frac{1}{\sqrt{2}}(|0\rangle + i\,|1\rangle)$ [47]

The principle of superposition is an essential part of quantum mechanics and may be found at the core of a wide variety of quantum phenomena, including quantum entanglement and quantum tunneling.

2.1.2 Entanglement

Quantum entanglement is a phenomenon in quantum mechanics that is a special kind of correlation that exists between two or more quantum systems. It is a fundamental property of all quantum systems [22], and has been discovered to have a number of uses in fields like quantum computing and cryptography [26].

Entanglement in quantum physics is mathematically explained by the idea of a composite system. A system made up of two or more smaller systems, sometimes referred to as subsystems, is referred to as a composite system. When the component subsystems of a composite system are not yet entangled, the state of the composite system can be indicated as the tensor product of the subsystems' states. Consider, for instance, a composite system with two smaller systems, A and B, each containing a single qubit. The state of the composite system can be written as:

$$|\psi\rangle = |\psi_A\rangle \otimes |\psi_B\rangle$$

Where $|\psi_A\rangle$ is the state of subsystem A and $|\psi_B\rangle$ is the state of subsystem B.

In order to ascertain the degree of entanglement existing in a system of multiple particles in pure states, a technique known as the Schmidt decomposition is utilized in quantum information and quantum computing [42]. It is a technique for breaking down a wavefunction into a collection of terms, each containing the wavefunction of a different subsystem. For every bipartite quantum system, this decomposition also enables us to represent a pure state vector in terms of a single sum rather than two sums [35]. The decomposition expresses the composite system's state as a sum of product states:

$$|\psi\rangle = \sum_i \lambda_i \, |\phi_i\rangle_A \, |\psi_i\rangle_B$$

where λ_i are known as the Schmidt coefficients, and $|\phi_i\rangle_A$ and $|\psi_i\rangle_B$ are known as the Schmidt basis states.

An important property in quantum computing and quantum information science is entanglement, which allows quantum systems to do particular tasks that are impossible for conventional computers to perform. Quantum teleportation, which shows the non-local features of entangled quantum systems and makes a significant addition to the science of quantum information, is one such activity made possible by entanglement. [14].

2.1.3 Density matrix

The density matrix, which characterizes the statistical traits of a quantum system, is one of the most important mathematical instruments employed in quantum computation. Mixed states, which are states that are a "probabilistic mixture of pure states", may be depicted by using the density matrix, which can be used to describe mixed states [34]. It additionally has the ability for describing the state of a quantum system when noise or decoherence is present [48, 27]. Here is an example of a density matrix for a qubit in a mixed state:

$$\rho = p\,|0\rangle\langle 0| + (1-p)\,|1\rangle\langle 1|$$

This density matrix represents a mixed state in which the qubit has a probability p of being measured as $|0\rangle$ and a probability $(1-p)$ of being measured as $|1\rangle$.

The density matrix can be used to calculate the expected value of observables, such as the Pauli matrices, which are a set of three 2x2 complex matrices that are commonly used to represent quantum states [48, 27]. The Pauli matrices are defined as:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Where the X, Y, and Z matrices represent the spin of a particle along the x, y, and z axes, respectively.

2.1.4 Unitary matrix

Unitary matrices are important in both linear algebra and quantum physics. A square matrix with an inverse equal to its transpose conjugate is known as a unitary matrix. This characteristic ensures that a vector's inner product and norm are preserved in unitary matrices. As a result, linear transformations that preserve a vector space's geometric structure are described by unitary matrices. Unitary matrices are used in quantum physics to express rotations and other spatial transformations, in addition to the chronological progression of a quantum system [27]. Here is an example of a unitary matrix in quantum computing:

$$U = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$$

This is the matrix illustration of a quantum gate called a rotation gate, which shifts a qubit by an angle of θ along the y-axis.

2.1.5 Hadamard gate

The Hadamard gate is a prime instance of a quantum gate, and it is one that works on a single qubit. The Hadamard gate is characterized by the matrix shown following:

$$H = \frac{1}{\sqrt{2}} \times \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

The Hadamard gate modifies the state of a qubit from its computational base state $|0\rangle$ or $|1\rangle$ to the superposition state $(|0\rangle + |1\rangle)/\sqrt{2}$ or $(|0\rangle - |1\rangle)/\sqrt{2}$, respectively [43].

2.1.6 Phase shift gate

To apply a phase shift to a qubit, the phase shift gate spins the qubit's state vector around the origin of the complex plane [25]. An unitary matrix which characterizes the phase-shift gate is shown below:

$$P = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}$$

where θ is the phase shift angle.

The phase shift gate's unitary transformation of the qubit state is represented by this matrix. For instance, the matrix would change the qubit to the state $|1\rangle$, if it were now in the state $|0\rangle$. The matrix would change the qubit from its current state of $|0\rangle$ to that of $e^{i}\theta\,|0\rangle$. The relative phase between a qubit's 0 and 1 states may be altered using the phase shift gate [25].

2.2 Quantum algorithms

In place of classical bits, qubits are used in quantum algorithms. A quantum algorithm typically consists of several key components:

1. Initialization: In this phase, the quantum computer's qubits' starting state is prepared. A mixture of single-qubit and multi-qubit gates may be used to accomplish this [49].

2. Quantum operations: This step is where the main computation takes place. It involves applying a series of quantum gates to the qubits in order to perform the desired computation. The specific operations used will depend on the algorithm being implemented [49].

3. Measurement: After the quantum operations have been performed, the algorithm's result is determined by measuring the eventual state of the qubits. This

step collapses the qubits into a classical state, which can be read out by the classical computer [49].

4. Classical post-processing: Some quantum algorithms require additional classical computation to process the output obtained in the measurement step [49].

Many quantum algorithms will also include error-correction methods in addition to these essential parts to counteract the effects of noise and decoherence on the quantum system.

In the subject of quantum computing, quantum algorithms and quantum circuits are ideas that are closely connected. A quantum algorithm is a process or collection of instructions that may be used by a quantum computer to carry out a particular calculation or solve a particular issue. High-level programming languages are used to create quantum algorithms, which are created to take use of the special capabilities of quantum computers [49]. Shor's method for factoring integers, Grover's algorithm for database searching, and quantum simulation algorithms are a few examples of quantum algorithms. A quantum circuit, on the other hand, uses a quantum computer to physically execute a quantum algorithm. To control the state of the qubits in the quantum computer, a network of quantum gates, which are the essential components of quantum computing, is employed. Circuit diagrams, which display the positioning of the quantum gates and the qubits on which they operate, are often used to describe quantum circuits. The actions outlined by the quantum algorithm are implemented by quantum circuits, which are also created to precisely control the qubits [14].

2.3 Quantum software frameworks

Quantum computing software development frameworks are tools and libraries that provide a set of functions and interfaces for constructing and manipulating quantum circuits, running them on quantum hardware or simulators, and analyzing the results. These frameworks are designed to make it easier to explore the potential of quantum computing. Software development frameworks for quantum computing comes in forms of software development kits (SDK) [52]. SDKs for quantum computing typically provide several key components, such as:

1. A quantum programming language: This is a high-level programming language that is used to write quantum algorithms. It is designed to be intuitive and

easy to use, and it allows developers to write quantum programs in a way that is similar to writing classical programs [52].

2. A quantum simulator: A conventional computer may now mimic the actions of a quantum computer thanks to this piece of software. Before launching their quantum applications on a real quantum computer, developers may use this tool to ensure they are bug-free [52].

3. A quantum compiler: A quantum program written in the quantum programming language may be translated by this program into a quantum circuit that can be run on a real quantum computer [52].

4. An interface to real quantum computers: Certain software development kits (SDKs) provide an interface to quantum computers that are hosted in the cloud. This enables software developers to execute their quantum applications on actual quantum hardware [52].

5. Additional tools and libraries: Some SDKs also provide additional tools and libraries for tasks such as visualization, optimization, and error correction [52].

Some of the popular and well-known SDKs include Qiskit (IBM), PyQuil (Rigetti), ProjectQ (Microsoft), and Cirq (Google). IBM's Qiskit is an open-source software development platform for quantum computing. It is designed to build and execute quantum programs on a variety of quantum hardware platforms, as well as simulate quantum circuits on classical computers [15].

The Qiskit is a collection of tools and libraries that may be used to create and manipulate quantum circuits, as well as run these circuits on quantum hardware or simulators and analyze the outcomes of these runs. In addition to that, it has a toolkit for controlling the running of quantum programs and enhancing the performance of quantum circuits [52].

One of the key features of Qiskit is its modular design, which allows users to customize and extend it to fit their specific needs. It is also designed to be user-friendly, with a simple, intuitive interface and extensive documentation.

2.4 Quantum computing cloud infrastructure

Building and maintaining a large-scale quantum computer requires significant investment in specialized hardware, software, and expertise. As a result, only a few organizations and research institutions have the resources and know-how to develop and operate quantum computers [4].

Cloud quantum computing offers access to quantum computing resources over the internet, eliminating the need for specialized expertise and expensive hardware. However, the performance is limited by internet connectivity and the processing power of quantum computers. Hybrid quantum computing overcomes these limitations by combining classical and quantum computing, allowing users to leverage the strengths of both approaches. The classical computer manages the quantum hardware and performs pre-processing and post-processing tasks [15]. The quantum hardware performs the quantum computations that are too difficult or slow to perform on classical hardware. Cloud hybrid quantum computing takes this approach further by combining cloud computing and hybrid quantum computing, providing the benefits of both [28].

Hybrid computing systems have become increasingly accessible to the general audience, largely due to the widespread availability and convenience of cloud infrastructure. Several prominent technology corporations, including IBM, Microsoft through its Microsoft Azure platform, Amazon via Amazon Web Services (AWS), and Google Cloud, have been instrumental in delivering these services to a broad user base. Cloud infrastructure, at its core, refers to the amalgamation of both physical and virtual resources that support the delivery of cloud services over the internet. These resources include servers, storage, and networking, which are managed and provisioned through virtualization and containerization technologies, and orchestration tools, to provide a consistent and programmable interface to customers.

There are 3 major distribution method of cloud infrastructure: [29, 44].

- IaaS is the most fundamental kind of cloud computing, entails the provisioning of virtualized components of a computer system via the internet. These resources can be used to run any software, including custom applications and operating systems [46]. Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) are examples of IaaS providers.

- PaaS offers a platform for the development and operation of custom applications. It consists of services such as databases, web servers, and middleware that can be used to deploy and operate applications without managing the required infrastructure [46]. Heroku, Amazon Elastic Beanstalk, and Microsoft Azure App Service are all examples of this distribution method.

- SaaS provides software applications over the internet, typically accessed via a web browser. These applications are fully managed and maintained by the provider and can be used by customers on a subscription basis [5, 46]. Salesforce, Google Apps, and Microsoft Office 365 are just a few examples of SaaS providers.

The provision of cloud infrastructure for quantum computing remains a specialized domain. Nevertheless, a select group of organizations and institutions that have managed to construct operational quantum computers are pioneering the accessibility of these potent systems [44]. They offer access to the public through various interfaces, including web-based platforms and application programming interfaces (APIs). These interfaces allow users to design and execute quantum algorithms without the need for direct physical access to the quantum hardware, thus democratizing quantum technology access. Some examples of companies that offer cloud-based quantum computing infrastructure include:

1. IBM: IBM Q is a family of quantum computers provided by IBM, which includes a range of systems with varying levels of quantum volume, qubit count, and connectivity. The IBM Q systems are designed to be accessible to a wide range of users, from researchers and scientists to businesses and organizations, through the IBM Cloud platform. IBM Q provides access to both hardware and software tools for quantum computing, including the IBM Quantum Experience, a cloud-based interface for experimenting with quantum algorithms and running quantum simulations [11, 15].

