CHAPTER

Emerging clouds 13

In this chapter, we explore several areas of cloud computing research impacting the clouds of the future. Past history is often a good predictor of the future, and the past shows that cloud computing has promptly absorbed and exploited new developments in science and engineering for the benefit of an increasingly larger user community. At the same time, cloud computing has forced significant revisions of traditional concepts and ideas in many areas of computer science and computer engineering. Thus, it seems safe to expect that these processes will continue in predictable, as well as unpredictable, ways.

Arguably, Google Research is the most reliable and active source of cloud computing research publications. To identify the most disruptive ideas affecting cloud computing, we compared the publications of Google researchers over the last five years in two traditional areas, Hardware and Architecture (HA) and Data Mining (DM) with three emerging areas, Machine Intelligence (MI), Machine Perception (MP), and Quantum Computing (QC). Table 13.1 indicates that the interests of Google researchers have evolved with fewer publications about traditional areas and an increasing number of publications in AI/ML and QC.

The chapter starts with a brief discussion of the obvious challenges cloud computing is expected to address in Section 13.1. Artificial Intelligence (AI) and Machine Learning (ML) are the most impactful areas of research for virtually all aspects of human activities requiring computing. Quantum computing provides a glimmer of hope and, hopefully, a practical alternative to silicon as the effective end of Moore's law is in sight. The data in Table 13.1 justify why AI and ML applications and quantum computing on the cloud are covered next, in Sections 13.2 and 13.3.

Thousands of cars in parking garages or airport parking lots can contribute to edge cloud computing if their owners are offered adequate incentives. Vehicular clouds is a topic of practical interest because powerful systems on a chip are already used in cars to support advanced safety features and Level 0 ("basic safety"), 1 ("hands on"), 2 ("hands off"), using the SAE classification of driving automation

Table 13.1 The number of papers published by Google researchers in two traditional areas, Hardware and Architecture (HA) and Data Mining (DM) and three emerging areas, Machine Intelligence (MI), Machine Perception (MP), and Quantum Computing (QC), during the period January 2015–July 2020 and ALL, i.e., since Google was founded in 1998.

Area	2015	2016	2017	2018	2019	2020	All
HA	6	7	5	13	7	3	99
DM	22	21	18	24	32	14	284
MI	100	169	270	386	498	189	2 051
MP	40	57	99	158	151	58	807
QC	3	7	8	14	11	6	56

Cloud Computing. https://doi.org/10.1016/B978-0-32-385277-7.00020-8

levels. Prof. Stefan Olariu, one of the pioneers of this field, has graciously consented to overview this topic in Section 13.4. Vehicular clouds support an alternative type of edge computing.

13.1 A short-term forecast

Cloud functionality is evolving as new services and more diverse and powerful instance types are released every year. For example, Amazon introduced Lambda, a service in which applications are triggered by user-defined conditions and events. In late 2016, AWS added P2, a powerful, scalable instance with GPU-based parallel compute capabilities and a range of services supporting machine learning and AI. Google has been adding to the software stack for coarse-gram parallelism based on MapReduce and adding TPUs (Tensor Processing Units) to its cloud infrastructure. IBM's efforts target computational intelligence as evidenced by the success of Watson in healthcare and data analytics. Microsoft is attempting to extend the range of commercial applications supported by its growing cloud infrastructure.

The cloud research community will most likely be faced with new and challenging problems in the future. *Variability* and *conflicting requirements* are disruptive qualities of cloud computing demanding new thinking in system design. The physical infrastructure consisting of servers with different architecture and performance is constantly updated as solid-state technologies and processor architecture evolve. New software orchestrating a coherent system view is developed every month, and the depth of the software stack is continually increasing. If cloud computing is to continue to be successful, it is very likely that new applications will emerge. The size of the cloud user population and the diversity of their requirements will grow.

Conflicting system design requirements are not a novelty, but the depth and the breadth of such conflicts is unprecedented and qualitatively different due to scale of the cloud infrastructure and application diversity. Many contradictory requirements in the design of computer clouds have to be carefully balanced. For example, resource sharing is a basic design principle, yet strict performance and security isolation of cloud application are also critical. To deliver cheap computing cycles, the cloud infrastructure should maximize resource utilization, while sufficient resources should be kept in reserve to respond to large workload spikes. The system as a whole should present itself as flawless and extremely reliable though the failure rate of cheap, off-the-shelf hardware components can be fairly high. Performance guarantees should be provided, while a mix of workloads with very different requirements will continue to dynamically share system resources.

Unquestionably, computer clouds will continue to evolve, but how? A parallel of clouds with the Internet is unavoidable. Initially, the Arpanet, the precursor of the Internet, was a network used to transfer data files from one location to another. It was designed as a best-effort network doing its best to transport data packets without providing end-to-end delivery guarantees. Support for communication with real-time delivery constraints was not foreseen. The Internet's success forced changes, and, as a result, today's Internet supports low-latency and high-bandwidth data streaming. Data-streaming traffic is *shaped*[1] to guarantee that routers have enough resources to transmit a continuous stream of data with low jitter.

[1] Traffic shaping is a bandwidth management technique used on datagram networks that delays some or all datagrams to bring them into compliance with a desired traffic profile.

How will computer clouds simultaneously provide QoS guarantees, increase resource utilization, support elasticity, and be more secure? This is the question. Cloud evolution poses fundamental questions that deserve further research. A first question is whether applications with special requirements, such as real-time constraints or applications exhibiting fine-grained parallelism, could migrate to the cloud. Such a migration requires changes in software, in particular, in resource management and scheduling components of the software stack. At the same time, the hardware and, in particular, the cloud interconnection networks have to offer lower latency and higher bandwidth.

Another question is how to reduce cost by increasing resource utilization, without affecting the QoS promises of cloud computing. It is self-evident that cloud elasticity cannot be supported without some form of over-provisioning which implies lower average resource utilization. In conclusion, alternative means to reduce cost should be considered.

A solution practiced by AWS and others is to combine a reservation system with spot allocations. Spot allocations are designed to consume excess resources if and when such resources are available. Cloud users with a good understanding of the resources needed and the time required by their applications should use the reservation system. They will benefit from QoS guarantees and pay more for cloud services. The other cloud users should compete for lower-cost spot allocations.

An alternative that may be explored in the future is to profile the cloud users using large databases of historic data and using machine learning algorithms and data analytics to predict the resource needs and the time required to carry out the computation. Once this information is available, a virtual private cloud that best fits the profile of the application should be configured. Another alternative is to support cloud self-organization and self-management. Market-based resource allocation could be at the heart of such an approach in spite of the problems it poses [446], including auctions as suggested in [333,334].

Homogeneity of the cloud computing infrastructure was one of the fundamental design tenets. The obvious advantages of infrastructure homogeneity are simplification of resource management and lowering hardware and software maintenance costs. Also, acquiring large volumes of identical hardware components lowers the infrastructure cost.

In the last few years CSPs realized why cloud users are clamoring for heterogeneity. As a result, today's clouds have multiple types of processors and co-processors, e.g., GPUs and TPUs and clouds use solid-state disks for real-time applications. The cloud infrastructure may include data flow engines in the near future. It is also likely that islands of systems communicating through InfiniBand, or other high-performance networks will be part of the cloud computing landscape. Such cloud islands will allow fine-grained parallel scientific and engineering applications to perform well on computer clouds.

It is necessary to address the question of how to accommodate cloud heterogeneity, while preventing a dramatic increase of infrastructure cost and an increase of resource management policies and mechanisms implementing these policies. Market mechanisms, which proved so successful in dealing with a diverse set of goods and a large consumer population in a free-market economy, could provide the answer. Cloud computing will continue to have a profound influence on the very large number of individuals and institutions who are now empowered to process huge amounts of data.

13.2 Machine learning on clouds

Machine learning (ML) uses algorithms that learn to discover how to perform a specific task without being explicitly programmed to do so. ML is considered by some an area of artificial intelligence (AI),

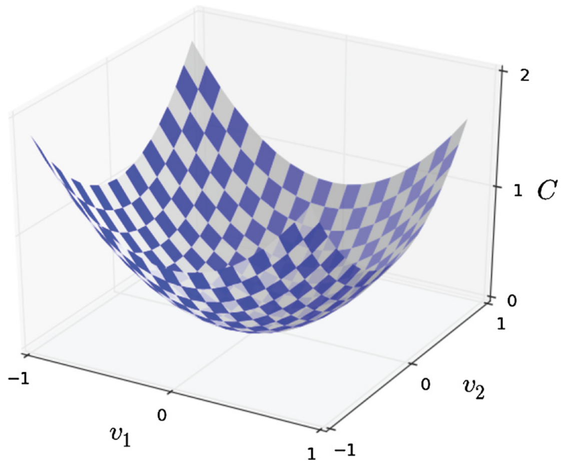


FIGURE 13.1

Graduate descent used to find a global minimum.

while others believe that ML is a field in itself. There is general agreement that ML algorithms refine a mathematical model of a system/environment using *training data* in several modes:

1. Supervised learning—the goal is to learn general rules that map inputs to outputs given example inputs and the desired output for each input provided by a "supervisor."
2. Unsupervised learning—the goal is to discover hidden patterns in input data supplied without labels or the use the unstructured input as means to achieve an end, the so-called feature learning.
3. Reinforcement learning—the goal is to perform a specific task through interactions with an environment that provides feedback after each attempt and, in the process, to maximize a rewards function.

Supervised learning uses stochastic *gradient descent* and adjusts the weights to maximize the descent of the gradient obtained by back-propagation, as shown in Fig. 13.1. *Back-propagation* repeatedly adjusts the weights of the connections in the network to minimize a measure of the difference between the actual output vector of the net and the desired output vector [425].

Given a cost function $C(v_1, v_2, \ldots)$, the change of the cost, ΔC, produced by a small $\Delta V = (\Delta V_1, \Delta V_2, \ldots)^T$ is $\Delta C \approx \nabla C \times \Delta V$, with the gradient ∇C given by the vector

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots \frac{\partial C}{\partial v_m} \right)^T . \tag{13.1}$$

Neural networks. ML uses *neural networks* (NNs), inspired by networks of human brain neurons, to learn how to accomplish a desired task. NNs were developed in the 1950s and 1960s by Frank Rosenblatt, who introduced the concept of *perceptron* [422]. Rosenblatt was inspired by earlier work by Warren McCulloch and Walter Pitts [342]. A perceptron takes several binary inputs $x_1, x_2, \ldots$ and produces a single binary output. Rosenblatt introduced weights, $w_1, w_2, \ldots$, real numbers expressing the importance of the respective inputs to the output. The binary output of a perceptron is determined

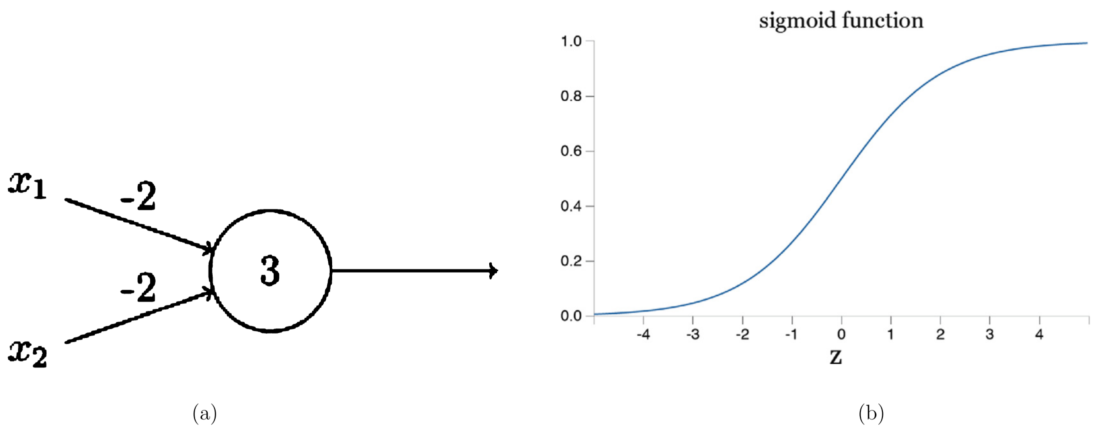


FIGURE 13.2

(a) A perceptron with two inputs, each with weight -2 and a bias of 3, can simulate a NAND gate. (b) A sigmoid function.

by the *weights* and a real number, the *threshold*, as follows:

$$\text{output} = \begin{cases} 0 & \text{if } \sum_i w_i x_i \leq \text{ threshold} \\ 1 & \text{if } \sum_i w_i x_i > \text{ threshold.} \end{cases} \tag{13.2}$$

The *perceptron rule* can be expressed using a *bias*, b:

$$\text{output} = \begin{cases} 0 & \text{if } \left(\sum_i w_i \times x_i\right) + b \leq \text{ threshold} \\ 1 & \text{if } \left(\sum_i w_i \times x_i\right) + b > \text{ threshold.} \end{cases} \tag{13.3}$$

A perceptron can be used to implement a NAND gate; NAND are *universal gates*; therefore the perceptron is also universal, i,e., one can implement any Boolean function using only perceptrons. To convince ourselves, consider a perceptron with two inputs, each with weight -2 and a bias of 3, as in Fig. 13.2(a). The input 00 produces output 1 as $0 \times (-2) + 0 \times (-2) + 3 = 3$ is positive. Similar calculations show that the inputs 01 and 10 produce output 0. But the input 11 produces output 0 as $1 \times (-2) + 1 \times (-2) + 3 = -1$ is negative.

A *sigmoid neuron* (SN) is a perceptron with the weights vector $w = (w_1, w_2, \ldots)$, the input vector $x = (x_1, x_2, \ldots)$, and a bias b. The output of an SN is $\sigma = (w \times x + b)$, rather than 0 or 1, where σ is a nonlinear activation function, the sigmoid function depicted in Fig. 13.2(b), and defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad \text{or} \quad \sigma(z) = \frac{1}{1 + exp(-\sum_i w_i \times x_i - b)}. \tag{13.4}$$

There are several types of neural networks, including:

1. *Multi-Layer Perceptrons (MLP).* Each new layer is a set of nonlinear functions of the weighted sum of all outputs (fully connected) from a prior one; the weighted sums reuse the weights of the previous layer.
2. *Convolutional Neural Networks (CNN).* Each layer is a set of nonlinear functions of weighted sums of spatially nearby subsets of outputs from the prior layer; the weighted sums reuse the weights of the previous layer.
3. *Recurrent Neural Networks (RNN).* Each subsequent layer is a collection of nonlinear functions of weighted sums of outputs and the previous state.
4. *Long Short-Term Memory (LSTM).* The decision what to forget and what to pass on as state to the next layer is critical. Weights are reused across time steps. Most popular RNN.

Convolutional Neural Networks are widely used multilayer networks where each layer raises the level of abstraction. For example, in computer vision, the first CNN layer recognizes horizontal and vertical lines, the second layer recognizes corners, the third layer recognizes shapes, a fourth layer recognizes features, such as tree leaves, and higher layers recognize multiple types of trees. A class project to study ML scalability is summarized in Section A.7.

Deep Neural Networks (DNNs) have multiple neuron layers and nonlinear activations functions, such as sigmoids. For example, Fig. 13.3 shows a DNN for digit recognition using a training dataset of $28 \times 28 = 784$ pixel images. There are 784 neurons in the input layer, one for each image in the training set [203].

DNN development consist of two phases: (a) definition of architecture, including: number of DNN layers, number of neurons in each layer, topology, and size of training data; (b) training and learning phase in which a back-propagation algorithm is used to determine the weights for each edge of the DNN. Once developed, a DNN enters a production phase involving testing, scoring, and running.

Google realized early on the potential impact of DNN applications and how computationally expensive DNN training is. For example, the training set size for activity recognition for an 8-layer DNN requires 10^6 videos, and the training time using 10 NVIDIA GPUs was reported to take some 30 days [232]. Training a CNN, internally labeled CNN1, with 89 layers, 100×10^6 weights, and 1 750 operations/weight requires 17.5×10^9 operations. Another CNN, CNN0 with 16 layers, 8×10^6 weights requires 2 888 operations. In 2016, Google decided to develop tensor processing units (TPUs) and attach them to some of its instances. The design goal for TPUs was to achieve at least one order of magnitude improvement versus GPUs as discussed in Section 3.6.

Clouds ML services. Amazon, Google, IBM, Microsoft, and virtually every other CSP offer a palette of ML services. There are countless applications of ML in areas as diverse as medicine, resource management, network optimizations, robotics, manufacturing, and marketing, among others.

AWS offers an impressive number of ML-related services. *SageMaker* is a fully managed service that facilitates building, training, and deploying machine learning (ML) models. *SageMaker Ground Truth* helps improve the quality of labels through annotation consolidation and audit workflows. *Kendra* is an enterprise search service powered by machine learning. *Forecast* uses machine learning to automatically discover how time series data and variables, such as product features and store locations, affect each other. *Augmented AI* makes it easy to build the workflows required for human review of ML predictions. *Polly* turns text into lifelike speech for applications that build entirely new types of speech-enabled products. *Elastic Inference* supports TensorFlow, Apache MXNet, PyTorch, and ONNX

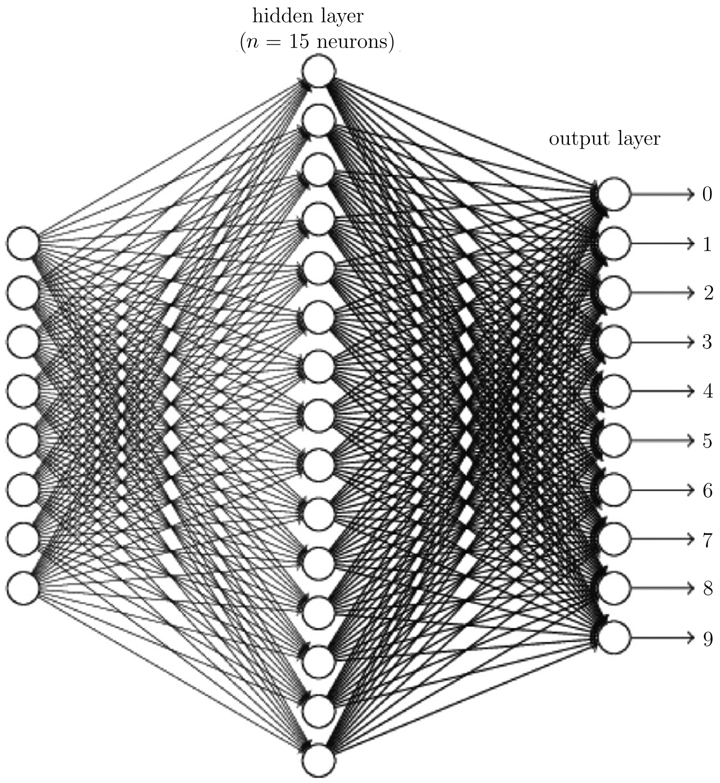


FIGURE 13.3

A DNN used to recognize digits with three layers: an input layer with 784 neurons, a hidden layer with 15 neurons, and an output layer with 10 neurons.

models; it attaches low-cost GPU-powered acceleration to Amazon EC2 and Sagemaker instances or Amazon ECS tasks and reduces the cost of running deep-learning inference by up to 75%.

AWS Deep Learning AMIs provide the infrastructure and the tools to accelerate deep learning on the cloud, at any scale. *Lex* can be use for building conversational interfaces into any application using voice and text; it provides the advanced deep-learning functionalities of automatic speech recognition (ASR) for converting speech to text, and natural language understanding (NLU) to recognize the intent of the text. *Rekognition* makes it easy to add image and video analysis to applications using deep-learning technology. Applications using this service can identify objects, people, text, scenes, and activities in images and videos, as well as detect any inappropriate content. *Comprehend* is a natural language processing (NLP) service using machine learning to discover insights and relationships in text. *Amazon Comprehend Medical* can be used to extract complex medical information from unstructured text.

The impact of AI and ML on computer architecture and implicitly on cloud computing is quite remarkable. David Patterson, the recipient of the 2017 Turing Award, shared his thoughts on Domain Specific Architectures (DSA) and the evolution of Tensor Processing Units (TPUs) at Google: "In 2013

Google calculated that if 100 million users started doing DNN three minutes per day on CPUs they would need to double the size of their data centers...Within 15 months they went from ideas to working hardware and software. The TPUv1 that Google designed had around a 80X performance per Watt of the 2015 Intel CPU and a 30X performance per Watt of the NVIDIA CPU because they were using 8-bit integer data rather than 32-bit floating point data and they dropped general purpose CPU/GPU features, which saves area and energy. Next Google created the TPUv2 that was designed to do ML training, which requires more computation, more memory, and bigger data. Google decided to build into the TPUv2 chips four Inter-Core Interconnect (ICI) links that each runs at 500 gigabits per second. Thus the links are approximately five times faster than those in a classic data center network at only one tenth of the costs. Eventually they created TPUv3 which further improved the system performance." Half a century of wisdom in designing general-purpose processors is now competing with an idea that would have been regarded an architectural blasphemy a decade ago.

13.3 Quantum computing on clouds

Most of the time, we are oblivious to the world of atomic and subatomic particles with strange, yet remarkable properties governed by the laws of quantum mechanics. The word "quantum" seems to always promise something extraordinary. Indeed, the quantum world tempts with the immense computing power generated by reversible circuits consuming little or no energy and with secure communication where an intrusion can be detected with very high probability.