2. Rigetti Computing: Rigetti Computing is a technology company that provides quantum computing systems and services. The company offers a range of quantum computers, including superconducting and hybrid systems, which are designed for use in a variety of applications, including optimization, machine learning, and cryptography. Rigetti Computing also provides a cloud-based platform, Forest, that allows users to program and run quantum algorithms, as well as access to its quantum processors through APIs [55, 15].

3. D-Wave: D-Wave Systems Inc. exists as a Canadian corporation specializing in the commercialization and development of quantum computing. Quantum computers from D-Wave use a quantum annealing technique, which is designed to tackle optimization problems, and are used in many different kinds of applications, such as machine learning, cryptography, and drug discovery. The company provides access through the cloud to its quantum computers, and

its systems are utilized by government agencies, universities, and enterprises worldwide [32].

4. Google: Google Quantum Cloud is a quantum computing platform provided by Google. It provides cloud-based access to quantum computing resources, such as quantum processors and quantum simulators, for a broad spectrum of applications, such as optimization, machine learning, and cryptography. The Google Quantum Cloud also provides a suite of tools and services for developing, testing, and deploying quantum algorithms, making it accessible to a wide range of users, from researchers and scientists to businesses and organizations [40].

5. Microsoft Azure: Microsoft Azure offers a cloud-based quantum computing service called Azure Quantum. This service provides access to a range of quantum processors and simulators, including Microsoft's own quantum processors and simulators, as well as third-party quantum processors and simulators [40, 21].

6. AWS: Amazon Braket is a cloud-based quantum computing utility provided by AWS. This service provides access to a variety of quantum processors and simulators, including those offered by Amazon Web Services and by third-party vendors [17].

Despite the still-specialized nature of this field, the efforts made by these organizations to provide cloud-based access to quantum computing resources mark a significant stride in integrating quantum technology into the broader digital ecosystem.

2.4.1 IBM

IBM's Quantum Cloud platform provides access to its state-of-the-art quantum processors, including the IBM Q system, which has processors with up to 53 qubits. It also provides access to its simulators, software development kits and libraries, and other tools to help developers and researchers build and run quantum applications [11].

IBM has been actively working in the field of quantum computing for quite a while, and its quantum computing cloud infrastructure is considered one of the most advanced and mature in the industry. IBM Quantum Experience is an open platform that gives consumers access to the IBM Q systems and enables them to undertake experiments. IBM Quantum Composer is a graphical user interface that allows users to create quantum circuits by dragging and dropping components. IBM Quantum

Lab also provides a collaborative platform for researchers and developers to share their work, resources and expertise [11].

2.4.2 Microsoft Azure

Azure Quantum provides access to its own quantum processors, including the topological qubits based on anyons which is a unique approach in the field of quantum computing, in addition to providing access to third-party quantum processors such as IonQ and Quantinuum. It also offers a development environment that includes libraries, software development kits, and other tools to help developers and researchers build and run quantum applications [21].

Azure Quantum also provides access to a wide range of classical resources, such as virtual machines and storage, that can be used to run and support quantum applications. Microsoft has also partnered with a number of leading companies in the quantum computing industry, such as 1QBit, Zapata Computing, and Cambridge Quantum Computing, to provide customers with access to the latest quantum computing technologies and expertise [21].

Microsoft is actively working in the field of quantum computing and its Azure Quantum is considered one of the most comprehensive and well-integrated quantum computing platforms on the market.

2.4.3 AWS

Amazon Braket provides access to its own quantum simulators, as well as access to third-party quantum processors from companies such as IonQ, D-Wave, and Rigetti Computing. It also provides a development environment that includes libraries, software development kits, and other tools to help developers and researchers build and run quantum applications [17].

AWS also provides access to a wide range of classical resources, such as virtual machines and storage, that can be used to run and support quantum applications. Additionally, the service allows customers to use the same AWS Identity and Access Management (IAM) policies, security groups, and VPCs that they use for their other AWS resources.

2.4.4 Google Cloud

Google's Quantum Cloud platform provides access to its state-of-the-art quantum processors, including the Sycamore processor which was the first quantum computer that demonstrated quantum supremacy in 2019. Google also provides access to its simulators, software development kits and libraries, and other tools to help developers and researchers build and run quantum applications [40].

2.5 Infrastructure as code

A software engineering technique called infrastructure as code (IaC) enables programmers and system administrators to manage and provide computer resources without the need of manual procedures. IaC offers a consistent and repeatable method for building, deploying, and managing infrastructure, which lowers the possibility of human mistake, accelerates deployments, and boosts the agility of the development process [2, 1, 30, 2].

IaC entails describing the ideal state of the infrastructure using a high-level programming language, such as YAML or JSON. The infrastructure's setup, including the amount and size of servers, the software bundles to install, the network configuration, and the security rules, is defined by the code. IaC's "as code" suffix alludes to the use of software engineering practices, including version control, for upkeep of IaC scripts, which enables developers to work together on the code and monitor changes over time [38].

When the code is executed, it creates or updates the infrastructure to match the desired state. This can be done using a variety of tools, such as Ansible, Terraform, or CloudFormation, which automate the deployment and configuration of the infrastructure [2]. The tools provide a declarative approach to infrastructure management, which means that they focus on the desired state of the infrastructure, rather than the steps needed to get there. This makes it easier to manage complex infrastructures and to make changes without causing disruptions [2].

Infrastructure as Code (IaC) is particularly valuable in cloud hybrid computing, where the combination of cloud computing and hybrid quantum computing creates a complex and dynamic infrastructure. In cloud hybrid computing, the infrastructure consists of both classical and quantum computing resources, which must be provisioned and managed in a consistent and efficient way. IaC provides a way to manage both types of resources using the same code base, which simplifies the management of the infrastructure and reduces the risk of inconsistencies and errors.

At present, there exists no Infrastructure as Code (IaC) tools or frameworks that have been explicitly designed for the management of cloud hybrid quantum computing infrastructure. Nonetheless, certain IaC tools, including Terraform, provide extensibility via support for custom Application Programming Interfaces (APIs) implemented in Go programming language. This enables developers to create their own frameworks to support IaC for cloud hybrid quantum computing by leveraging the flexibility and extensibility of Terraform. In this way, developers can utilize the capability of Terraform to manage hybrid cloud infrastructure in combination with quantum computing resources [20]. While custom development is required to achieve this, the provision of a foundation via the use of tools such as Terraform that provide flexibility in API design and extensibility, supports the ability to create bespoke solutions for the management of cloud hybrid quantum computing infrastructure via IaC.

2.5.1 AWS cloud development kit

AWS Cloud Development Kit (CDK) is an Infrastructure as Code (IaC) tool that enables developers to define cloud infrastructure resources using familiar programming languages, such as TypeScript, Python, and Java, instead of domain-specific languages (DSLs). AWS CDK provides an object-oriented approach to defining cloud resources, which allows developers to create reusable and modular components that can be shared across projects. With AWS CDK, developers can define their infrastructure as code, which means that they can manage their cloud resources using a consistent and repeatable process. AWS CDK generates a CloudFormation template from the code, which can be used to create and manage the infrastructure resources on AWS [45].

Notably, AWS CDK is limited in its applicability to multi-cloud infrastructures as it solely supports a single cloud provider. This characteristic impedes its versatility in scenarios where multiple cloud providers are used, as developers may need to maintain disparate infrastructure configurations for each respective cloud provider.

2.5.2 Terraform

An infrastructure as code (IaC) tool called Terraform is intended to assist users in managing infrastructure and cloud resources. It is an open-source application created by HashiCorp that enables users to design infrastructure as code (IAC) and deliver it across several cloud service providers, including as AWS, Azure, and Google Cloud, as well as on-premises settings like bare metal servers, private clouds, and more. It enables users to define and provision a data center's infrastructure using configuration files. With Terraform, users can create, modify, and version their infrastructure while managing the entire lifecycle of their cloud resources. It supports a wide range of resource types, including virtual machines, databases, containers, networks, and storage. It also allows users to create reusable modules for their infrastructure, which can be shared and reused across multiple projects. This helps users to save time and effort, as well as increase consistency across their projects. Terraform uses a domain-specific language (DSL) called HashiCorp Configuration Language (HCL) to define infrastructure resources, and it can also be integrated with other tools such as Ansible and Puppet for configuration management [20].

One of the key benefits of using Terraform is its ability to manage infrastructure as code, which enables version control, collaboration, and reusability of infrastructure configurations. This allows for easier management and scaling of resources, as well as the ability to roll back changes in the event of errors or failures. Additionally, Terraform's support for multiple cloud providers allows for the creation of multi-cloud and hybrid environments, which can provide added flexibility and scalability [8].

Terraform is widely used in industry and academia for provisioning and managing cloud infrastructure. It is particularly useful for organizations that want to automate the provisioning and management of cloud resources, as well as for organizations that are moving towards a DevOps model for infrastructure management.

Here is an example of Terraform code that provisions an Amazon Web Services (AWS) EC2 instance:

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_instance" "example" {
  ami           = "ami-0ff8a91507f77f867"
  instance_type = "t2.micro"