Quantum theory and quantum mechanics. All started in 1900 when Max Plank, professor at the Friedrich-Wilhelms-Universität in Berlin, postulated that electromagnetic energy could be emitted only in quantized form $E = h \times \nu$, with h Planck's constant and ν the frequency of the radiation. The discovery of energy quanta won him the 1918 Nobel Prize in Physics.

Albert Einstein's elucidation of the photoelectric effect in 1905 confirmed Plank's hypothesis, and *quantum theory* was universally accepted. Einstein was rewarded with the 1921 Nobel Prize in Physics. Louis de Broglie developed the theory of electron waves in 1924. The *wave-particle duality principle* of de Broglie was soon followed by *quantum mechanics*, which describes the transformation and evolution of quantum systems.

Werner Heisenberg, Max Born, and Pasqual Jordan proposed the matrix formulation of quantum mechanics in 1925, and Erwin Schrödinger introduced the wave function that carries his name later the same year. In 1927, Heisenberg formulated the *Uncertainty Principle* asserting a fundamental limit to the precision the values of certain pairs of physical quantities characterizing a particle, such as position, x, and momentum, p, that can be predicted from initial conditions. Louis de Broglie, Werner Heisenberg, Erwin Schrödinger, and Max Born received the Nobel Prize in Physics in 1929, 1932, 1933, and 1954, respectively.

Quantum Information Processing. In 1982, Richard Phillips Feynman, who received the 1965 Nobel Prize in Physics for contributions to quantum electrodynamics, envisioned the development of a quantum computer and argued that only a quantum computer could *exactly simulate a quantum system*. In spite of the tremendous progress in quantum information processing during the past three decades, access to quantum computers is limited and simulation of quantum physical processes can only be done using powerful classical computers, for example simulation of quantum many-body systems on the Amazon cloud [414].

The November 20, 2020, issue of *Physics Today*, a publication of the American Institute of Physics, includes the article *Quantum computing ramps up in the private sector,* which reviews the state of the art of quantum computing: "...hurdles remain to achieving useful quantum computers. The number of qubits needs to be scaled up. The qubits are needed not only for computations but also for correcting errors due to decoherence of the fragile quantum state. Engineering infrastructure must be designed and built. Algorithm must be created." The article mentions the shift towards commercialization of Quantum Information Processing (QIP),[2] likely to create vast opportunities for young researchers in this field.

Quantum Computing. To compute in a digital world, *we transform the state of solid-state devices capable to perform Boolean operations as directed by a program stored in the memory of the system.* This process implies the ability to store information as the initial state of a physical system governed by the laws of classical physics and then to control the transformation of this information, i.e., the change of the system state, until we get the desired result in the shortest possible time and dissipating the least amount of energy.

If information can be stored as the state of a quantum system, e.g., the spin of electrons or the polarization of photons, then we can exploit the strange and often counterintuitive properties of quantum systems to process and communicate information faster and with little or no energy dissipation. There is a rich population of quantum particles, and several quantum particles and processes are being investigated for quantum computing; the list includes quantum dots[3] based on spin or charge, trapped ions in a cavity, superconducting qubits, and liquid NMR (Nuclear Magnetic Resonance). At this time, ion traps and superconducting qubits are widely considered the leading candidates for quantum computers. Quantum communication only deals with information carried by photon polarization or other photon modes.

The physical processes in each embodiment of information as the state of a quantum system are investigated by various fields of physics, and for this reason we only discuss a unifying framework, the mathematical model of a quantum system used by quantum mechanics. In quantum mechanics, all state transformations happen in a finite-dimensional Hilbert space, a vector space over complex numbers. A set of 2^n complex numbers $\{\alpha_0, \alpha_1, \ldots, \alpha_{2^n-1}\}$ with the property that the sum of their moduli is 1, i.e., $|\alpha_0|^2 + |\alpha_1|^2 + \ldots |\alpha_{2^{n-1}}|^2 = 1$, describes the state of a quantum system in an n-dimensional Hilbert space.

Bits, qubits, and the quantum circuit model. Quantum magic starts with the *qubit*, the quantum corespondent of a bit of classical information. A qubit embodies the state of the simplest quantum system. In the ket-bra notation introduced by Dirac, the state of a qubit is a vector in a two-dimensional Hilbert space $|\varphi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$.

This possibly intimidating expression simply reflects the fact that a qubit is in a *superposition state*; $|0\rangle$ and $|1\rangle$ are the two basis vectors, and α_0 and α_1 are complex numbers and $|\alpha_0|^2 + |\alpha_1|^2 = 1$. The projections of the qubit state, $|\varphi\rangle$, on the two basis vectors $|0\rangle$ and $|1\rangle$ are $|\alpha_0|^2$ and $|\alpha_1|^2$, respectively.

[2] QIP refers to quantum computing and quantum communication.

[3] Quantum dots are man-made nanoscale crystals that can transport electrons.

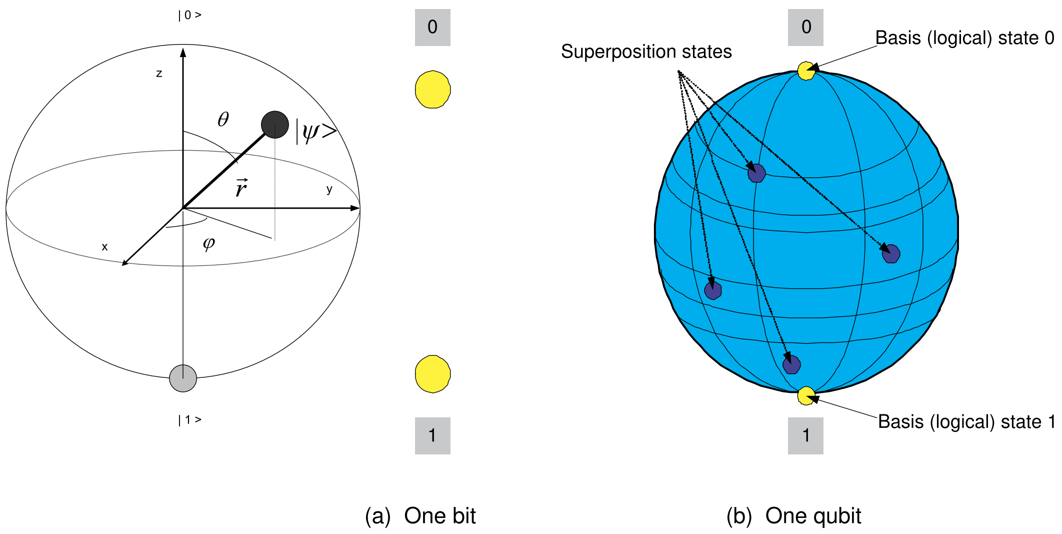


FIGURE 13.4

(Left) A qubit $|\,\psi\rangle$ is represented as a vector r from the origin to a point with angular coordinates $(\theta, \varphi, \gamma)$ on the Bloch sphere. (Right) A bit can only be in one of two states, 0 or 1, while a qubit can be anywhere on the surface of the Bloch sphere, a sphere of radius 1, i.e., can be in one of the two basis states $|\,0\rangle$ or $|\,1\rangle$ or in a superposition state $|\,\psi\rangle = \alpha_0\,|\,0\rangle + \alpha_1\,|\,1\rangle$ with $|\,\alpha_0\,|^2 + |\,\alpha_1\,|^2 = 1$.

To fully appreciate the impressive power of this model, consider the image of the qubit on a sphere of radius one called the *Bloch sphere*; see Fig. 13.4. A qubit can be anywhere on the surface of the Bloch sphere so a qubit can represent infinitely many bits (no pun intended) of information for different combinations of α_0 and α_1 [326].

When we measure a qubit, we project its state on the two basis vectors $|\,0\rangle$ and $|\,1\rangle$, and we obtain classical information: a bit with value 0 with probability $|\,\alpha_0\,|^2$ and one with value 1 with probability $|\,\alpha_1\,|^2$. In other words, when we observe or measure a qubit, we transform quantum information into classical information and force the qubit to either the south pole of the Bloch sphere corresponding to a classical bit with value 1 or to its north pole corresponding to a classical bit with value 0.

This is pure heresy: "God does not play dice" is the famous pronouncement of Albert Einstein reflecting his view about nondeterministic models of physical reality!! Fortunately, all experiments carried out during the past 100 years or so are consistent with the prediction of quantum mechanics, so is seems that Einstein and others questioning the nondeterminism of quantum mechanics are wrong.

Quantum parallelism and quantum algorithms. So far, we have looked at a single qubit, but imagine the immensity of a 2^n-dimensional Hilbert space and its potential to host quantum state transformations for a quantum computation. The implication of using qubits to process information brings us to the concept of *quantum parallelism*. Given a function $f(x), x \in \mathcal{I}$, assume there are $N = p^q$ elements in $\mathcal{I}$, the domain of the function. A classical computer can compute a particular value of the function one at a time, i.e., $f(a)$ for $x = a$. A quantum system computing $f(a)$ will result in a superposition state including all values in the codomain of function f, i.e., $f(x)\ \forall x \in \mathcal{I}$.

Regardless of the cardinality N of the function domain, all values of the function are computed at once, while a classical computer would need a time T to compute one value; a parallel computer would need N functional units to do so in T/N time. This sounds wonderful, but how could we find the value we are interested in, $f(a)$, from this superposition? A process called *amplitude amplifications* enables a particular waveform of the superposition to stand out.

Another important observation is that quantum parallelism is most useful for computations requiring the values of a function for its entire domain, as is the case of algorithms using the Fast Fourier Transform (FFT). The quantum factoring algorithm published by Peter Shor in 1994 cleverly exploits quantum parallelism using QFFT, the quantum version of FFT. When a quantum computer with a large number of qubits will be available, this algorithm will allow us to decrypt any messages encrypted since April 21, 753 AD,[4] and possibly earlier.

Another class of quantum algorithms was proposed by Lov Grover in 1996. Grover's algorithms find with high probability the unique input to a function that produces a particular output value, using just $\mathcal{O}(\sqrt(N))$ function evaluations, where N is the size of function's domain. Algorithms for quantum simulation of both classical and quantum systems are yet another important class of applications of quantum computers.

Quantum gates and quantum circuits. Quantum gates are the building blocks of a quantum computer based on the quantum circuit model. A gate carries out a unitary transformation U of its input qubits in state $\mid I\rangle$ into a set of qubits in state $\mid O\rangle$, such that $\mid O\rangle = U \mid I\rangle$. Single qubit gates, such as Pauli `X, Y, Z` gates and the `Hadamard` gate, rotate one qubit on the Bloch sphere. The target qubit of a `CNOT` gate is flipped when the control qubit is in state $\mid 1\rangle$ and left unchanged otherwise. A `Toffoli` gate is a three-qubit gate with two control qubits and one target qubit; the target qubit is flipped when both control qubits are in state $\mid 1\rangle$. Figs. 13.5(a) and (b) show one-qubit full adder circuit and reversible circuit with `CNOT` and `Toffoli` gates, respectively.

A set G of quantum gates is *universal* if, for any $\epsilon > 0$ and any unitary transformation U on n qubits, there is a sequence of gates $g_1, g_2, \ldots \in G$ such that the following norm satisfies the condition:

$$||U - U_{g_1} U_{g_2} \ldots U_{g_2} U_{g_1}|| \leq \epsilon. \tag{13.5}$$

Here, $U_g = V \otimes I$, where V is the unitary transformation on k qubits operated on by the quantum gate g, I is the identity operator acting on the remaining $n-k$ qubits, and $\otimes$ means the tensor product. Unitary transformations carried out by quantum gates are reversible, and Eq. (13.5) illustrates that a sequence of transformations carried out to compute the result should then be executed in reverse order to bring return the system to its original state. Examples of universal gate sets include: (a) `CNOT` and all single qubit gates; (b) `CNOT`, `Hadamard`, and `phase flip` gates; and (c) `Toffoli` and `Hadamard` gates.

Quantum computers. In mid-2020 there were several quantum processors with a small number of qubits, the most powerful being the 53 qubit IBM Q quantum computer operational since October 2019.

What makes building a quantum computer so challenging, why does it take so much effort to have a useful quantum computer when there are several embodiments of quantum information? In a seminal paper on the physical implementation of quantum computers, David DiVincenzo, at IBM at that time,

[4] According to a myth, on April 21, 753 B.C., Romulus and his twin brother, Remus, founded Rome.

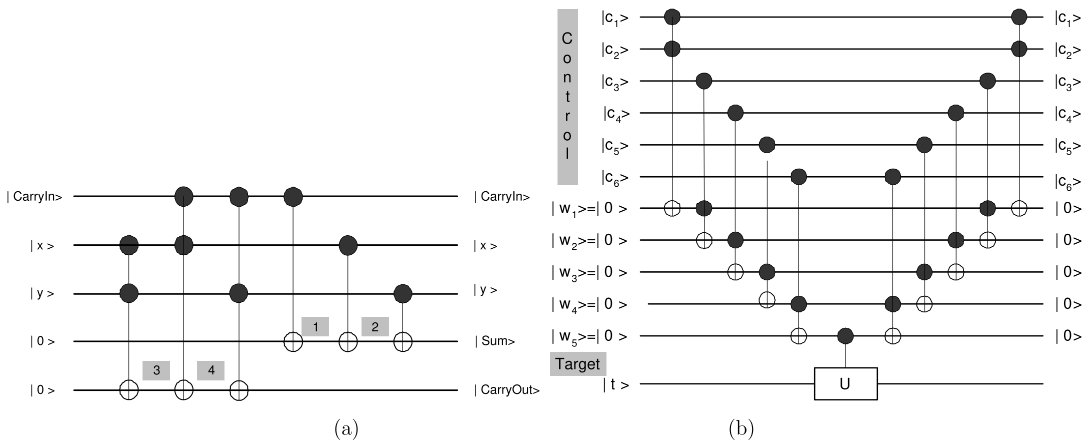


FIGURE 13.5

(a) One-qubit full adder circuit with 3 `Toffoli` gates and 3 `CNOT` gates. (b) Quantum circuit with 10 `Toffoli` gates, 6 control qubits, $| c_1\rangle$, $| c_2\rangle$, $| c_3\rangle$, $| c_4\rangle$, $| c_5\rangle$, and $| c_6\rangle$, one target qubit $| t\rangle$, and 5 *work qubits* $| w_1\rangle$, $| w_2\rangle$, $| w_3\rangle$, $| w_4\rangle$, and $| w_5\rangle$ initially in state $| 0\rangle$. The first stage produces the logical product, `AND`, of all six control qubits in $| w_5\rangle$.The second stage performs a single qubit U-controlled transformation of the target qubit. The last stage returns the work qubits to their original state, $| 0\rangle$.

formulated five criteria for building a useful quantum computer, plus two additional ones required for transfer of information. These criteria are:

1. A scalable physical system with well-characterized qubits.
2. The ability to initialize the qubits to a pure state $| 000......0\rangle$.
3. Long decoherence times, much longer than the gate operation times.
4. Existence of a “universal” set of quantum gates.
5. A qubit-specific measurement capability.
6. Ability to interconvert “stationary” (memory) and “flying” (communication) qubits.
7. Ability to faithfully transmit “flying” qubits between specified locations.

The last two requirements are related to communication. There are obvious similarities between these requirements and the ones for a classical computing engine. A recent paper [307] compares two architectures, superconducting transmon qubits[5] and trapped ions. The two fully programmable multi-qubit machines provide the user with the flexibility to implement arbitrary quantum circuits with high-level interface. This makes it possible, for the first time, to test quantum computers irrespective of their particular physical implementation.

A research group at Google claimed quantum supremacy according to an October 2019 issue of the journal *Nature*. *Quantum supremacy,* a long awaited milestone for quantum computing, is achieved

[5] A transmon is a superconducting charge qubit designed to have reduced sensitivity to charge noise. It was described in 2007 by Robert J. Schoelkopf, Michel Devoret, Steven M. Girvin, and their colleagues at Yale University.

when a quantum system solves a computational problem that is beyond the practical capabilities of "classical" computers. Google's claim that the problem solved using the Sycamore chip would take 10 000 years to complete on the fastest classical supercomputer was contested by IBM researchers who argued that the problem solved by Sycamore could in fact be solved in 2.5 days on a classical supercomputer.

Decoherence and quantum error correction. Quantum systems are affected by *decoherence*, the loss of information from the quantum system into the environment the system is loosely coupled with. The decoherence time, the time elapsed since the state is set until the decoherence alters it, is very short. For example, the decoherence time for quantum dots-based charge and quantum dots-based spin are 10^{-9} and 10^{-6} seconds, respectively; the decoherence time for trapped indium atoms in quantum dots is 10^{-1} seconds [329]. This implies that error correction should be done periodically at intervals shorter than the decoherence time. "The speed at which decoherence occurs can make or break a qubit" summarizes the thinking of QIT researchers. Quantum error correction faces multiple challenges:

1. Only orthogonal quantum states can be distinguished with absolute certainty; if a quantum state is transformed by two distinct quantum errors into nonorthogonal states, then neither error can be corrected because the two states are not distinguishable.
2. A quantum measurement projects the state, thus, in the general case, it alters the state of a quantum system; we have to devise ingenious means to carry out the measurements necessary to identify a qubit in error.
3. A qubit may be affected by bit-flip, phase-flip, or by both bit- and phase-flip errors.
4. Entanglement adds a new dimension to quantum error correction; an error may propagate to some or all qubits entangled with one another, and we can expect non-Markovian quantum errors, errors correlated in time and/or space.

A major discovery is due to Peter Shor, who in 1995 showed that it is possible to restore the state of a quantum system using only partial information about the state. Until then, it was not known how to perform an error correction step without a measurement that would alter the state. Shor's idea was *to fight entanglement with entanglement.*

The intuition that a classical binary $[n, k]$ linear code, with n the number of bits of every codeword and $k < n$ the number of information bits, is able to correct t errors iff (if and only if) Hamming spheres of radius t about each codeword are disjoint and exhaust the entire space of n-tuples extends to quantum codes. A quantum error correcting code encodes one logical qubit L to n physical qubits in a 2^n Hilbert space with basis vectors $\{| 0\rangle, | 1\rangle, \ldots | i\rangle, \ldots | 2^n - 1\rangle\}$ with

$$| 0\rangle_L = \sum_{i=0}^{2^n-1} \alpha_i \, | i\rangle \quad \text{and} \quad | 1\rangle_L = \sum_{i=0}^{2^n-1} \beta_i \, | i\rangle$$

The 2^n-dimensional Hilbert space should accommodate orthogonal subspaces for possible bit-flip, phase-flip, and bit- and phase-flip errors affecting each one of the n physical qubits.

The quantum error correcting code must map coherently the two-dimensional Hilbert space spanned by $| 0\rangle_L$ and $| 1\rangle_L$ to multi-dimensional Hilbert spaces to ensure that the code is capable of correcting the three types of errors for each of the n qubits of the basis vectors $| 0\rangle_L$ and $| 1\rangle_L$, and, in addition, one subspace for entangled state $| 0\rangle_L$ and $| 1\rangle_L$.

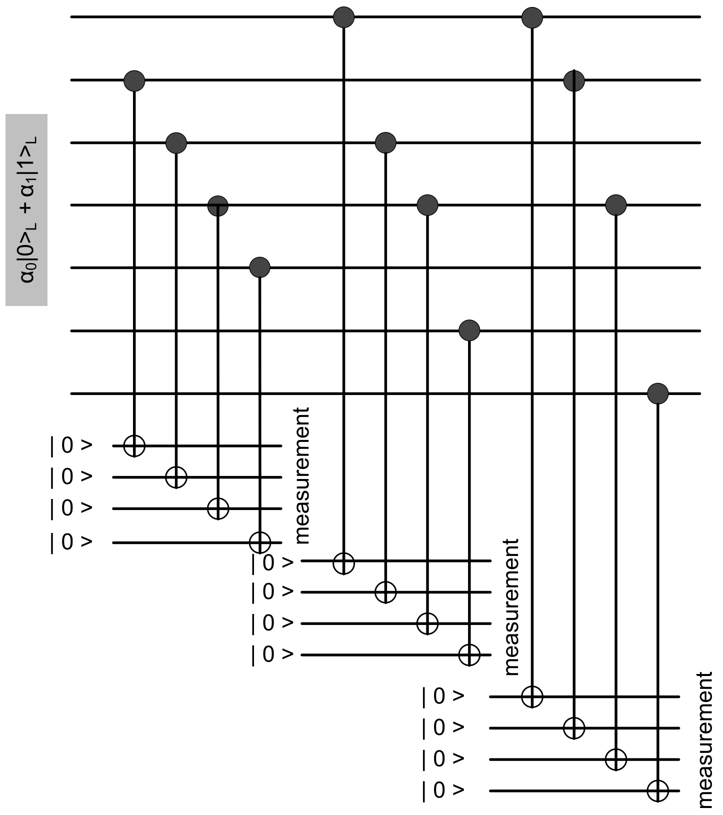


FIGURE 13.6

A circuit for calculating the syndrome for bit-flip errors for the 7-qubit Steane code. The input is the logical qubit $| \varphi\rangle = \alpha_0 \mid 0\rangle_L + \alpha_1 \mid 1\rangle_L$. The seven physical qubits act as the control qubits of the 12 `CNOT` gates that set the ancilla qubits; a nondemolition measurement of the syndrome reveals if one of the seven physical qubits is in error.