  tags = {
    Name = "example-instance"
  }
}
```

Listing 1: "Simple EC2 Deployment using Terraform"

In this example, the provider block specifies that we are going to use the AWS provider and the region is set to us-west-2.

The resource block creates an EC2 instance using the aws_instance resource type. The ami (Amazon Machine Image) is set to a specific image ID and instance_type is set to t2.micro. The tags block is used to assign a name to the instance. Once this code is run, Terraform will create the specified EC2 instance and the user can verify that the instance has been created by looking at the AWS management console.

2.5.3 Cloud development kit for Terraform

The Infrastructure as Code (IaC) tool Terraform Cloud Development Kit (CDK) allows programmers to design cloud infrastructure resources. With the help of Terraform CDK, developers may define cloud resources using a class-based, object-oriented approach, resulting in reusable, modular components that can be used in several projects. Terraform CDK also provides a library of pre-built constructs, called "construct libraries", that represent the different infrastructure resources and their configurations. These libraries provide programmers access to a high-level abstraction that is tailored for the Terraform platform, allowing them to construct and manage infrastructure resources. In order to construct and manage infrastructure resources on a variety of cloud and on-premises platforms, Terraform CDK produces Terraform configurations from the code [19].

3 Methods

The chapter describes the benchmarking method of quantum computers with different quantum circuits. We examine the properties of quantum circuits and explore the impact these properties have on the selection of the experiment. The advantages of Infrastructure as Code (IaC) on the portability of quantum algorithm deployment is discussed.

The portability of quantum circuit execution is crucial because it enables the transfer of quantum circuits from one environment to another, while ensuring that they continue to function correctly [51]. By guaranteeing the portability of quantum circuit execution, researchers and developers can more easily test and deploy quantum circuits in new environments, which can help to advance the field more quickly and effectively [24]. Additionally, portability enables quantum circuits to be executed in a variety of settings, from local workstations to large-scale cloud-based quantum computers. This can help to ensure that quantum circuits are accessible to a wider range of users, regardless of their computational resources. Furthermore, it allows for easier collaboration between researchers and institutions, as quantum circuits can be shared and executed on different platforms [31].

In order to assess the portability of quantum circuits in various settings, it is imperative to ensure the transportability of their execution [51]. Infrastructure as Code (IaC) facilitates the seamless migration of quantum circuits across different environments. Infrastructrure as Code (IaC) eliminates the need for manual configuration, which can be time-consuming and prone to errors. By using IaC, the deployment process can be automated and streamlined, reducing the risk of errors and enabling quantum circuits to be deployed more quickly and easily. This can help to ensure that quantum circuits are executed consistently and accurately, even as they are moved between different environments. Another advantage of IaC is that it provides a single source of truth for the configuration of the infrastructure required to run quantum circuits. This makes it easier to maintain and update the infrastructure over time, as changes can be made in a centralized manner, rather than having to manually update each environment.

Terraform [20] leverages the application programming interface (API) of each cloud environment to instantiate and allocate cloud resources, thereby obviating the need for direct interaction with the cloud platform. Additionally, Terraform standardizes cloud deployment by providing a unified programming interface, facilitating the management of cloud resources across disparate environments [20]. At present, the conventional implementation of Terraform in certain cloud providers does not yet provide support for the deployment of a native quantum circuit execution environment. In such instances, it is advisable to deploy a generic execution environment, such as a virtual machine, in the cloud provider as an alternative.

3.1 Random quantum circuits and quantum algorithms

A random generated quantum circuit is one in which the quantum gates and other circuit elements are selected at random. One example of a randomly generated circuit is shown on Figure 2.

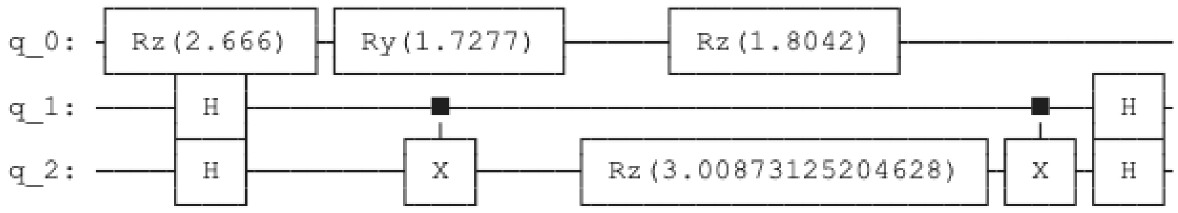


Figure 2: A randomly generated circuit created from Qiskit library

In contrast, a quantum algorithm is a structured and methodical sequence of quantum gates and other operations that are designed to solve a specific computational problem or to achieve a certain goal. One example of a quantum algorithm circuit is shown on Figure 3.

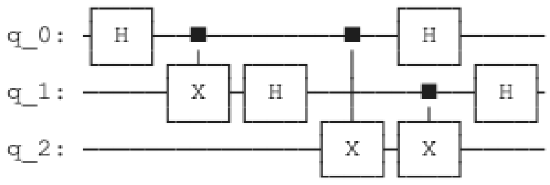


Figure 3: A quantum circuit for creating a hidden linear function

In other words, a random generated quantum circuit provides an unknown objective or purpose, while a quantum algorithm is created with a well-defined objective in mind. The properties and behavior of a random generated quantum circuit can be highly unpredictable, whereas a quantum algorithm is designed to produce a predictable outcome.

Consequently, it is advisable to utilize a quantum algorithm as opposed to a randomly generated quantum circuit when conducting benchmarking. In order to make informed decisions regarding which quantum algorithms to experiment with and benchmark, it is necessary to comprehend the properties of quantum algorithms in order to predict and accurately record the outcomes of the experiments.

There are various properties associated with quantum circuits. Within the context of this thesis. These properties form a basis for benchmarking. It is essential to have an understanding of:

1. The depth of the quantum circuits;

2. The number of quantum gates involved;

3. The required runtime of the quantum circuit;

4. The potential for parallelization.

3.1.1 Depth of a quantum circuit

An important consideration in computing with qubits is the depth of a quantum circuit. The number of quantum gate layers, that must be applied to the input quantum state in order to create the desired output, is referred to as the depth of a quantum circuit. The depth refers to the quantity of time-steps necessary to perform the circuit. It is a crucial element in defining a quantum circuit's overall complexity, as well as its error potential and the resources needed to execute it.

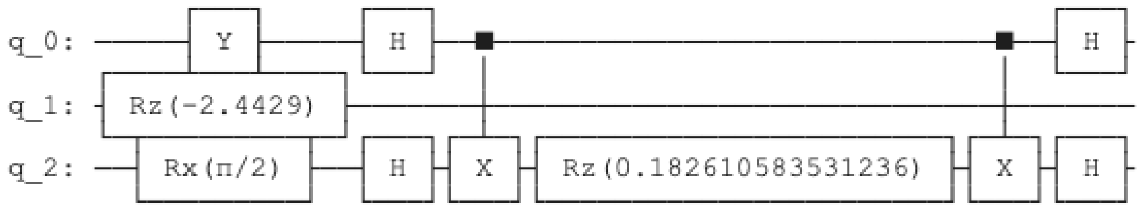


Figure 4: A randomly generated circuit with the depth of 2

Another definition of depth is the total number of nearest-neighbor qubits in the circuit, or as the number of two-qubit gates making up the circuit [6]. In either scenario, the execution time of a quantum circuit is precisely proportional to its depth [9].

3.1.2 Number of quantum gates in a quantum circuit

The number of gates in a quantum circuit is the total number of quantum operations, or quantum gates, carried out to the qubits throughout the computation.

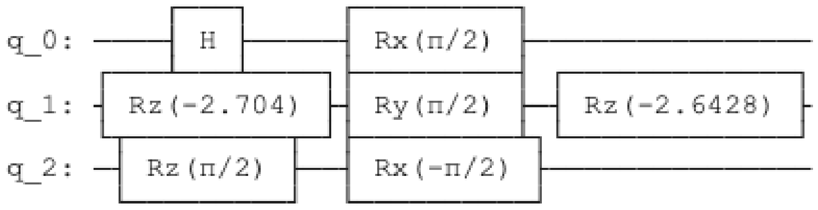


Figure 5: A randomly generated circuit with 7 quantum gates

Quantum circuit efficacy can be affected by a variety of factors, including the number of quantum gates utilized by the circuit. The correlation is not, however, inverse. In general, additional time would be required for additional entrances. Not all gates are created equal; some are more complicated and may require more time to implement. In certain circumstances, gates can be employed concurrently if they operate on distinct qubits and the quantum computing architecture permits it, which is discussed in Section 3.1.4.

3.1.3 Run time of a quantum circuit

The amount of time needed to execute a quantum circuit and produce the desired output is referred to as the runtime. Given that it directly affects the actual viability and scalability of the algorithms, the runtime is a key parameter for measuring the efficiency of quantum algorithms and quantum circuits.

The runtime of a quantum circuit is influenced by various factors, such as the number and type of quantum gates used, the number of qubits involved, and the specific requirements of the computational problem being addressed. In order to identify the required run time of a quantum circuit, classical pre- and post-processing tasks must be performed. This could include algorithm-independent tasks involved in the orchestration of a quantum circuit's pre- or post-processing tasks. For instance, if the workflow is going to allow the selection of appropriate quantum hardware depending on the input data at runtime, then the workflow is going to need to describe all of the various alternative implementations for the jobs [50].

The required runtime can also be impacted by the underlying hardware and infrastructure used to implement the circuit, such as the quality and performance of the quantum processors and the availability of quantum error correction techniques [50].

3.1.4 Parallelize quantum circuit execution

The ability to perform parallel execution of a quantum circuit is a key requirement for efficient quantum computing. To achieve this, the quantum circuit includes a finite-size universal instruction set of quantum gates such as $\{H, T, CNOT\}$ or a continuously-parameterized set such as $\{H, Rz(\theta), CNOT\}$. The quantum circuit is used to replicate a quantum system's dynamics for any given input state. To further optimize the circuit for parallel execution of the quantum gates, a classical algorithm can be devised to reduce the depth of the quantum circuit (e.g. [41]).

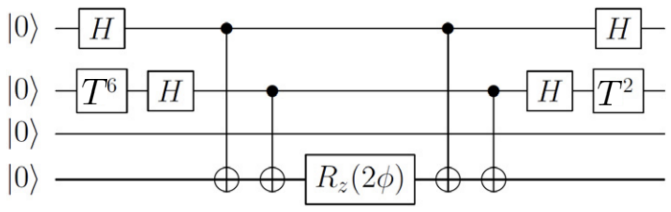


Figure 6: A quantum circuit to realize $exp(-i\theta X \otimes Y \otimes 1 \otimes Z)$ for a dimensionless evolution time θ using the Hadamard gate [41]

With the quantum circuit introduced in Figure 6, one alternative of parallelizing such circuit is to perform the quantum gates that are not dependent on the result of other gates.

Auxiliary qubits are additional qubits used to assist in quantum computing algorithms. They can be viewed as transient storage for a quantum computation, analogous to how temporary variables are utilized in classical computing. By acting as intermediaries, auxiliary qubits can allow additional operations to be conducted concurrently [36, 3].

3.2 Quantum circuit selection

The following three different benchmark quantum circuits will be used as benchmarks: Graph State Circuit, Greenberger-Horne-Zeilinger (GHZ) Circuit, and Hidden Linear Function Circuit. The selected quantum devices for executing these benchmark circuits include the IonQ device, Rigetti Aspen M-3, and Lucy. These devices were chosen based on their availability at the time of writing this thesis.

The Terraform script within this module plays a crucial role in spawning the AWS API client in different regions to support execution on the aforementioned quantum devices. By doing so, the module ensures that the benchmark circuits can be executed on a variety of quantum devices, irrespective of their geographical location.

For the benchmark circuits, the number of shots is determined based on the number of qubits in the circuit. Specifically, the shots are set equal to 10^{n-1}, where n represents the number of qubits. This approach allows for an appropriate balance between the level of accuracy and computational resources required for each benchmark circuit.

3.2.1 Graph state circuit

The structure of a graph state is grounded in graph theory, which offers an efficient and intuitive representation of quantum states in the context of quantum computing. In this framework, quantum states are mapped onto undirected graphs, where vertices correspond to qubits and edges signify the entanglement between these qubits. The underlying mathematical foundation of graph states is stabilizer formalism, where each qubit is associated with a set of stabilizer operators. These operators, which are tensor products of Pauli matrices, commute with the quantum state and can be used to capture the correlations between entangled qubits effectively. The graph representation allows researchers to perform various quantum operations and transformations by modifying the graph's structure, such as adding or removing vertices or edges, as well as local complementation.

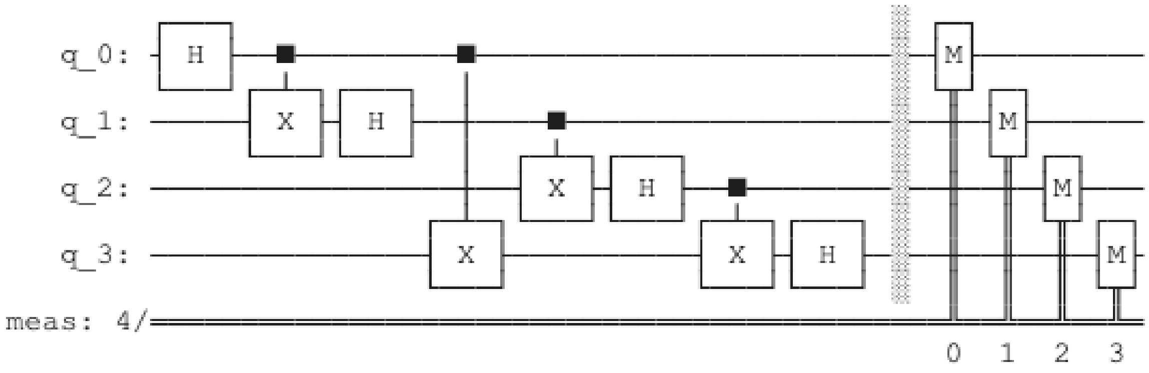
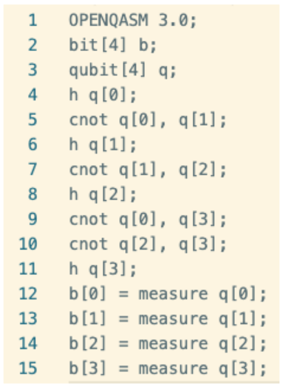


Figure 7: Qiskit rendered visualization of a Graph State circuit with 4 qubits

3.2.2 Greenberger-Horne-Zeilinger circuit

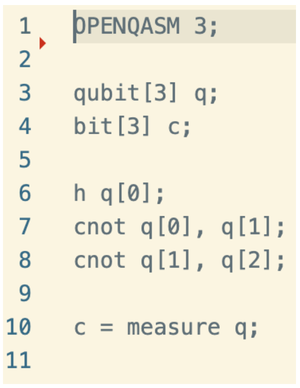
The GHZ circuit is constructed using a combination of single-qubit Hadamard (H) gates and multi-qubit controlled-NOT (CNOT) gates. The Hadamard gate is applied to the first qubit, initializing it in a superposition state, while the remaining qubits



```
OPENQASM 3.0;
bit[4] b;
qubit[4] q;
h q[0];
cnot q[0], q[1];
h q[1];
cnot q[1], q[2];
h q[2];
cnot q[0], q[3];
cnot q[2], q[3];
h q[3];
b[0] = measure q[0];
b[1] = measure q[1];
b[2] = measure q[2];
b[3] = measure q[3];
```

Figure 8: OpenQASM representation of a graph state circuit with 4 qubits

are left in their initial $|0\rangle$ state. Subsequently, a series of CNOT gates is applied, with the first qubit serving as the control and the remaining qubits as targets, successively.