The Quantum Hamming Bound, $2(3n + 1) \leq 2^n$, reflects these requirements; it shows that the smallest number of physical qubits used to encode the superposition states $\mid 0\rangle_L$ and $\mid 1\rangle_L$ and then recover them regardless of the qubit in error and the type of error is $n = 5$.

The *syndrome* Hv^T of a linear code with the parity check matrix H uniquely identifies the bit(s) in error of the n-tuple v (v^T is the transpose of vector v). We apply the same basic strategy to identify a qubit in error, and we carry out a *nondemolition measurement* to determine the syndrome. Computation of the syndrome for a quantum code requires a number of ancilla (auxiliary) qubits; each ancilla qubit is entangled with multiple qubits of the encoded quantum word—see, for example, Fig. 13.6. The degree of entanglement of an ancilla qubit with each one of the n physical qubits is relatively weak; the nondemolition measurement of ancilla qubits allows us to determine the syndrome without altering the state of the n physical qubits.

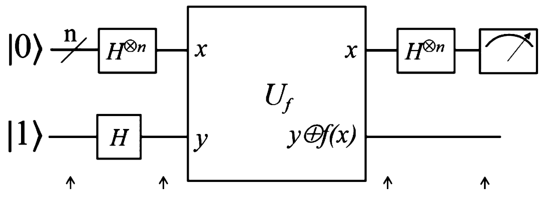


FIGURE 13.7

Quantum circuit for the Deutsch–Jozsa problem.

Implementation of quantum error-correcting codes significantly complicates the design of quantum processors. Quantum error correction increases by one or more orders of magnitude the number of qubits and the number of elementary steps required by a quantum computation. The overhead of the error correction is measured by: (i) *scale-up*—the ratio of the number of physical to logical qubits; and (ii) *slow-down*—the ratio of the number of gates to the number of the elementary steps of computation.

Quantum circuit simulation with Qiskit. Qiskit is an open-source simulation software for experimentation with quantum computers at the level of circuits, pulses, and algorithms; see https://qiskit.org/documentation/intro_tutorial1.html. Qiskit use involves several steps: (i) design the quantum circuit(s); (ii) compile the circuit(s); (iii) run the compiled circuits; and (iv) analyze, i.e., compute summary statistics and visualize the results of the experiments. We illustrate now the use of this wonderful tool for a simple but insightful problem named after quantum computing researchers who proposed the problem and its solution. The Deutsch–Jozsa problem was conceived to illustrate the power and elegance of quantum algorithms, rather than solve a practical problem.

Deutsch–Jozsa problem. Assume that Alice plays the following game with Bob: (i) Bob chooses an either constant or balanced function $f(x)$. Function $f(x)$ is constant if it has the same value for all arguments x, i.e., $f(x) = 0\ \forall x$ or $f(x) = 1\ \forall x$; $f(x)$ is balanced if $f(x) = 1$ for half of values x and $f(x) = 0$ for the other half; (ii) Alice sends an n bit integer N $(0 \le N \le 2^n - 1)$ to Bob; and (iii) Bob calculates $f(N)$ and sends Alice the one bit of information 0 if f is constant and 1 if f is balanced.

The question is how fast can Alice succeed, i.e., how many times does she need to send an n-bit integer to Bob before she is certain whether Bob had chosen a constant or a balanced function $f(x)$? Classically, in the worst case, Alice has to send 2^{n-1} values and receive a 0 before receiving a 1 or vice versa. So, she needs to query Bob $2^{n-1} + 1$ times. If Alice is very lucky, she can guess after two trials; if she gets different values, she will know that $f(x)$ is balanced. Fortunately, she can do much better using the quantum circuit discussed next.

Fig. 13.7 shows the quantum circuit for the Deutsch–Jozsa problem. The input consists of an n-qubit query register to store the value N and a one qubit answer register for Bob to store $f(N)$. An n-dimensional Hadamard Transform $H^{\otimes n}$ is applied to the first register, and a Hadamard Transform is applied to the one qubit answer register. The operator $H^{\otimes n} \otimes H$ transforms the input state at the leftmost uparrow $\uparrow$, and the resulting state at the next $\uparrow$ is the input to a unitary transformation U_f carried out by an *oracle*. Call x and y the two inputs of U_f and x and $y \oplus U_f$ with $\oplus$ the binary addition, at the output, at the third $\uparrow$ in the figure. Then the top register is transformed again by $H^{\otimes n}$ and then measured.

The quantum solution exploits quantum parallelism. Alice measures the query register in state

$$| \varphi\rangle = \sum_z \sum_x \frac{(-1)^{x \cdot z + f(x)}}{\sqrt{2^n}}$$

with $x \cdot z$ being the bitwise inner product of x and z modulo 2. The amplitude of state $| 0^{\otimes n}\rangle$ is $A = \frac{\sum_x (-1)^{f(x)}}{2^n}$. If f is constant, A is either $+1$ or -1, depending on $f(x)$ and the measurement of the query register will be 0 because $| \varphi\rangle$ is of unit length and all other amplitudes must be 0. If f is balanced, the positive and negative contributions to A cancel, thus $A = 0$; a measurement must show a result different from 0 for at least one qubit of the query register. In conclusion, Alice gets the result 0 if and only if function f is balanced after only one evaluation of function f, regardless of how large n is.

Qiskit simulation is presented in Fig. 13.8. Fig. 13.8a displays the initialization code for a simulation with a 4-qubit query register. Fig. 13.8b displays the code for Hadamard Transformations, oracle invocation, and measurement; the code includes drawing the circuit for each stage, but omitted in this presentation. The oracle implements the unitary transformation U_f. Fig. 13.8c shows the code for running the simulation, the full circuit, and the state at each stage of the circuit.

Cloud access to quantum simulation software; experimenting with quantum systems. To spark interest in quantum computing among computer programmers, venture capitalists, and students wishing to explore this fascinating field, several companies support cloud access to platforms for experimenting with quantum computing. Multiple sources for information about quantum software are also available; Quantum Open Source Foundation maintains a GitHub open-source quantum software projects repository; see https://github.com/qosf/awesome-quantum-software. An overview of open source quantum computing software can be found in [174].

Several organizations offer access to a set of quantum programming tools and to quantum systems with a small number of qubits, while others are only involved in quantum programming. Quantum programming is the process of assembling sequences of instructions, called quantum programs, capable of running on a quantum computer. Quantum programming languages help express quantum algorithms using high-level constructs.

Arguably, the most sophisticated quantum environment is *IBM Q Experience,* an online platform launched in 2016. Q Experience allows general public access to a set of IBM's prototype quantum computers including two 5-qubit processors and one 16-qubit processor. A quantum system with 20 qubits is also available through the IBM Q Network; see https://www.ibm.com/quantum-computing/.

IBM's Q simulators and quantum devices can be accessed over the cloud using an open-source quantum programming framework launched in 2017. Qiskit integrates quantum development into regular workflows using common programming languages and standard development tools.

Quantum Computing Playground is a WebGL Chrome Experiment offered by Google. The experiment uses a GPU-accelerated quantum computer with an IDE interface and a scripting language with debugging and 3D quantum state visualization support. The system has several quantum gates built into the scripting language, can simulate quantum registers of up to 22 qubits, and runs Shor factorization and Grover database search algorithms.

Microsoft released recently the *Quantum Development Kit,* replacing an earlier version called LIQUi$|\rangle$. This system includes: (i) a quantum programming language, Q#, integrated with Visual Studio development environment; (ii) simulators that run on a local system or on the Azure cloud platform;

```
# importing Qiskit
from qiskit import *
%matplotlib inline
```

```
# Set number of bits of input
n = 4
# Create circuit object
qc = QuantumCircuit(n+1,n)
```

```
# Create the oracle
b = '0101'
oracle = QuantumCircuit(n+1)

# Invert where we set the b string to 1
for i in range(n):
    if b[i] == '1':
        oracle.x(i)

# CNOT with the target bit for each bit in the input
for i in range(n):
    oracle.cx(i,n)

# Revert any inverted bits from the b string
for i in range(n):
    if b[i] == '1':
        oracle.x(i)
```

```
# Set the working bit to 1 (initial state)
qc.x(n)
qc.barrier()
qc.draw(output='mpl')
```

(a) Qiskit initialization; 4-qubit query register.

```
# Apply Hadamard gates
for i in range(n+1):
    qc.h(i)
qc.barrier()
qc.draw(output='mpl')
```

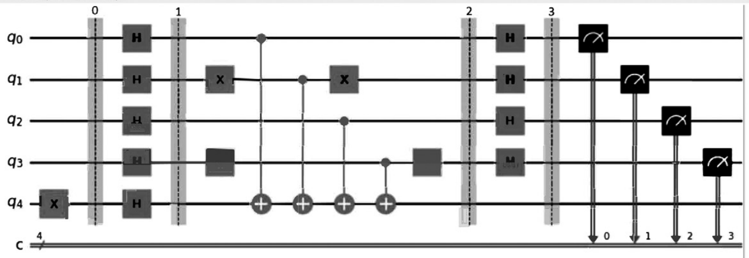
```
# Apply the oracle
qc += oracle
qc.barrier()
qc.draw(output='mpl')
```

```
# Repeat hadarmard gate
for i in range(n):
    qc.h(i)
qc.barrier()
qc.draw(output='mpl')
```

```
# Measure
for i in range(n):
    qc.measure(i,i)
qc.draw(output='mpl')
```

(b) Qiskit for Hadamard Transformations, oracle invocation, and measurement; circuit drawing after each operation omitted.

```
# Run a simulation
results = Aer.get_backend('qasm_simulator').run(assemble(qc, shots=10000)).result()
counts = results.get_counts()
print(counts)
```



Stage	Step	State
0	Input	00001
1	Hadamard	$\frac{1}{\sqrt{2^{n+1}}}\sum_{x=0}^{2^n-1}\lvert x\rangle(\lvert 0\rangle - \lvert 1\rangle)$
2	Oracle	$\frac{1}{\sqrt{2^{n+1}}}\sum_{x=0}^{2^n-1}(-1)^{f(x)}\lvert x\rangle(\lvert 0\rangle - \lvert 1\rangle)$
3	Hadamard	$\frac{1}{2^n}\sum_{y=0}^{2^n-1}\left[\sum_{x=0}^{2^n-1}(-1)^{f(x)}(-1)^{x\cdot y}\right]\lvert y\rangle$

(c) Run simulation, the circuit, and state.