```
OPENQASM 3;

qubit[3] q;
bit[3] c;

h q[0];
cnot q[0], q[1];
cnot q[1], q[2];

c = measure q;

```

Figure 9: OpenQASM representation of a GHZ circuit with 3 qubits

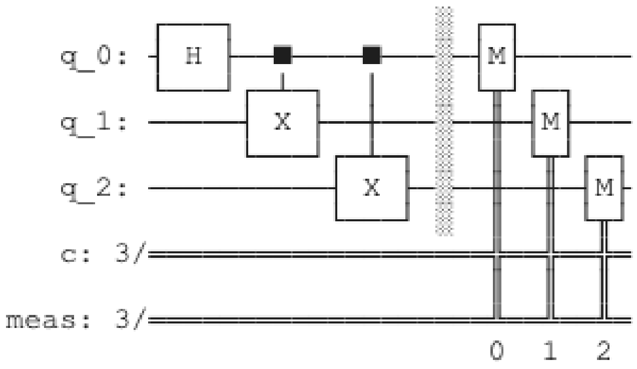
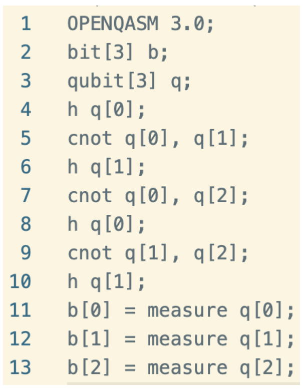


Figure 10: Qiskit rendered visualization of a GHZ circuit with 3 qubits

GHZ state in case of 2 qubits is commonly known as Bell state [16].

3.2.3 Hidden linear function circuit

The structure of the Hidden Linear Function (HLF) circuit involves multiple single-qubit and multi-qubit operations. An n-qubit input register is initialized to the $|0\rangle$ state, and a single qubit is prepared in the $|1\rangle$ state for the output register. A Hadamard (H) gate is applied to all qubits, placing them in a superposition. The quantum oracle, which encodes the hidden linear function, is then applied to the superposed state. The oracle operation involves a series of controlled-X (CNOT) gates that entangle the input register with the output register according to the secret linear function. After the oracle operation, another layer of Hadamard gates is applied to the input register, followed by a measurement. The measurement outcome reveals information about the secret linear function.



```
OPENQASM 3.0;
bit[3] b;
qubit[3] q;
h q[0];
cnot q[0], q[1];
h q[1];
cnot q[0], q[2];
h q[0];
cnot q[1], q[2];
h q[1];
b[0] = measure q[0];
b[1] = measure q[1];
b[2] = measure q[2];
```

Figure 11: OpenQASM representation of a HLF circuit with 3 qubits

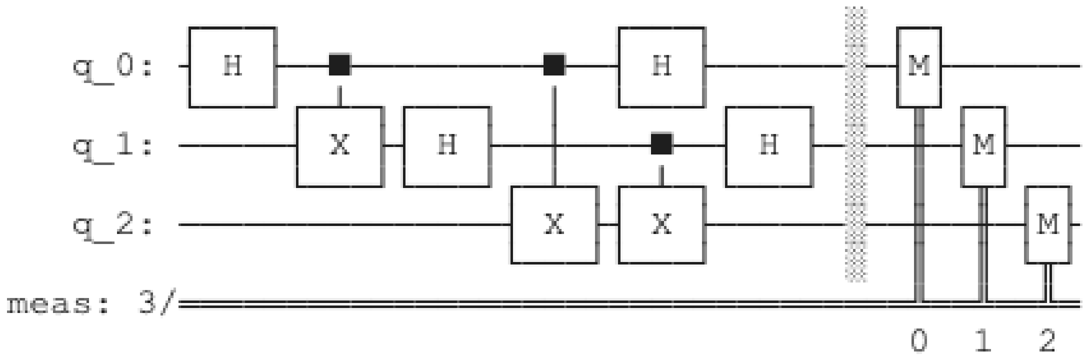


Figure 12: Qiskit rendered visualization of a HLF circuit with 3 qubits

3.2.4 Commonly supported quantum gates

In accordance with the AWS Braket documentation, Table 1 lists the common basis gates supported by each quantum device utilized in this study. By employing the *transpile* function from the *qiskit.compiler* package, it is possible to transpile any given circuit into the most frequently supported quantum gates.

	IonQ	**Rigetti Aspen M-3**	**Lucy**
x	✓	✓	✓
y	✓	✓	✓
z	✓	✓	✓
h	✓	✓	✓
cnot/cx	✓	✓	✓
cz		✓	✓
cy			✓
swap	✓	✓	✓
rx	✓	✓	✓
ry	✓	✓	✓
rz	✓	✓	✓

Table 1: Quantum gates supported by IonQ, Rigetti Aspen M-3, and Lucy

4 Implementation and evaluation

To determine the quantum computers whose execution results most closely align with simulated outcomes, various benchmarking methodologies can be employed. One viable approach involves normalizing all execution results to represent the probability of all conceivable quantum states across all qubits. Subsequently, a benchmarking metric can be utilized to compute a fidelity value for each quantum computer, effectively facilitating the identification of those that exhibit the highest degree of congruence with simulated results.

There are two different benchmarking methods implemented in this thesis:

1. Hellinger fidelity provides a means to quantitatively assess the similarity between two probability distributions.

2. Total Value Distance number serves as a means to quantitatively evaluate the dissimilarity between two probability distributions.

The Hellinger fidelity measure is particularly advantageous due to its ability to offer an intuitive, geometric interpretation of the distance between two distributions. In the realm of quantum computing, the Hellinger fidelity is commonly employed to evaluate the closeness of the execution results of quantum circuits and their corresponding theoretical or simulated outcomes [23].

Given two probability distributions P and Q, the Hellinger fidelity is calculated using the following formula:

$$H(P,Q) = (1 - \sum(\sqrt{p_i} - \sqrt{q_i})^2)^2$$

Where p_i and q_i denote the individual probability values of the distributions P and Q, respectively, and the summation runs over all possible outcomes. The resulting fidelity value ranges between 0 and 1, with 1 indicating perfect agreement between the two distributions, and 0 signifying complete dissimilarity [23].

The Total Variation Distance (TVD) fidelity is calculated using the following formula (P and Q are two probability distributions P and Q):

$$TVD(P,Q) = \frac{1}{2} * \sum |p_i - q_i|$$

Where p_i and q_i represent the individual probability values of the distributions P and Q, respectively, and the summation is carried out over all possible outcomes. The resulting fidelity value ranges from 0 to 1, with 0 indicating perfect agreement between the two distributions, and 1 signifying complete dissimilarity.

4.1 Architecture

The proposed system shown in Figure 13 aims to address the challenges of optimizing, deploying, and executing quantum circuits on various quantum computing devices. The system comprises three primary components: Circuit Exporter, Main Circuit Execution, and Benchmark Quantum Device Executor.

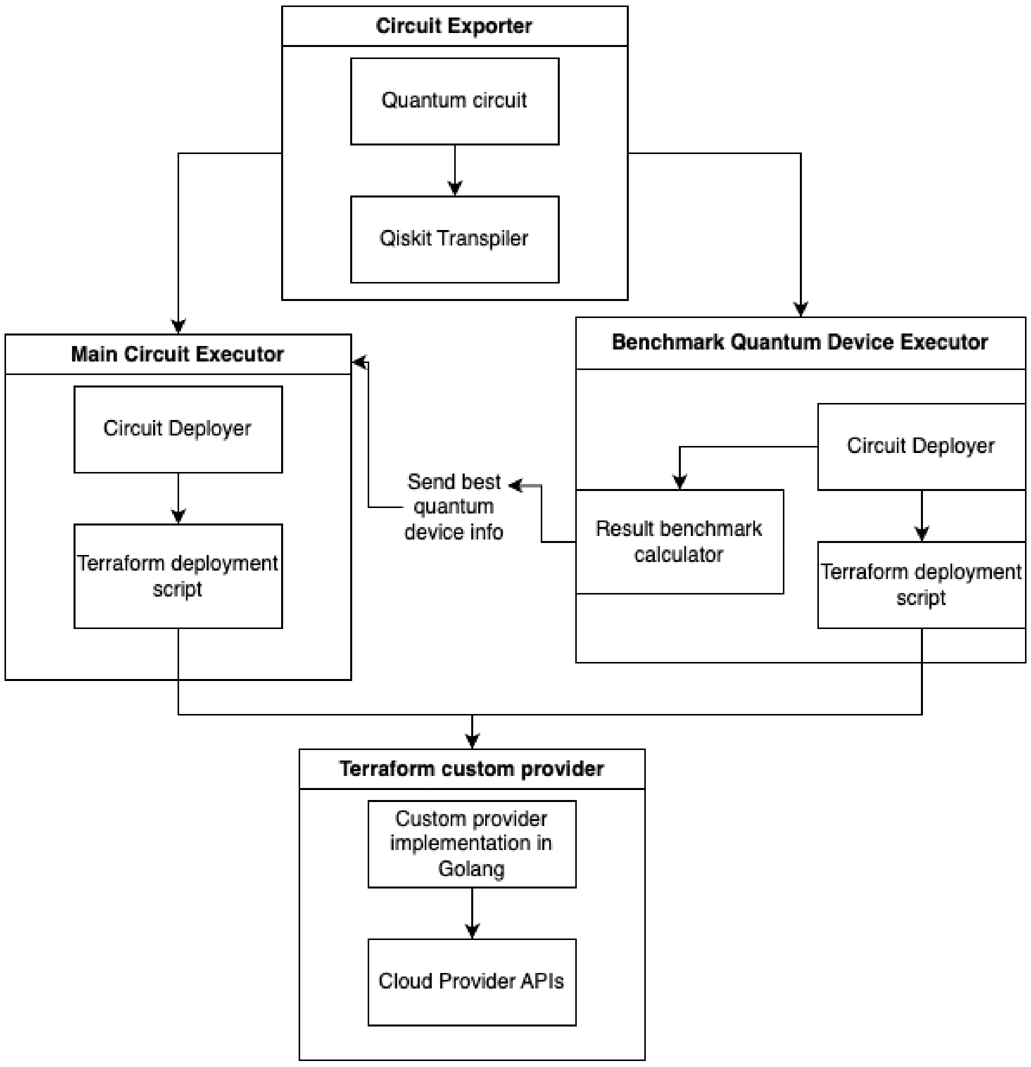

Circuit Exporter
Quantum circuit
Qiskit Transpiler
Main Circuit Executor
Circuit Deployer
Terraform deployment script
Benchmark Quantum Device Executor
Circuit Deployer
Result benchmark calculator
Terraform deployment script
Send best quantum device info
Terraform custom provider
Custom provider implementation in Golang
Cloud Provider APIs

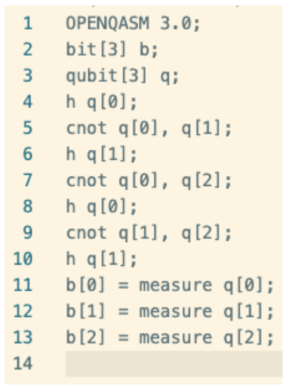
Figure 13: Dependency Graph of the System

The Circuit Exporter module utilizes Qiskit Transpiler to optimize quantum circuits for efficient execution on target devices. The Main Circuit Execution module is responsible for deploying and executing the optimized circuits using Terraform deployment scripts. These scripts automate the process of deploying quantum circuits to different cloud-based quantum computing devices. The Benchmark Quantum Device Executor module benchmarks available quantum devices by deploying and executing predefined tasks on them, while collecting and comparing the results to determine the best device for a given circuit. The system leverages a Terraform custom provider and cloud provider APIs to interact with quantum computing platforms.

4.1.1 Circuit exporter

The Circuit Exporter module is implemented as a Python script, this module leverages Qiskit [37] and Qiskit Braket [10] utilities to carry out its primary functions. Quantum circuits are created using the QuantumCircuit class from the Qiskit library, ensuring compatibility with a wide range of quantum computing devices.

Optimization of quantum circuits is achieved through Qiskit's transpilation process, which is configured to use a specific set of quantum gates: 'x', 'y', 'z', 'h', 'cnot', 'rx', 'ry', 'rz', and 'cx'. This choice of gates is informed by their universal compatibility that is listed on section 3.2.4, as all supported quantum devices are capable of executing these gates. Consequently, the Circuit Exporter module is equipped to maintain full compatibility across diverse quantum computing platforms.