FIGURE 13.8

Qiskit simulation: (a) and (b) the simulation code; (c) the circuit and the state.

and (iii) libraries and code samples. The software can be downloaded from https://www.microsoft.com/en-us/quantum/development-kit.

Quantum Inspire (https://www.quantum-inspire.com/) is the first European platform providing on-line access to a fully programmable 2-qubit electron spin quantum processor, *Spin-2*, and to a quantum processor with five superconducting qubits, *stramon-5*. *Spin-2* is a quantum processor hosting two single electron spin qubits in a double quantum dot in isotopically purified Si (Silicon). *Starmon-5* consists of five superconducting transmon qubits in an X configuration.

The QX emulator from Quantum Inspire simulates up to 26 qubits on a commodity cloud-based server and up to 31 qubits using "fat" nodes on the Cartesius supercomputer. Circuit-based quantum algorithms can be created using either a GUI or the Python-based Quantum Inspire SDK. A knowledge base is also available at https://www.quantum-inspire.com/kbase/introduction-to-quantum-computing/.

In 2018, QC Ware, a quantum-computing-as-a-service company from Palo Alto, launched *Forge,* a cloud platform allowing developers to run algorithms on hardware provided by multiple vendors. QC Ware is focused on software development rather than building its own quantum computers. For now, Forge offers access to a D-Wave quantum computer for simulations of Google and IBM quantum devices. D-Wave One, a quantum computer operating on a 128-qubit chipset became operational in 2011, D-Wave 2X, a 1000+ qubit quantum computer and D-Wave 2000Q and an open source repository containing software tools for quantum annealers became available in 2015 and 2017, respectively. D-Wave quantum computers use quantum annealing for solving optimization problems. Quantum annealing is a method for finding the global minimum of a function using quantum fluctuations.

Rigetti Computing is a Berkeley-based developer of quantum integrated circuits used for quantum computers. The company also develops a cloud platform called *Forest* that enables programmers to write quantum algorithms. The platform is based on a custom instruction language the company developed called Quantum Instruction Language (Quil) using a set of Python tools. The system allows users to write quantum algorithms for a simulation of a quantum chip with 36 qubits.

IonQ, a company founded in 2015, aims to take trapped-ion quantum computing out of the lab and into the market. In August 2020 IonQ opened an 11-qubit quantum computer to the public via the Amazon Braket cloud platform, and in October unveiled a 32-qubit version of it. Palo Alto-based PsiQuantum, also founded in 2015, patterns silicon wafers into thousands of photonic components containing waveguides for carrying the single photons encoding the qubits. PsiQuantum founders predict that they will have a useful quantum computer with a million qubits in a few years.

13.4 Vehicular clouds

By way of motivation. Present-day vehicles feature powerful on-board computers, ample storage, sophisticated transceivers, and an impressive panoply of sensing devices. In the near future, we expect to see millions of these advanced vehicles crisscrossing our roadways and city streets. We hold the view that it is only a matter of time before this fleet of sophisticated vehicles will be recognized as an abundant and underutilized compute resource that can be tapped into for the purpose of providing third-party or community services.

One of the characteristics that distinguishes vehicles from servers in a conventional cloud is mobility. Indeed, vehicles participating in traffic are involved, on a daily basis, in various dynamically changing situations, ranging from normal traffic to congestion, to accidents, to other similar traffic-related events.

Under the present-day state of the practice, our vehicles are mere spectators that witness traffic-related events without being able to participate in the mitigation of their effects. We suggest that in such situations the vehicles have the potential to cooperate with various authorities to help solve problems that otherwise either take an inordinate amount of time to solve (e.g., dissipate traffic jams) or be solved in a timely manner for lack of adequate computing resources and accurate traffic information. In our vision, vehicles stuck in a traffic jam will be able to let municipal Traffic Management Centers use their on-board compute resources and sensors to run complex simulations designed to help alleviate the effects of congestion by city-wide rescheduling of traffic lights.

We also anticipate that, in planned evacuation situations, the combined compute power of the vehicles will enable evacuation management centers to compute and disseminate information about the safest evacuation routes.

Vehicular Clouds—How it all started. A decade ago, inspired by the promise of conventional clouds and by the realization that modern vehicles can act as servers in a vehicular data center, Abuelela and Olariu [7] and Eltoweissy et al. [163], introduced Vehicular Clouds (VCs) as: *"A group of vehicles whose corporate computing, sensing, communication and physical resources can be coordinated and dynamically allocated to authorized users"*.

Although the authors of [7] and [163] have contemplated from the start VCs set up on moving vehicles, it is interesting to note that the first papers discussing VCs considered the simpler case of parked vehicles. Thus, it has become customary in the VC literature to distinguish between *static* VCs involving stationary (e.g., parked) vehicles and *dynamic* VCs that are harnessing the compute resources of moving vehicles. We believe that this distinction is largely artificial and that, with the advent of ubiquitous network coverage available in the Smart Cities of the (near) future, connectivity will be ubiquitous and pervasive, and the on-board compute resources of vehicles will be available 24/7, whether these vehicles are parked or not.

Vehicular clouds are a nontrivial variant of conventional clouds. One of the key distinguishing characteristics of VCs is the dynamically changing amount of compute resources available. This characteristic is a direct consequence of mobility and of the distributed ownership of vehicles. As vehicles enter the VC, fresh compute resources become available; when vehicles leave, often unexpectedly, their resources depart with them, creating a highly dynamic environment. In turn, the dynamically changing availability of compute resources, due to vehicles joining and leaving the VC, unexpectedly leads to a volatile computing environment where reasoning about system performance becomes challenging [177,178].

The mobility attribute of VCs, combined with the fact that the presence of vehicles in close proximity to an event is very often unplanned, implies that the pooling of the resources of those vehicles into a VC in support of mitigating the event must occur spontaneously by the common recognition of a need for which there are no preassigned or dedicated resources available. This *agility of action* seldom exists in conventional clouds and turns out to be an important defining characteristic of VCs.

The early papers [7] and [163] hinted at an entire array of possible applications of VCs. In addition to numerous interesting applications of VCs, these early papers identified an entire series of research challenges. Given their large array of potential applications, it is reasonable to expect that, once adopted, the VCs will be the next paradigm shift, with a lasting technological and societal impact. In fact, the VCs may well turn out to be the "killer app" taking vehicular networks to the next level of relevance and innovation [371]. Recent years have seen the emergence of VCs as an active topic of research. Recent surveys [69], [372], and [373] summarized trends and advances in VC research and pointed out new application domains and implementation challenges ahead.

VC resource management. Resource management is a fundamental task in conventional clouds and remains of great importance in VCs. Recall that one of the defining ways in which VCs differ from conventional clouds is resource volatility. Therefore, managing VC resources is an important, albeit very challenging, task. The main goal of this section is to discuss a few important instances of resource management in VCs.

Estimating the expected number of vehicles in the VC. One of the key tasks to be performed by the VC *resource manager* is to predict, with some degree of accuracy, the number of vehicles present in the VC, given generic information about the vehicle arrival and departure rates. Specifically, assume that at time $t = 0$, the VC contains $n_0 \geq 0$ vehicles. After that, vehicles arrive at a time-dependent rate $\lambda(t)$ and depart at a time-dependent rate $\mu(t)$. Of interest are:

- the probability that, at a given time $t > 0$, there are exactly k vehicles in the VC;
- the variance of the number vehicles in the VC at time $t > 0$;
- the limit as $t \to \infty$ of the expected number of vehicles in the VC (provided such a limit exists).

Consider the counting process $\{N(t) \mid t \geq 0\}$ of continuous parameter t, where for some arbitrary integer k the event $\{N(t) = k\}$ occurs if the VC contains k vehicles at time t. We let $P_k(t) = \Pr[\{N(t) = k\}]$ denote the probability that the event $\{N(t) = k\}$ occurs and assume that the VC has infinite capacity. We model our problem as a nonhomogeneous birth-and-death process.

Letting $G(z,t) = \sum_k P_k(t) z^k$ denote the probability generating function of $P_k(t)$, it is fairly routine to see that $G(z,t)$ satisfies the partial differential equation

$$\frac{\partial G(z,t)}{\partial t} + \mu(t)(z-1)\frac{\partial G(z,t)}{\partial z} = \lambda(t)(z-1)G(z,t), \tag{13.6}$$

with the boundary condition $G(z,0) = z^{n_0}$. It is also straightforward to confirm that the solution to (13.6) is

$$\begin{aligned} G(z,t) = {} & \left[e^{-\int_0^t \mu(u)du} z + (1 - e^{-\int_0^t \mu(u)du})\right]^{n_0} \\ & \times \exp\left[-(1-z)e^{-\int_0^t \mu(u)du} \int_0^t \lambda(u) e^{\int_0^u \mu(s)ds} du\right]. \end{aligned} \tag{13.7}$$

In spite of its complexity, Eq. (13.7) reveals much of the structure of the process $\{N(t) \mid t \geq 0\}$. Indeed, observe that $G(z,t)$ is the product of two factors:

- $\left[e^{-\int_0^t \mu(u)du} z + (1 - e^{-\int_0^t \mu(u)du})\right]^{n_0}$ that is the probability generating function of a binomial random variable with parameter n_0 and "success" probability $p(t) = e^{-\int_0^t \mu(u)du}$;

- $\exp\left[-(1-z)e^{-\int_0^t \mu(u)du}\int_0^t \lambda(u)e^{\int_0^u \mu(s)ds}du\right]$ that, upon examination, turns out to be the probability generating function of a (nonhomogeneous) Poisson random variable with parameter

$$\Lambda(t) = \frac{\int_0^t \lambda(u)e^{\int_0^u \mu(s)ds}du}{e^{\int_0^t \mu(u)du}}. \tag{13.8}$$

We define two additional counting processes:

- $\{R(t) \mid t \geq 0\}$ that keeps track of the number of the n_0 vehicles in the VC at time $t = 0$ that are still in the VC at time t; it is clear that the "success" probability $p(t) = e^{-\int_0^t \mu(u)du}$ is precisely the probability that such a generic vehicle is still in the VC at time t;
- $\{S(t) \mid t \geq 0\}$ that keeps track of the number of vehicles in the VC at time t that were not in the VC at time $t = 0$; as already mentioned, this is a nonhomogeneous Poisson process with parameter $\Lambda(t)$ defined in (13.8).