```
OPENQASM 3.0;
bit[3] b;
qubit[3] q;
h q[0];
cnot q[0], q[1];
h q[1];
cnot q[0], q[2];
h q[0];
cnot q[1], q[2];
h q[1];
b[0] = measure q[0];
b[1] = measure q[1];
b[2] = measure q[2];

```

Figure 14: Output of the Circuit Exporter module for a hidden linear function circuit

Following the transpilation process, Qiskit Braket utilities are employed to convert the optimized quantum circuits into the OpenQASM [33] format. This conversion ensures that the circuits are represented in a standardized language, which further enhances their compatibility with various quantum computing devices.

4.1.2 Main circuit executor

The "Main Circuit Execution" section presents an essential aspect of the system that focuses on deploying and executing optimized quantum circuits on the best quantum computing device, as determined by the "Benchmark Quantum Device Executor" module. Implemented as a Python script, executed as a child process of a Python parent process using the *multiprocessing* library, the Main Circuit Execution module runs concurrently with the Benchmark Quantum Device Executor. The module is designed to accept a parameter representing the device ID of the optimal quantum computer, as identified by the Benchmark Quantum Device Executor.

One of the critical components within the Main Circuit Execution module is the Terraform script, which provisions Amazon Web Services (AWS) resources and declares AWS Braket quantum tasks to run a quantum circuit. The Terraform script leverages the custom Terraform provider, as mentioned in the system dependency chart, to interact seamlessly with the AWS infrastructure.

The module is designed to execute quantum circuits continuously, with each new execution commencing upon receipt of results from the previous run. This continuous execution approach allows for rapid feedback and analysis, enabling iterations and refinements of quantum circuits based on the results obtained from each execution.

4.1.3 Benchmark quantum device executor

The "Benchmark Quantum Device Executor" section is an integral component of the system that is responsible for benchmarking available quantum computing devices by deploying and executing predefined tasks on them. Implemented as a Python script, the Benchmark Quantum Device Executor module runs in parallel with the "Main Circuit Execution" module.

Upon completion of the benchmark executions, the results from all quantum devices are compared with the outcome of the quantum simulator DM1 from Amazon Braket service. The comparison utilizes the Hellinger fidelity calculation to determine which result is closest to the simulator's output. By identifying the quantum device with the highest fidelity, the Benchmark Quantum Device Executor module enables the system to select the most suitable quantum computing device for executing a given circuit.

4.1.4 Shared custom Terraform provider

At present, in the context of the emerging field of quantum computing services, Terraform does not offer innate support for deploying these cutting-edge services, necessitating the exploration of alternative approaches. To accommodate these evolving demands, the Terraform Provider Framework has been developed as an external

Application Programming Interface (API). This sophisticated framework empowers developers with a programming interface specifically designed to create bespoke deployment implementations for a diverse array of cloud services. Consequently, this mitigates the limitations of Terraform's core functionality with respect to the specialized needs of quantum computing service deployment. The Terraform custom provider encompasses several crucial components that facilitate seamless interaction with cloud services:

1. Provider API client definition: This component serves as an access point for provider users, enabling them to input credentials or configuration settings for authenticating with the cloud service API.
2. A collection of data sources retrievable through the provider: Each data source boasts its own implementation, offering users an interface to define parameters for fetching and acquiring data.
3. An assortment of resources deployable and manageable via the provider: Each resource constitutes a distinct Golang interface, equipped with methods for creating, updating, reading, and deleting cloud-based resources.

From the capabilities provided by the Terraform custom provider, a comprehensive implementation plan has been devised to facilitate AWS Braket deployment, which encompasses the following components:

1. Provider API definition: This configuration incorporates the retrieval of AWS credentials, which subsequently initializes an AWS Braket API client. Additionally, the provider definition enables the specification of the region in which the API client operates, allowing for the execution of quantum tasks in various geographical locations (e.g., running a quantum task on Lucy in the eu-central-1 region, while IonQ operates in the us-east-1 region).
2. A curated list of quantum devices as data sources: This data source provides essential information regarding the status of each quantum device, indicating whether a given device is online, queued, offline, or retired.
3. A quantum circuit definition as a resource: This component outlines the characteristics and parameters of a quantum circuit to be executed on the designated quantum device.

4. A quantum task as a resource: The task specifies the circuit to be executed, the quantum device to be employed, and additional parameters such as the number of shots and the storage location for the resulting data.

By implementing these crucial elements, the Terraform custom provider effectively supports AWS Braket deployment and streamlines the management of quantum tasks on various devices and within distinct regions. The custom Terraform provider has been meticulously designed to operate in conjunction with the standard AWS Terraform provider. This collaborative approach ensures the seamless deployment of essential dependencies, such as S3 buckets and IAM roles, which are crucial for the successful execution of AWS Braket. By integrating the capabilities of both providers, users can effectively manage their AWS resources and expedite the deployment process for quantum computing services.

4.2 System data and logic flows

We outline how each component in the experiment system interact, shown in Figure 15. The Circuit Exporter component is designed to be operated manually, allowing users to have full control over the selection of quantum circuits. The primary function of this component is to export OpenQASM circuits in the form of a text file. Since the rest of the components in the system execute automatically, this manual control enables users to choose the appropriate circuit for specific tasks or requirements.

Both the Main Circuit Execution and Benchmark Quantum Device Executor components read the text file provided by the Circuit Exporter. Upon receiving the file, these components initiate a Terraform script that calls upon cloud provider APIs to deploy the quantum circuits. Each component processes its own results independently, ensuring a seamless operation. The Benchmark Quantum Device Executor identifies the best-performing quantum device and submits its result to a parameter queue. This parameter is then picked up by the Main Circuit Executor during each execution cycle. In the event that the Main Circuit Executor does not receive a new parameter, it continues to use the previously obtained value, maintaining consistent performance. Terraform scripts employed in the aforementioned components utilize both custom Terraform providers and standard cloud service providers, such as Amazon Web Services (AWS) and Microsoft Azure. The standard providers supply the prerequisite cloud resources to the custom provider, enabling cross-provider reference through Terraform's built-in support.

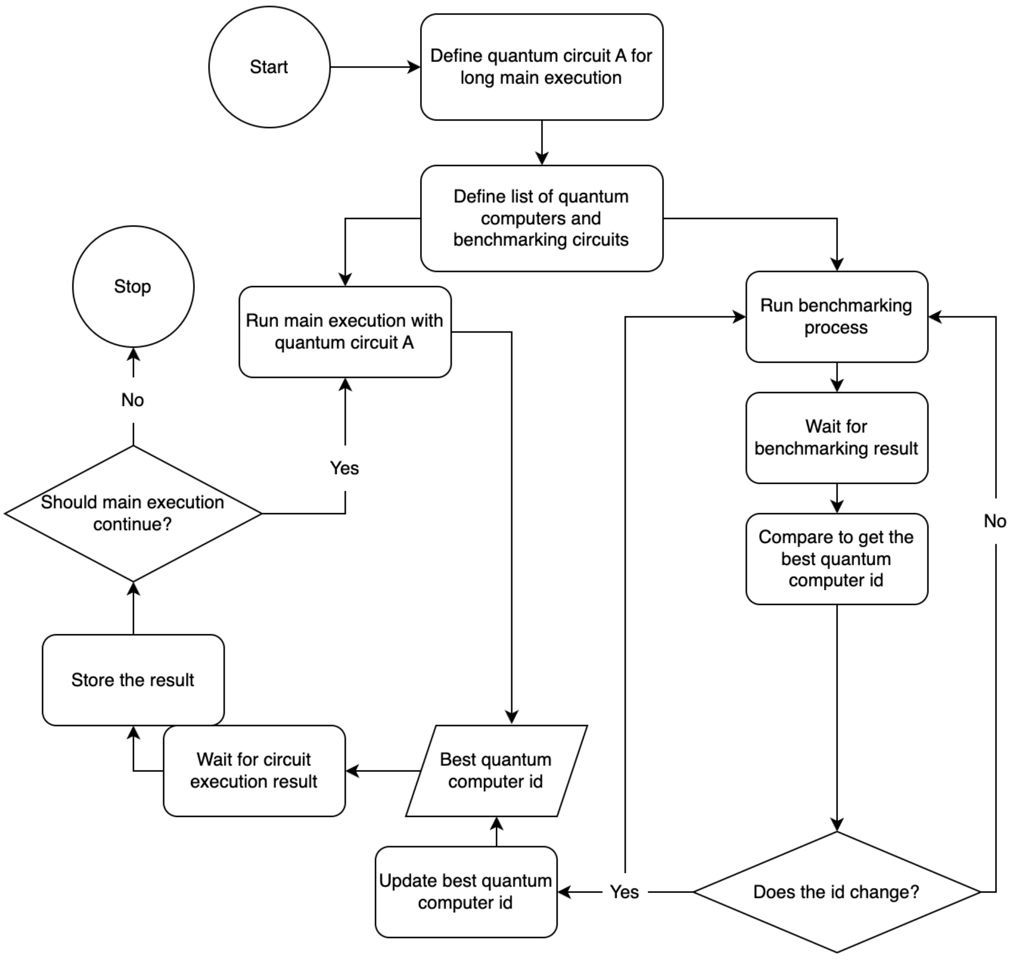

Start
Define quantum circuit A for long main execution
Define list of quantum computers and benchmarking circuits
Run main execution with quantum circuit A
Run benchmarking process
Wait for benchmarking result
Compare to get the best quantum computer id
Does the id change?
Yes
No
Update best quantum computer id
Best quantum computer id
Wait for circuit execution result
Store the result
Should main execution continue?
Yes
No
Stop

Figure 15: Logic flow of the system

5 Results and discussion

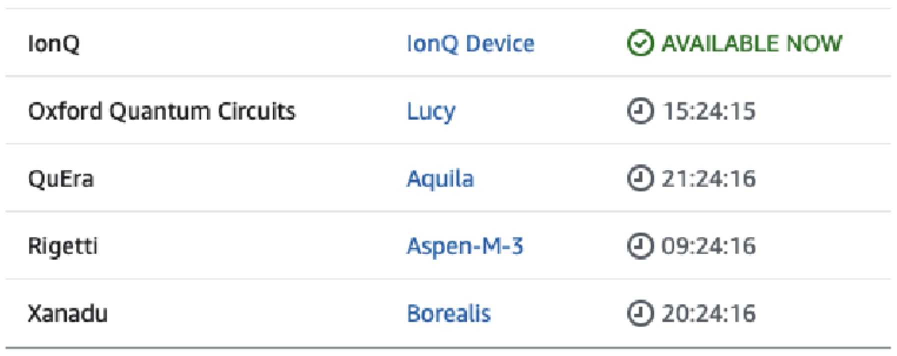