It is immediately clear that, for all $t \geq 0$, $R(t)$ and $S(t)$ are independent random variables. Furthermore, the expression of $G(z,t)$ as a product implies that for all $t \geq 0$, $N(t)$ is the convolution of $R(t)$ and $R(t)$, and so

$$N(t) = R(t) + S(t). \tag{13.9}$$

A direct consequence of (13.9) and of the linearity of expectation is that

$$\begin{aligned} E[N(t)] &= E[R(t)] + E[S(t)] \\ &= n_0 e^{-\int_0^t \mu(u)du} + e^{-\int_0^t \mu(u)du}\int_0^t \lambda(u)e^{\int_0^u \mu(s)ds}du \\ &= e^{-\int_0^t \mu(u)du}\left[n_0 + \int_0^t \lambda(u)e^{\int_0^u \mu(s)ds}du\right]. \end{aligned} \tag{13.10}$$

Even though the standard way of obtaining $E[N(t)]$ would be to compute the partial derivative of $G(z,t)$ with respect to z and then take $z = 1$, the merit of Eq. (13.10) is that it provides a more direct way of computing the expected number of vehicles in the VC. The same observation holds, *mutatis mutandis*, for computing the variance $Var[N(t)]$ of $N(t)$.

Virtual machine migration in VC. We assume that each vehicle has a *virtualizable* on-board computer preloaded with a Virtual Machine Monitor in charge of mapping between the guest VM and the host vehicle's resources. VM migration is one instance of *resource management* in cloud computing. It remains a fundamental performance attribute in VCs, as well.

One of the fundamental resources in the VC are the various VMs hosted by vehicles. When a vehicle leaves the VC while its hosted VM is still active, a failure occurs and various strategies need to be implemented to minimize disruption and performance degradation. It is intuitively clear that, the longer and more predictable the vehicle residency times are in the VC, the easier it is to predict the optimal moment when VM migration has to be undertaken. By contrast, when vehicular residency times in the VC are short and/or unpredictable, VM migration becomes very challenging.

Baron et al. [49] used mobility traces of city buses in Dublin, Ireland, to investigate the feasibility of VM migration between buses at communication hotspots. Refaat et al. [409] have proposed a Vehicular

Virtual Machine Migration (VVMM) framework. They have studied strategies for selecting the target vehicle for VM migration. Unlike the VM migration strategy of [49], VVMM requires preinstalled roadside infrastructure. Finally, Florin et al. [178] have investigated VM migration in the context of VCs running compute-intensive applications in support of Big Data.

Estimating vehicular residency times. To identify the types of workloads supported by a given VC architecture, it is exceedingly important to model vehicular residency times as a function of its specific mobility parameters. Although few authors seem to pay much attention, it is of fundamental importance to distinguish between the characteristics of vehicle residency times in the VC and the VM lifetime. VMs are associated with job execution, while vehicle residency times reflect the amount of time vehicles spend in the VC.

Recently, three papers investigated *quantitatively* vehicular residency times. The first, [31], was concerned with estimating the number of vehicles present in a VC given stochastically changing arrival and departure parameters. The second, [540], assumed a VC built on top of moving vehicles and have studied the amount of residency times of vehicles in the VC assuming various characteristics of (urban) traffic flow. Finally, [524] also looked at the residency times of vehicles in the downtown area of a smart city. Their result can be extended with minor changes to assessing the vehicular residency times in a VC set up in the downtown area of a smart city, assuming that a sufficient level of supporting roadside infrastructure is available.

Identifying feasible VC workloads and applications. The VC literature is full of papers discussing VC architectures implemented on top of moving vehicles and supported by ubiquitous and pervasive roadside infrastructure. However, some of those authors do not seem to be concerned with the obvious fact that moving vehicles' residency times in the VC may, indeed, be very short and therefore so is their contribution to the amount of useful work performed. While such supporting infrastructure may well become available in the future, it does not exist today. Similarly, zero-infrastructure VCs built on top of fast-moving vehicle are utterly unrealistic under present-day technology since the DSRC-mandated V2V transmission range, combined with the limited bandwidth inherent to wireless communications, cannot guarantee sufficient time to permit VM migration and/or data offloading in case vehicles exit the VC.

Most conventional clouds are supported by tens of thousands of servers. By comparison, VCs involve a much smaller number of vehicles. As a result, while conventional clouds can handle arbitrary workloads and provide high reliability of the order of "four nines" (i.e., 99.99%) availability, the workloads supported by VCs depend, to a large extent, on the number and residency characteristics of participating vehicles.

Thus, various VC architectures, distinguished by the number of participating vehicles, existing communication bandwidth, and vehicle residency times, are, in all likelihood, apt to support multiple types of user applications and workloads.

Given the significant differences between various instances of VCs, one of the important tasks is to identify *feasible* applications for each of them. For example, VCs with a very short vehicle residency times are clearly not suited for long-lived applications involving a lot of data. This is so, among other reasons, because VM migration and data replication will be next to impossible to support efficiently.

The workloads specific to these application categories differ substantially and require different architectural support. Can VCs support these applications? Very little work along these lines has been reported in the literature. It is clear that a VC, built on top of a large number of participating vehicle known to spend a predictable amount of time in the VC, is suitable for demanding applications.

By contrast, VCs set up in the parking lot of a convenience store where vehicles spend a short amount of time will not be able to support effectively the same application. They, however, can support different, more lightweight applications.

We suggest using VCs in support of the following types of workloads: (i) transaction-oriented processing for decision support systems and business analytics; and (ii) mobile interactive applications that process large volumes of data from different types of sensors. As an illustration, imagine a VC set up in the parking lot of a shopping mall. During the normal business hours, the various businesses at the mall can leverage the compute power of the vehicles in the shopping-mall parking lot for business analytics, mostly involving transaction-oriented processing. Likewise, the mall security personnel can harness the compute power of the cars in the parking lot to process various sensor inputs, including face recognition and other similar software programs intended to provide security for the shoppers at the mall.

Assigning job to vehicles in VC. As mentioned before, the dynamically changing availability of compute resources due to vehicles joining and leaving the VC leads to a volatile computing environment where the task of assigning incoming user jobs to vehicles is quite challenging.

To get a feel for the problem, assume that a user job was assigned to a vehicle in the VC. If the vehicle remains in the VC until the job completes, all is well. Difficulties arise when the vehicle leaves the VC before job completion. In this case, unless special measures are taken, the work is lost, and the job must be restarted, taking chances on another vehicle, and so on, until eventually the job is completed. Clearly, losing the entire work of a vehicle, in case of a premature departure, must be mitigated. One possible method is to use checkpointing, a strategy originally proposed in databases. This strategy requires making regular backups of the job's state, and storing them in a separate location. In the event a vehicle leaves before job completion, the job can be restarted, on a new vehicle, using the last backup. While this seems simple and intuitive, checkpointing is not implemented efficiently in VCs.

Reliability and availability can be enhanced, as it is often done in other computer systems, by employing various forms of redundant job-assignment strategies. Recently, Ghazizadeh et al. [196] have studied redundancy-based job assignment strategies and have derived analytical expressions for the corresponding MTTF. Building on the results in [196], Florin et al. [177] have developed an analytical model for estimating job completion time in VCs assuming a redundant job-assignment strategy.

Research Challenges. Several important VC research challenges are discussed next.

Architectural Challenges. The architectural challenges faced by the VC community include issues related to the organization of the cyber structure of the VC and its interaction with the physical resources. There is a critical need to efficiently manage host mobility and heterogeneity (including compute, communication, and storage capabilities) and vehicle membership.

One of the obvious applications of VCs is to help the authorities mitigate the effects of traffic-related events whose timely resolution cannot be met by preassigned assets or in a proactive fashion. It is, therefore, clear that the VCs can only achieve their full potential if their basic architecture is engineered to offer seamless integration of the resources of the participating vehicles. In particular, the architecture must adapt its managed vehicular resources allocated to an application according to dynamically changing requirements and system conditions.

Operational and Policy Challenges. For the VC to operate seamlessly, issues related to authority establishment and management, decision support and control structure, the establishment of incentives, accountability metrics, assessment and intervention strategies, rules and regulations, standardization, etc. must be addressed. By the same token, there is a need for economic models and metrics to determine reasonable pricing and billing for VC services. We anticipate that some forms of VCs will be contract-driven, where the owner of the vehicle or vehicular fleet consents to renting out some form of excess computational or storage capacity. At the same time, mobility concerns dictate that, in addition to the contract-based form of VC, it should be possible to form a VC in an ad hoc manner as necessitated by dynamically changing situations or demand.

VC server consolidation. In conventional clouds, server consolidation is one of the fundamental performance booster and cost-reduction strategies. The idea behind server consolidation is to migrate VMs from lightly loaded servers so that they can be powered off (to save energy) or to migrate VMs from overloaded servers to improve processing or response times. Many strategies for VM consolidation to optimize a given objective functions have been proposed in the cloud computing literature.

It is known that server consolidation can be reduced to *bin-packing*, a known NP-hard problem. Thus, general exact solutions for server consolidation are very unlikely to exist. This has motivated the search for heuristics of which some are quite efficient. Server consolidation in cloud computing is still an active area of investigation. In spite of its fundamental importance, server consolidation has not been addressed in the context of VCs.

Promoting VCs reliability and availability. Dependability is an integrative concept that encompasses a number of attributes. Of these, *reliability* and *availability* are fundamental to conventional clouds and also to VCs. Availability is a measure of the delivery of correct service with respect to the alternation of correct and incorrect service. Reliability is a measure of continuous delivery of correct service, equivalently of the time to the next failure. Depending on the application domain, different emphasis is placed on these attributes.

The high availability requirements of conventional clouds can only be met through redundancy of storage (in addition to execution redundancy, where each user job is be executed by two or more servers). Storage redundancy implies that virtually each data item must be stored at several locations in the network of data centers, often in data centers at various locations around the world.

The dynamically changing availability of compute resources due to vehicles joining and leaving the VC unexpectedly leads to a volatile computing environment where promoting system reliability and availability is very challenging. Yet, if VCs are to see an adoption rate and success similar to that of conventional clouds, reliability and availability issues must be addressed adequately.

Recently, Florin et al. [180] studied the reliability of VCs with *short vehicular residency times*, as is the case in shopping mall parking lots or the short-term parking lot of a major airport. They showed how to enhance system reliability and availability in these types of VCs through a family of redundancy-based job assignment strategies that attempt to mitigate the effect of compute-resource volatility. They offered a theoretical prediction of the corporate MTTF and availability offered by these strategies in terms of the MTTF of individual vehicles and of the MTTR. Extensive simulations using data from shopping mall statistics have confirmed the accuracy of their analytical predictions. To the best of our knowledge, [180] is the *only* paper in the literature that addresses evaluating the MTTF and availability in VCs built on top of vehicles with a short residency time.

Incentivizing participation in VCs. The early VC papers [7,163] have suggested that, to set up credible VCs, the owners of the vehicles involved will have to be suitably incentivized. For example, we anticipate that in the near future air travelers will park and plug their cars in airport long-term parking lots.

In return for free parking and other perks, they will allow their cars to participate, during their absence, in the airport data center, as discussed in [31].

We now discuss a number of possible approaches to incentivize participation in VCs. The first possible approach will be a lightweight bidding/auction-based solution. Specifically, the data center will announce the list of resources needed by the application along with (1) their qualitative and quantitative attributes, and (2) a specified remuneration level for each resource. In response to the solicitation, vehicles will bid. To make the workforce recruiting operation fast, the number of bidding rounds will be kept to a minimum.

A second approach for incentivizing VC participation is an extension of the approach proposed for scaling cloud resources in [334]. The idea is to provide atomic pricing/compensation per resource used and provide incentives at the individual resource level. For simple tasks involving a small number of basic resources to be provided by edge devices, this approach seems to be very promising.

VC Security and Privacy. It goes without saying that, if VCs are to gain universal acceptance, security and privacy issues need to be suitably addressed. The establishment of trust relationships between the various players is a key component of secure computation and communication. Since some of the vehicles involved in a VC may have met before, the task of establishing proactively a basic trust relationship between vehicles is possible and may be even desirable. Research is needed on developing a trustworthy base, a negotiation and strategy formulation methodology, efficient communication protocols, data processing, etc. VS security and privacy in VC were first addressed in [523] under very general conditions; the authors pointed out novel security problems and challenges specific to VCs and proposed initial solutions to a number of the identified security and privacy problems.

Motivated by this early paper and the importance of security and privacy, a number of authors are investigating security and privacy problems in VCs. For example, Ahmad et al. [14] have identified a number of security threats at various levels of the VC architecture. Similarly, Huang et al. [247] have looked at privacy issues in VCs. It would be interesting to see if lightweight security solutions can be ported to VCs. In spite of these efforts, providing security and privacy in VC remains an open challenge.

VC support for compute-intensive applications. One of the significant research challenges in VCs is to identify conditions under which VCs can support Big Data applications. It is apparent that compute-intensive applications, with stringent data processing requirements, cannot be supported by ephemeral VCs, where the residency time of vehicles in the cloud is too short to support VM setup and migration.

Recently, Florin et al. [177] have identified sufficient conditions under which Big Data applications can be effectively supported by data centers built on top of vehicles in a parking lot. This is the first time researchers are looking at evaluating the feasibility of the VC concept and its suitability for supporting Big Data applications. The main findings of [177] are that, if the residency times of the vehicles are sufficiently long and the interconnection fabric has a sufficient amount of bandwidth, then Big Data applications can be supported effectively by VCs. In spite of this result, more work is needed to understand what it takes for VCs to gracefully support data- and processing-intensive applications.

VC support for Smart Cities. Visionaries depicted the smart cities of the future as fully connected entities supported by various forms of *road-side infrastructure,* as well as advanced in-vehicle resources, such as embedded powerful computing and storage devices, cognitive radios,and multi-modal programmable sensor nodes. As a result, in the near future, vehicles equipped with computing, communication, and sensing capabilities will be organized into ubiquitous and pervasive networks with a significant Internet presence while on the move. This will revolutionize the driving experience, making it safer, more enjoyable, and more environmentally friendly.

One of the characteristics of Smart Cities is the interconnectivity of the city's infrastructure, which allows traffic data to be collected from various human-generated or machine-generated sources or, indeed, by using the vehicles participating in the traffic. Future vehicles with powerful on-board computers, communication capabilities, and vast sensor arrays are perfect candidates in this hyper-connected environment to utilize to create a fluid VC capable of performing large-scale computations.

The way we see it, the main challenge for VCs in the context of Smart Cities should be aligned with the 2015–2019 strategic priorities recently spelled out by the US-DOT [484]. To demonstrate the relevance of VCs to Smart Cities, the following objectives were proposed:

1. *Enhance urban mobility by exploring methods and management strategies that increase system efficiency and improve individual mobility through information sharing:* VCs should combine detailed knowledge of real-time traffic flow data with stochastic predictions within a given time horizon to help: (1) the formation of urban platoons containing vehicles with a similar destination and trajectory; (2) adjust traffic-signal timing to reduce unnecessary platoon idling at traffic lights; and (3) present the driving public with high-quality information that will enable them to reduce their trip time and its variability, thus eliminating the conditions that lead to congestion or reduce its effects.
2. *Avoid congestion of key transportation corridors through cooperative navigation systems:* Congestion-avoidance techniques that become possible in SC environments will be supplemented by route guidance strategies to reduce unnecessary idling and will limit environmental impact of urban transportation.
3. *Handling nonrecurring congestion:* VCs will explore strategies to efficiently dissipate congestion by a combination of traffic-light retiming and route guidance to avoid more traffic buildup in congested areas.

Managing VC ecosystems. Recently, reference [334] made the point that, given its current trajectory, the complexity of cloud ecosystems will evolve to where traditional resource management strategies will struggle to remain fit for the purpose. Unlike conventional clouds where, for cost efficiency purposes, the compute resources are as uniform as possible, in VCs, the compute resources of individual vehicles are heterogeneous. This, in conjunction with the possible federation of VCs (suggested for the first time by [163]) and the emergence of new services that these VC will have to offer, will make it increasingly hard to manage VC resources efficiently. To the best of our knowledge, the topics of VC federation and creation of VC ecosystems have not been addressed in the VC literature. This promises to be an exciting area for future work.

Concluding remarks. It is well known that implementing cloud services, whether in conventional clouds or VCs, requires substantial architectural support ranging from virtualization, to server consolidation, and to file-system support. The amount of such support must be compared with the resulting performance; conversely, for a desired level of performance, it is of great interest to determine the level of architectural support required. This leads to interesting cost-performance tradeoffs that are fundamental to understanding the feasibility of cloud services and, therefore, cannot be ignored.

In the past decade, various VC architectures and services were outlined in terms of desirable *qualitative* characteristics without much regard to, or credible study of, their feasibility and quantitative performance characteristics. All this is changing—more and more researchers are turning their attention to *quantitative* aspects of VCs and of the services they contemplate.

The success of conventional cloud computing was due, to a large extent, to its ability to provide quantifiable (i.e., quantitative) functional characteristics such as high scalability, reliability and availability. Cloud service providers have come to equate reliability and availability with customer satisfaction and, ultimately, with revenue. By the same token, if the VCs are to see a widespread adoption, the same quantitative aspects have to be addressed here, too. Feasibility issues in terms of sufficient compute power, communication bandwidth, reliability, availability, and job-duration time are all fundamental quantitative aspects of VCs that need to be studied and understood before one can claim with any degree of certainty that they can support the workload for which they are intended.

It is our belief that reliability, availability, user job duration [177,179,180] and other similar quantitative aspects will play a fundamental contributing role in the large-scale adoption of VCs. We therefore believe that it is time for the research community to devote effort to deepen our understanding of these attributes.

13.5 Final thoughts

A cursory look at the cloud computing literature reveals the extraordinary attention given to this emerging field of computer science. Areas, such as computer architecture, concurrency, data management and databases, resource management, scheduling, and mobile computing, have bloomed in response to the need to find efficient solutions to the challenges brought about by cloud computing. Even somewhat ossified areas such as operating systems have been brought back to life by problems posed by virtualization and containerization.

Several areas critical for the future of cloud computing, including communication and security, demand special attention. Increasing the bandwidth and lowering the communication latency will make cloud computing more attractive for real-time applications and for integrations of services related to IoT. Optimization of communication protocols could lower the latency up to the limits imposed by the laws of physics. Ultimately, communication latency depends on the distance between the producer and the consumer of data.

Computer clouds and mobile devices are in a symbiotic relationship with one another and effective communication to/from clouds and inside the cloud infrastructure has to keep pace with advances in processor and storage technology. Faster cloud interconnects are also necessary to accommodate data-intensive and communication-intensive applications in need of a large number of servers working in concert. Applications in computational sciences and engineering exhibiting fine-grained parallelism would greatly benefit from lower latency.

Data security and privacy are major concerns not properly addressed by existing SLAs. Though sensitive information has been leaked or stolen from large data centers, many cloud users are unaware of the potential dangers they are exposed to when entrusting their data to a third party and trusting the protection guaranteed by SLAs.

Strong encryption protects data in storage, but processing encrypted data is only feasible for some types of queries. Most applications only operate with plaintext data; thus encrypted data has to be decrypted before processing. This creates a window of vulnerability that can be exploited by insider attacks. Hybrid clouds offer an alternative to protect sensitive information. In this case, effective mechanisms to hide sensitive information stored on the public cloud and revealed only on the private side of the cloud must be conceived.

Virtualization, in spite of its benefits, creates significant complications for software maintenance. A checkpointed virtual machine containing an older version of an operating system without the current security patches may be activated at a later time, opening a window of vulnerability that could affect the entire cloud infrastructure.

Response times plagued by a heavy-tail distribution cannot be tolerated by most interactive or real-time applications, but eliminating the tail of the latency at the scale of clouds is an enduring challenge. Another enduring challenge is reducing energy consumption and, implicitly, increasing the average server utilization. Elasticity without over provisioning requires accurate knowledge of resource consumption.

Resource reservations can help, but reservations place an additional burden on cloud users expected to know well the needs of their applications. Moreover, accurately predicting resource consumption is possible if and only if the system enforces strict performance isolation, yet another major headache for systems based on multitenancy.

Even skeptics cautioning about the dangers inherent to systems "too big to fail" have to recognize that the cloud ecosystem plays an important role in the modern society and that it has democratized computing, in the same way the web has completely changed the manner we access and use information. The Internet will continue to morph, and the web will evolve to a semantic web or Web 3.0. Thus, it is fair to expect that computer clouds will continue to change under the pressure from consumers of cloud services and from new technologies.

It is hard to predict how the cloud ecosystem will look five or ten years from now, but it should be clear that the disruptive qualities of computer clouds ultimately demand a new thinking in system design. The design of large-scale systems requires an *ab initio* preparation for the unexpected because low probability events occur and can cause major disruptions.

We have seen that the separation of control and routing planes in the Internet is partially responsible for the rapid assimilation of new communication technologies. Only a holistic approach could lead to a similar separation of concerns for computer clouds and allow computing technology to evolve at a lightning speed.

There is a glimmer of hope that machine learning, data analytics, and market-based resource management will play a transformative role in cloud computing [42,332]. As more data are collected after the execution of all instances of an application, it may be possible to construct the application profile, optimize its execution, and, ultimately, optimize the overall system performance. It may also be possible to identify conditions leading to phase transitions and prevent their occurrence that often lead to data center shutdown.

In this maze of challenges and uncertainties, there is one prediction few could argue against: The interest in cloud computing, as well as the need for individuals well trained in this field, will continue to grow for the foreseeable future. The incredible pace of developments in cloud computing poses its own challenges and demands the grasp of fundamental concepts in many areas of computer science and computer engineering, as well as curiosity and the desire to continually learn.