In this chapter, we present the results of the experimental setup from Chapter 4. We offer insights into the current level of maturity in cloud support for quantum computing, as well as the efficiency of execution queues and the time required for execution. Additionally, the findings encompass observations regarding the performance of the custom Terraform provider and the efficacy of the benchmark methods applied to the specified quantum computers.

We had to undertake numerous iterations and testing to successfully deploy an accurate circuit and obtain the necessary data. This is because the current state of quantum computing service support within prominent cloud computing platforms presents a multitude of challenges, such as inadequate documentation pertaining to deployment parameters and the specific regions in which the service is available. For instance, the Azure platform, at the time of writing this thesis, maintains a beta designation for their quantum computing service through both the RestAPI and CLI. This deficiency in comprehensive guidance hinders the efficient onboarding of new developers and users, compounded by the scarcity of readily available online resources such as tutorials, examples, and courses.

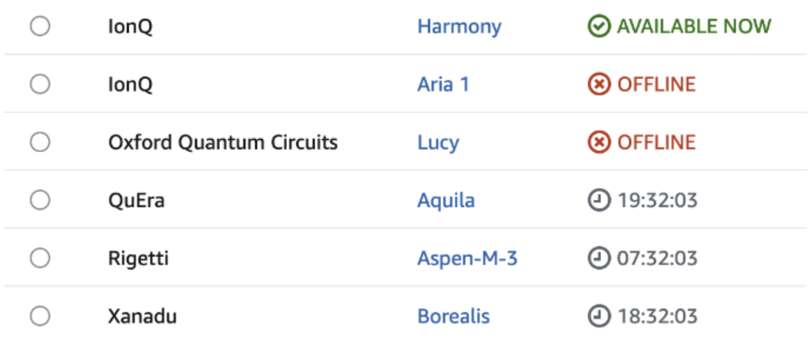
The absence of standardized methods for storing the results of quantum circuit executions contributes to the complexity of the field. The heterogeneity of quantum devices' output further complicates matters; some devices yield a probability list encompassing all states, whereas others provide the measured bit state for each shot. Additionally, the adoption of open formats for quantum circuits, such as OpenQASM, has not been universally embraced by all quantum devices, leading to a lack of consensus on a standardized description for quantum circuits. This discordance generates significant difficulties in establishing cross-platform support for quantum circuits, as it necessitates the development of circuit translators to bridge the gap between differing formats. Such a task is both laborious and unscalable, further underscoring the need for a unified approach to the implementation and management of quantum computing services within the cloud computing domain.

5.1 Execution efficiency

This section describes the effectiveness of the implementation. This consists of the execution time, the execution queue on quantum computers, and the CPU architecture support for the custom Terraform provider. In addition, the section discloses implementation flaws and proposed solutions for future works.



IonQ	IonQ Device	✓ AVAILABLE NOW
Oxford Quantum Circuits	Lucy	15:24:15
QuEra	Aquila	21:24:16
Rigetti	Aspen-M-3	09:24:16
Xanadu	Borealis	20:24:16



○	IonQ	Harmony	✓ AVAILABLE NOW
○	IonQ	Aria 1	⊗ OFFLINE
○	Oxford Quantum Circuits	Lucy	⊗ OFFLINE
○	QuEra	Aquila	19:32:03
○	Rigetti	Aspen-M-3	07:32:03
○	Xanadu	Borealis	18:32:03

Figure 16: Snapshots of the execution queue in hours in Amazon Braket service

5.1.1 Execution queue on quantum computers

At the time of composing this thesis, it is noteworthy that the execution queue for each quantum processing unit (QPU) tends to be lengthy and exhibits a lack of consistency between different units. This inconsistency ultimately results in protracted execution durations, posing challenges for efficient quantum computing. Furthermore, the frequent offline status of some quantum devices significantly hampers the effectiveness of circuit execution, creating additional obstacles in the pursuit of optimal performance in this rapidly developing field.

During implementation, it is noticed that the processing queue shown on the cloud service's user interface (UI) does not match the real time spent waiting. During the completion of the implementation, there have been times when the quantum job was done both earlier and later than the time shown on the queue.

5.1.2 Efficiency on running the task loop

In the context of energy efficiency, executing the task loop on a personal computer proves to be suboptimal, as the majority of the process involves waiting for results. Consequently, transitioning the loop to a cloud service that utilizes serverless execution appears to be a more prudent approach, as it enables processing of the quantum device results only when they are readily available. However, it is essential to acknowledge the limited cross-cloud support in this regard, which introduces its own set of challenges to be addressed in order to fully capitalize on the potential benefits of such an approach in the realm of quantum computing.

5.1.3 Custom Terraform provider

The Terraform provider, as it currently stands, solely supports the x86 architecture [20], which subsequently necessitates emulation when implementing the provider on an ARM platform. This reliance on emulation inevitably results in diminished performance, as the process introduces additional computational overhead.

Standard Terraform providers do not currently provide precompiled executable binaries for this architecture. Therefore, the use of Terraform in arm64 environments requires the compilation of the source code from scratch, an additional, non-trivial phase that increases the complexity of the implementation setup procedure. This circumstance necessitates additional time and resources, as well as a heightened degree of caution. Each time the configuration is introduced to a new computing environment, meticulous care must be taken to ensure that the compiled source code is completely compatible and performs as expected. This situation highlights the imperative need for more universal support in Terraform's offerings, specifically to facilitate the adoption and administration of arm64 architecture systems, which are becoming increasingly popular.

There exists an identifiable anomaly within the Azure Command Line Interface (CLI): the missing reference to the azure.storage.blob package during quantum task submissions. Awaiting a formal resolution may be advisable. Meanwhile, a temporary solution exists, particularly relevant for MacOS users. By running az –version, one can ascertain the CLI's Python installation location, for example, /opt/homebrew/Cellar/azure-cli/2.46.0/libexec/bin/python. From this, one can invoke the corresponding Python's pip, typically found at /opt/homebrew/Cellar/azure-cli/2.46.0/libexec/bin/pip3, to install the absent azure.storage.blob package. Unfortunately, the current state of Azure CLI for quantum job submissions is severely impaired. It does not respect the CLI flag override, thereby necessitating the presetting of the default workspace, resource group, and location before initiating a job submission. Further exacerbating the situation is the conspicuous lack of comprehensive documentation detailing supported circuit and output formats. As a result, users face unanticipated input formats during job submissions, leading to execution challenges.

5.1.4 Qiskit circuit transpiler

During the transpilation process, an observed issue arises within the Qiskit transpiler, whereby it fails to recognize that the CNOT and CX gates are, in fact, equivalent. This oversight manifests as a limitation, as the transpiler throws an exception when the basis gate set does not encompass both CNOT and CX gates, even though they represent the same quantum operation. Addressing this bug is essential in order to ensure the accurate and efficient transpilation of quantum circuits, thereby facilitating a more robust framework for quantum computing applications.

5.1.5 Errors handling

Given the variety of potential error occurrences, the current implementation's lack of an error-handling mechanism is glaring and a significant shortcoming. Errors can arise from a variety of sources, with network disruptions being a prominent example that leads to the unsuccessful execution of API calls to the cloud provider. In addition, as long as the API remains in beta, cloud providers retain the right to modify response values within API requests. Consequently, a more robust system for managing errors is required. This should include comprehensive monitoring and graceful system termination protocols to guarantee that abrupt, unanticipated changes do not result in catastrophic system failures or data loss. These enhancements would unquestionably increase the system's stability and dependability.

5.2 Benchmark formula results

In accordance with the benchmarking formulas from Chapter 4, a consistent outcome has been observed when assessing the fidelity of the three distinct quantum circuits delineated in Section 3.2. The results are comprehensively displayed in Figure 17 and Figure 18.

Two figures list the number of qubits present in each circuit. The selection of quantum circuit and number of qubits in this instance are random. When benchmarking the performance of the quantum circuits execution across different quantum machines, IonQ device has consistently performed better than the other two in both metrics. It is difficult to determine why IonQ performs better than the other two devices without comparing the noise and error at the hardware level.

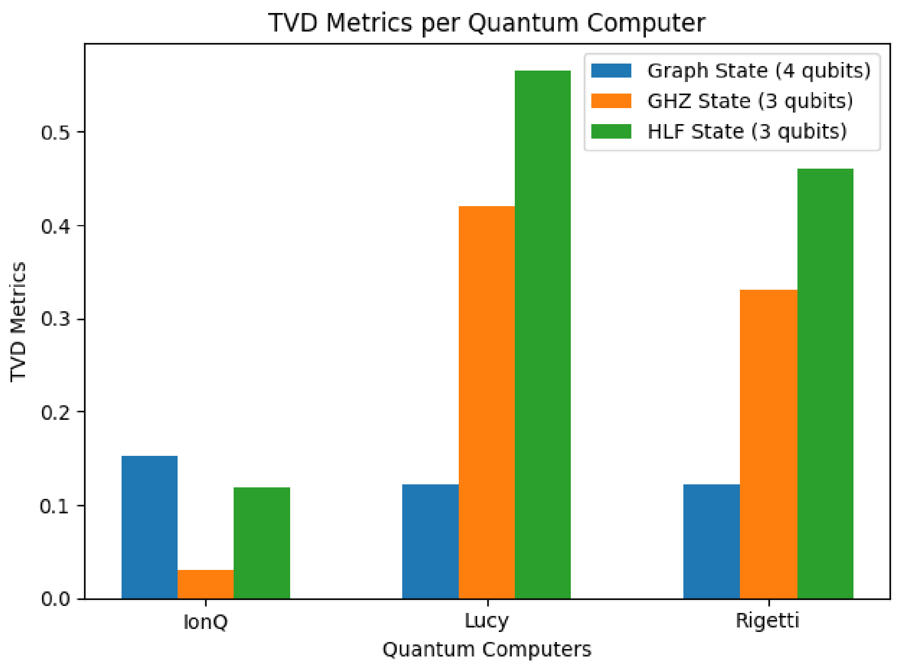


Figure 17: TVD metrics per quantum machines

Taking a closer look at one quantum computer, the Rigetti Aspen M-3, since it has the shortest execution queue, both the Total value distance metrics and the Hellinger fidelity give the same answer: the more qubits are used in a circuit, the less close the actual result is to what was expected. Those metrics are presented in Figure 19 and Figure 20. Interestingly, despite the fact that the GHZ circuit has fewer gates than the Graph State circuit, the GHZ circuit's fidelity metrics are inferior to those of Graph State circuit.

5.3 Execution cost

In determining the economic viability of quantum circuit execution, the number of rounds, or repetitive measurements of a quantum circuit, necessary to produce meaningful results is a crucial factor. The cost of operating quantum circuits is directly proportional to the number of these executions; consequently, it is crucial to select the optimal number of shoots in order to control expenditures without compromising the quality of the results.

A task in AWS Braket is operationally defined as a single deployment of a quantum circuit, executed a given number of shots (n). Each of these tasks incurs a certain cost that is calculated based on a formula:

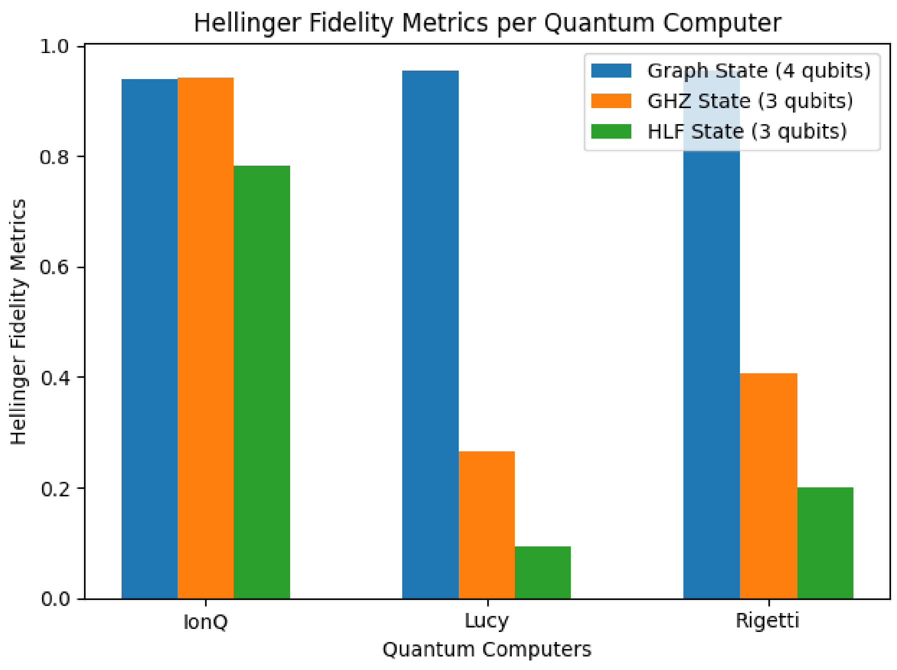


Figure 18: Hellinger fidelty per quantum machines

$$C = t + n * s$$

In this equation, t represents the base price of deploying a task, n is the quantity of shots, and s is the cost per shot. Figure 21 visualizes the cost to execute a single quantum task in 3 quantum computers used in the benchmarking.

Although the cost is not the focus of this thesis, represents a potential financial barrier for practical applications. It requires striking a balance between cost-effectiveness and the pursuit of statistically significant results.

	Per-task price	**Per-shot price**	**Per-minute price**
IonQ	\$0.3	\$0.03	
Lucy	\$0.3	\$0.00035	
Aspen-M-3	\$0.3	\$0.00035	
AWS Braket Simulators			\$0.075-\$0.275

Table 2: Execution cost on the quantum computers provided by AWS Braket used in benchmarking

The pricing strategy for quantum computing provided by the Azure Quantum service is distinct. Using the same Rigetti Aspen M-3 machine as an illustration, the price per quantum task is proportional to the quantum computer's execution time. However, since we were unable to effectively deploy a circuit for execution in the

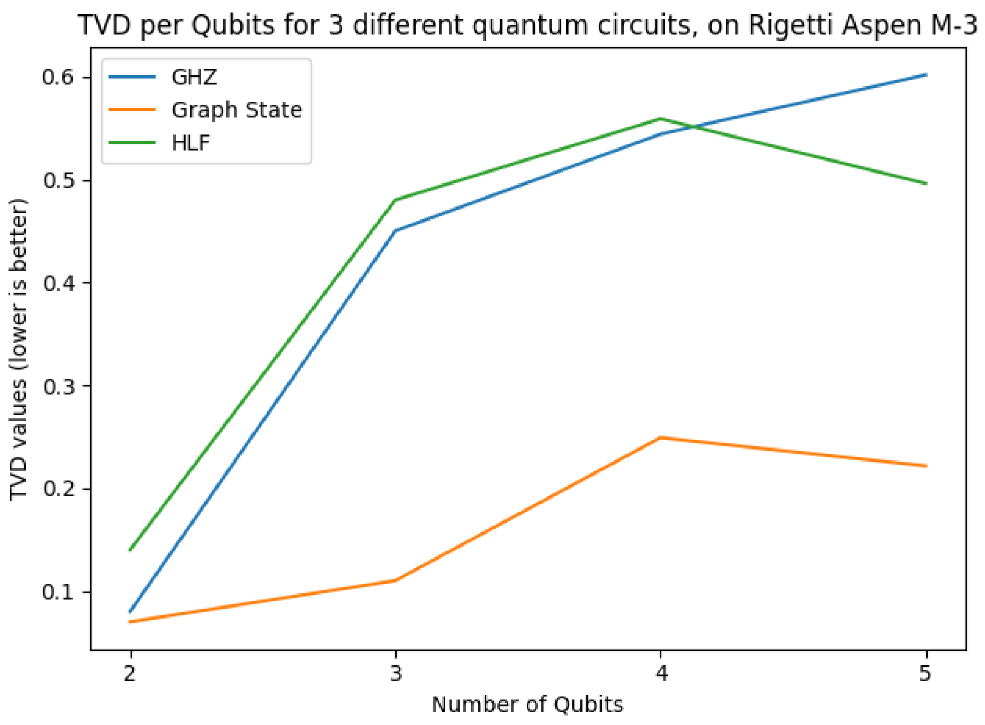


Figure 19: TVD metrics per number of qubits

implementation, we cannot compare the costs of AWS and Azure.

When considering simulators, the cost metric takes a different perspective. Here, the cost corresponds to the time required for the simulator to execute the quantum circuit. Simulator costs are more predictable, providing a platform for initial testing that is more cost-effective. This is especially true for the Microsoft Azure Quantum service, since all the quantum simulations are free to use at the time this thesis was written. The value of cloud-based services such as Amazon Web Services (AWS) and Microsoft Azure is rising in this context. These platforms facilitates preliminary circuit testing prior to deployment on actual QPUs. This strategy of conducting 'trial runs' on simulators could be instrumental in reducing the costs associated with quantum circuit execution and accelerating the transition of quantum computing from academic theory to practical, real-world applications.

5.4 Conclusion and future works

The primary objective of this research was the development of a custom Terraform provider to enhance the efficiency of deploying quantum circuits while concurrently conducting benchmarking calculations to ascertain the most effective quantum device, whose results closely align with simulated expected values. Moreover, the study

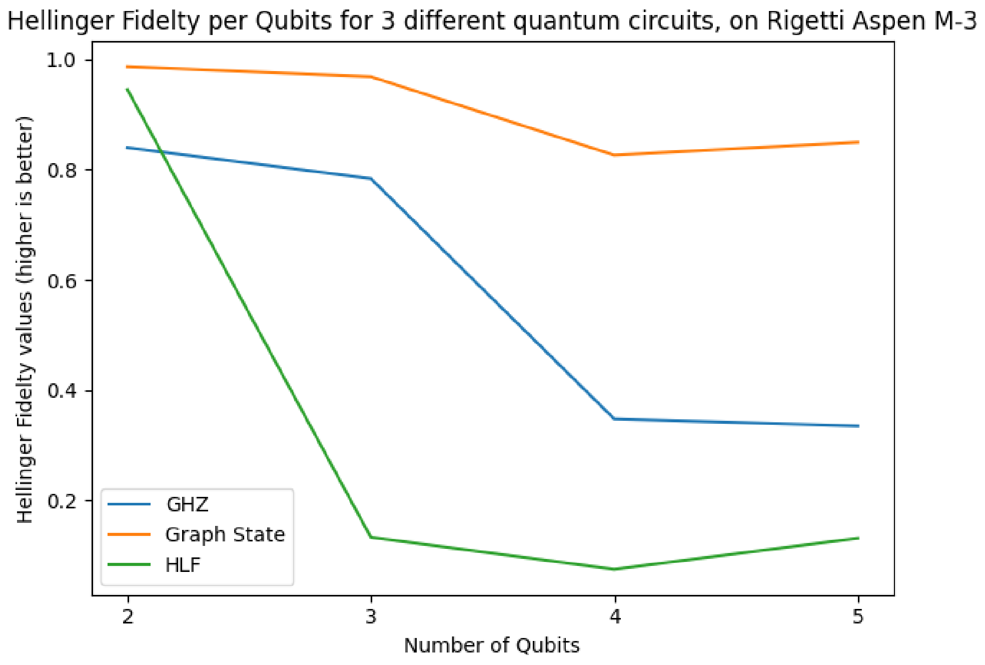


Figure 20: Hellinger fidelty per number of qubits

includes a proof of concept script designed to execute a main quantum circuit on a quantum device, as determined by a separate process.

Despite encountering limitations in cloud service support and experiencing slow, inefficient execution times of quantum computers, the project successfully demonstrated the deployment of various quantum circuits through a unified interface, while addressing the challenges faced and offering solutions for their mitigation.

Furthermore, the utilization of infrastructure as code significantly enhanced the quality of quantum circuit execution, particularly in terms of maintainability and security. This research, therefore, serves as a valuable contribution to the ongoing development and optimization of quantum computing deployment and benchmarking methodologies.

The research carried out in this thesis has established a pioneering framework, thereby opening a plethora of avenues for refining and augmenting the setup. Notably, the thesis managed to successfully deploy and benchmark quantum computers offered by Amazon Web Services (AWS). However, the implementation pertaining to Microsoft Azure encountered challenges primarily due to the insufficiency of documentation, inhibiting a successful execution. Future endeavors, building upon the foundational work of this thesis, could strive to expand the deployment to include Microsoft Azure and other cloud services, thereby amplifying the reach and utility of

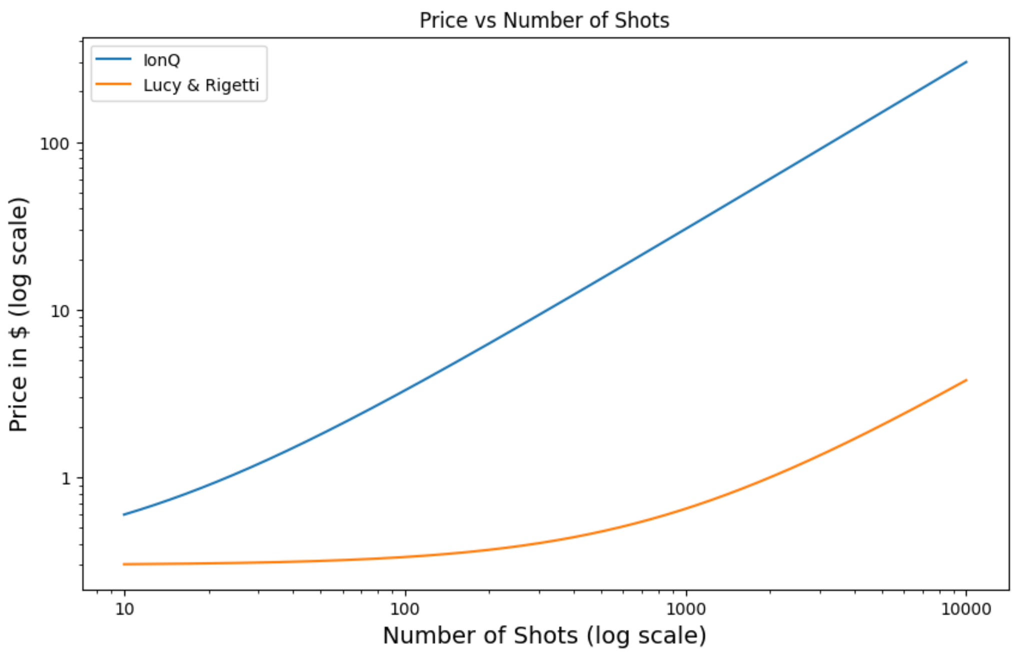


Figure 21: Execution Cost for IonQ, Lucy and Rigetti quantum computers in AWS Braket with relations to number of shots

quantum computing applications.

One significant area identified for future enhancement pertains to the quantum circuit transpiler implemented in the thesis, which, in its current form, has limited capabilities. The transpiler supports the output of OpenQASM exclusively, deploying a single static output to all supported quantum computers. This scope could be broadened in future work to allow dynamic output adjustment to a circuit, and also incorporate support for additional quantum circuit formats, hence offering a more adaptable and versatile transpiler.

Moreover, the benchmarking and execution of the setup, which presently rely on a local computer, could be migrated to a cloud service in subsequent iterations. This transition would circumvent the requirement of a perpetually operating local computer, offering a more scalable and efficient solution.

The infrastructure state instantiated by Terraform currently lacks secure storage and is instead retained within the local machine. This scenario poses a significant hindrance to system portability, particularly when multiple developers seek concurrent utilization of the same system for deploying distinct experiments. Due to the lack of a synchronization mechanism for the state, potential conflicts could arise, leading to phenomena such as deadlocks and unpredictable system behaviours. Potential solutions for centralized state management within Terraform could be implemented using services such as Amazon Web Services (AWS) DynamoDB or AWS Simple Storage Service (S3).

Lastly, considering the extended processing times innate to quantum computing, the implementation of a notification system could substantially enhance the usability of the setup. Currently, there is no mechanism to alert users when a benchmarking task has completed. Incorporating an email notification system would enable users to be promptly informed upon the completion of a benchmark, facilitating a more proactive response. Such additions would greatly augment the user experience and increase the practicality of the system, ultimately contributing to the broader accessibility and effectiveness of quantum computing.

References

[1] S. Achar. Enterprise saas workloads on new-generation infrastructure-as-code (iac) on multi-cloud platforms. *Global Disclosure of Economics and Business*, 10(2):55–74, 2021.

[2] R. Akond, M. Rezvan, and W. Laurie. A systematic mapping study of infrastructure as code research. *vol.*, 108:65–77, 2019.

[3] B. Anne and K. Elham. Parallelizing quantum circuits. *vol.*, 410:2489–2510, 2009.

[4] A. Bayerstadler, G. Becquin, J. Binder, T. Botter, H. Ehm, T. Ehmer, M. Erdmann, N. Gaus, P. Harbach, M. Hess, J. Klepsch, M. Leib, S. Luber, A. Luckow, M. Mansky, W. Mauerer, F. Neukart, C. Niedermeier, L. Palackal, R. Pfeiffer, C. Polenz, J. Sepulveda, T. Sievers, B. Standen, M. Streif, T. Strohm, C. Utschig-Utschig, D. Volz, H. Weiss, and F. Winter. Industry quantum computing applications. *EPJ Quantum Technology*, 8(1):25, 2021. Number: 1 Publisher: Springer Berlin Heidelberg.

[5] Alexander Benlian, Thomas Hess, and Peter Buxmann. Drivers of SaaS-adoption – an empirical study of different application types. *Business & Information Systems Engineering*, 1(5):357–369, 2009.

[6] F. G. S. L. Brandão, A. W. Harrow, and M. Horodecki. Local random quantum circuits are approximate polynomial-designs. *vol.*, 346:397–434, 2016.

[7] Gilles Brassard, Isaac Chuang, Seth Lloyd, and Christopher Monroe. Quantum computing. *Proceedings of the National Academy of Sciences*, 95(19):15, September 1998.

[8] Yevgeniy Brikman. *Terraform: Up and Running.* " O'Reilly Media, Inc.", 2022.

[9] Winton Brown and Omar Fawzi. Scrambling speed of random quantum circuits, 2012.

[10] Qiskit Community. Qiskit braket plugin, URL: https://github.com/qiskit-community/qiskit-braket-provider. Accessed on 7 May 2023.

[11] Andrew Cross. The IBM Q experience and QISKit open-source quantum computing software. In *APS March Meeting Abstracts*, volume 2018 of *APS Meeting Abstracts*, page L58.003, January 2018.

[12] David P. DiVincenzo. The physical implementation of quantum computation. *Fortschritte Der Physik*, 48(9):771–83, 2000.

[13] O. Gamel. *Entangled Bloch spheres: Bloch matrix and two-qubit state space.* Phys. Rev. A, 2016.

[14] Juan Carlos Garcia-Escartin and Pedro Chamorro-Posada. Equivalent quantum circuits, 2011.

[15] Sukhpal Singh Gill, Adarsh Kumar, Harvinder Singh, Manmeet Singh, Kamalpreet Kaur, Muhammad Usman, and Rajkumar Buyya. Quantum computing: A taxonomy, systematic review and future directions. *Software: Practice and Experience*, 52(1):66–114, 2022.

[16] N Gisin and H Bechmann-Pasquinucci. Bell inequality, bell states and maximally entangled states for n qubits. *Physics Letters A*, 246(1):1–6, 1998.

[17] Constantin Gonzalez. Cloud based QC with Amazon Braket. *Digitale Welt*, 5(2):14–17, 2021.

[18] Teresa Guarda, Washington Torres, and Maria Fernanda Augusto. The impact of quantum computing on businesses. In Osvaldo Gervasi, Beniamino Murgante, Sanjay Misra, Ana Maria A. C. Rocha, and Chiara Garau, editors, *Computational Science and Its Applications – ICCSA 2022 Workshops*, Lecture Notes in Computer Science, pages 3–14. Springer International Publishing, 2022.

[19] Inc HashiCorp. CDK for terraform, URL: https://developer.hashicorp.com/terraform/cdktf. Accessed on 14 Feb 2023.

[20] Inc HashiCorp. Terraform documentation, URL: https://developer.hashicorp.com/terraform/docs. Accessed on 14 Feb 2023.

[21] Johnny Hooyberghs. Azure quantum. In Johnny Hooyberghs, editor, *Introducing Microsoft Quantum Computing for Developers: Using the Quantum Development Kit and Q#*, pages 307–339. Apress, 2022.

[22] Ryszard Horodecki, PawełHorodecki, MichałHorodecki, and Karol Horodecki. Quantum entanglement. *vol.*, 81(2):865–942, June 2009.

[23] Zhi-Xiang Jin and Shao-Ming Fei. Quantifying quantum coherence and nonclassical correlation based on hellinger distance. *Phys. Rev. A*, 97:062342, Jun 2018.

[24] N. Khammassi, I. Ashraf, J. V. Someren, R. Nane, A. M. Krol, M. A. Rol, L. Lao, K. Bertels, and C. G. Almudever. Openql: A portable quantum programming framework for quantum accelerators. *ACM Journal on Emerging Technologies in Computing Systems*, 18:1, 2021.

[25] E. Knill. Quantum gates using linear optics and postselection. *Phys. Rev. A*, 66:052306, Nov 2002.

[26] G. Leonid. Classical complexity and quantum entanglement. *vol.*, 69(3):448–484, November 2004.

[27] Chi-Kwong Li, Rebecca Roberts, and Xiaoyan Yin. Decomposition of unitary matrices and quantum gates. *International Journal of Quantum Information*, 2013. Publisher: World Scientific Publishing Company.

[28] Thomas Lubinski, Cassandra Granade, Amos Anderson, Alan Geller, Martin Roetteler, Andrei Petrenko, and Bettina Heim. Advancing hybrid quantum–classical computation with real-time execution. *Frontiers in Physics*, 10, 2022.

[29] Peter Mell, Tim Grance, et al. The NIST definition of cloud computing, 2011.

[30] R. Modi. Infrastructure as code. In *Deep-Dive Terraform on Azure.* Apress Berkeley, CA, 2021.

[31] Thien Nguyen, Dmitry Lyakh, Eugene Dumitrescu, David Clark, Jeff Larkin, and Alexander McCaskey. Tensor network quantum virtual machine for simulating quantum circuits at exascale. *ACM Transactions on Quantum Computing*, 4:1, 2022.

[32] Shuntaro Okada, Masayuki Ohzeki, Masayoshi Terabe, and Shinichiro Taguchi. Improving solutions by embedding larger subproblems in a d-wave quantum annealer. *Scientific Reports*, 9(1):2098, 2019. Number: 1 Publisher: Nature Publishing Group.

[33] Openqasm. Quantum assembly language for extended quantum circuits, URL: https://github.com/openqasm/openqasm.

[34] Rajiv Pandey, Pratibha Maurya, Guru Dev Singh, and Mohd. Sarfaraz Faiyaz. *Evolutionary Analysis: Classical Bits to Quantum Qubits*, pages 115–129. Springer Nature Singapore, Singapore, 2023.

[35] Arun K. Pati. Existence of the schmidt decomposition for tripartite systems. *vol.*, 278:118–122, 2000.

[36] Dong Pyo Chi, Jeong San Kim, and Soojoon Lee. Quantum algorithms without initializing the auxiliary qubits. *Phys. Rev. Lett.*, 95:080504, Aug 2005.

[37] Qiskit. Qiskit website, URL: https://qiskit.org/. Accessed on 7 May 2023.

[38] A. Rahman, E. Farhana, and L. Williams. The 'as code' activities: development anti-patterns for infrastructure as code. *Empir Software Eng*, 25:3430–3467, 2020.

[39] Gokul Subramanian Ravi, Kaitlin N. Smith, Pranav Gokhale, and Frederic T. Chong. Quantum computing in the cloud: Analyzing job and machine characteristics. In *2021 IEEE International Symposium on Workload Characterization (IISWC*, pages 39–50, 2021.

[40] Gokul Subramanian Ravi, Kaitlin N. Smith, Prakash Murali, and Frederic T. Chong. Adaptive job and resource management for the growing quantum cloud. In *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 301–312, 2021.

[41] R. Sadegh, W. Nathan, and C. S. Barry. Quantum-circuit design for efficient simulations of many-body quantum dynamics. *vol.*, 14, 2012.

[42] S. Sciara, F. R. Lo, and G. Compagno. Universality of schmidt decomposition and particle identity. *Sci Rep*, 7, 2017.

[43] Enaul haq Shaik and Nakkeeran Rangaswamy. Implementation of quantum gates based logic circuits using ibm qiskit. In *2020 5th International Conference on Computing, Communication and Security (ICCCS)*, pages 1–6, 2020.

[44] Harpreet Singh and Abha Sachdev. The quantum way of cloud computing. In *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*, pages 397–400, 2014.

[45] Daniel Sokolowski. Infrastructure as code for dynamic deployments. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2022, page 1775–1779, New York, NY, USA, 2022. Association for Computing Machinery.

[46] Anthony T Velte, Toby J Velte, Robert C Elsenpeter, and Robert C Elsenpeter. *Cloud computing: a practical approach.* McGraw-Hill New York, 2010.

[47] K. Venkateswaran. *Fundamentals of Quantum Computing.* Springer Cham, 2022.

[48] Alexander Yu Vlasov. On symmetric sets of projectors for reconstruction of a density matrix, 2003.

[49] Dave Wecker and Krysta M. Svore. Liqui| : A software design architecture and domain-specific language for quantum computing, 2014.

[50] B. Weder, J. Barzen, F. Leymann, and M. Salm. Automated quantum hardware selection for quantum workflows. *vol.*, 10, 2021.

[51] Benjamin Weder, Uwe Breitenb"ucher, Frank Leymann, and Karoline Wild. Integrating quantum computing into workflow modeling and execution. In *2020IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC*, pages 279–291, 2020.

[52] Robert Wille, Rod Van Meter, and Yehuda Naveh. Ibm's qiskit tool chain: Working with and developing for real quantum computers. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1234–1240, 2019.

[53] Marcus Woo. What is a quantum computer? *Engineering and Science*, 76(1):20–25, 2013.

[54] William K. Wootters and Wojciech H. Zurek. The no-cloning theorem. *Physics Today*, 62(2):76–77, 02 2009.

[55] Will Zeng, Blake Johnson, Robert Smith, Nick Rubin, Matt Reagor, Colm Ryan, and Chad Rigetti. First quantum computers need smart software. *Nature*, 549(7671):149–151, 2017. Number: 7671 Publisher: Nature Publishing Group.