IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON BLOCKCHAIN-ENABLED TRUSTWORTHY SYSTEMS

Received June 29, 2020, accepted July 11, 2020, date of publication July 17, 2020, date of current version July 29, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3010119

ChainFaaS: An Open Blockchain-Based Serverless Platform

SARA GHAEMI[1], (Graduate Student Member, IEEE), HAMZEH KHAZAEI[2], (Member, IEEE), AND PETR MUSILEK[1,3], (Senior Member, IEEE)

[1]Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Alberta, AB T6G 2R3, Canada
[2]Department of Electrical Engineering and Computer Science, York University, Toronto, ON M3J 1P3, Canada
[3]Department of Applied Cybernetics, University of Hradec Králové, 50003 Hradec Králové, Czech Republic

Corresponding author: Sara Ghaemi (sghaemi@ualberta.ca)

This work was supported in part by the Future Energy Systems Research Initiative under the Canada First Research Excellence Fund (CFREF) at the University of Alberta, Canada, in part by the Compute Canada, and in part by the SAVI Testbed Cloud.

ABSTRACT Due to the rapid increase in the total amount of data generated in the world, the need for more computational resources is also increasing dramatically. This trend results in huge data centers and massive server farms being built around the world, which have a negative impact on global carbon emissions. On the other hand, there are many underutilized personal computers around the world that can be used towards distributed computing. To better understand the capacity of personal computers, we have conducted a survey that aims to find their unused computational power. The results indicate that the typical CPU utilization of a personal computer is only 24.5% and, on average, a personal computer is only used 4.5 hours per day. This shows a significant computational potential that can be used towards distributed computing. In this paper, we introduce ChainFaaS with the motivation to use the computational capacity of personal computers as well as to improve developers' experience of internet-based computing services by reducing their costs, enabling transparency, and providing reliability. ChainFaaS is an open, public, blockchain-based serverless platform that takes advantage of personal computers' computational capacity to run serverless tasks. If a substantial number of personal computers were connected to this platform, some tasks could be offloaded from data centers. As a result, the need for building new data centers would be reduced with a positive impact on the environment. We have proposed the design of ChainFaaS, and then implemented and evaluated a prototype of this platform to show its feasibility.

INDEX TERMS Blockchain, distributed ledger technology, serverless computing, green computing, distributed computing.

I. INTRODUCTION

Due to the increasing need for more computational resources, many data centers with massive server farms have been built around the globe. Data centers around the world consumed about 416 terawatts of electricity during 2016, which was about 3% of the world's electricity consumption. This amount was also about 40% more than the consumption of the entire United Kingdom in the same year. This energy expenditure is expected to double every four years [1]. In another study, Jones [2] has found the contribution of the data centers in the global carbon emission to be about 0.3%, and the entire information and communications technology (ICT) ecosystem's contribution to be about 2%. It is hard to predict what the future holds, but in a worrying recent study conducted by Belkhir and Elmeligi [3], the information and communication industry's global carbon footprint is estimated to reach about 6-14% of the total worldwide footprint by 2040. This calls for new solutions to reduce the carbon emission of this industry.

On the other hand, hundreds of million units of personal computers are manufactured worldwide every year, each leaving their own share of emissions [4]. These computers are highly underutilized both in industries and in households. At any given time, they are either not being used at all or running on a small fraction of their capacity. Based on our survey, which is explained in Appendix, personal computers run on an average CPU utilization of 24.5%. Moreover,

The associate editor coordinating the review of this manuscript and approving it for publication was Patrick Hung.

the survey also shows that, on average, personal computers are only being used 4.5 hours per day. These numbers confirm the initial assumption that personal computers are highly underutilized.

The idea behind the proposed approach in this paper, called ChainFaaS, is to use the untapped computational power of the current computers as a serverless platform. In ChainFaaS, regular computer users can rent out their unused computational power by connecting it to a large network of resources. On the other hand, those who need computing resources can rent from this vast pool of compute at scale. If enough personal computers were connected to ChainFaaS, the need for building new data centers would decline. However, the goal in ChainFaaS is not to replace the data centers and servers but to only offload some of their tasks by reusing the idle cycles of personal computers.

Another motivation for creating ChainFaaS is to improve developers' experience of internet-based computing services. Currently, cloud giants, such as Amazon, Google, and Microsoft, are the leading players in serverless computing. Serverless computing platforms offered by these companies are highly centralized, and such companies control every single detail of the platforms. There is no way for the developers to verify the reported billing information. On the other hand, ChainFaaS offers a low-price, transparent, reliable and easy-to-use serverless platform which is not managed by one entity. Anyone can join the network to participate in the management of the platform as well as to observe the transactions. Moreover, since in ChainFaaS, functions run on personal computers, it can be a great platform for edge and fog computing. Developers who need to run IoT applications closer to end-users can use this platform instead of other computing services that run on servers.

For developers, one of the main incentives to use serverless computing platforms is to reduce their costs. In serverless, unlike in other cloud computing solutions, billing is based on the program execution time rather than on the provisioned capacity. In ChainFaaS, one of the motivations is to reduce these costs even more by running the tasks on excess computation power of personal computers.

ChainFaaS is designed with the motivation to provide a reliable and transparent serverless platform. Reliability is achieved through the dispersion of nodes in the network. Since the computing providers can be located anywhere in the world, and a large number of computers are available, ChainFaaS is reliable by design and sheer scale. Moreover, transparency is achieved through the use of blockchain. Every single transaction on the network is recorded on the immutable ledger of the blockchain. Blockchain peers keep a record of all the changes and every user can easily access this information at any time and verify the transactions.

Rather than being managed by a central entity, the transparent public network of ChainFaaS uses blockchain. At first, this may seem to contrast with the initial motivation of ChainFaaS: to decrease carbon emissions of the ICT ecosystem. The main reason for this apparent conflict is that most well-known blockchains, based on proof-of-work consensus algorithms, require miners to complete computationally intensive tasks. For instance, the carbon emissions corresponding to the electricity consumption of Bitcoin is estimated to be about 22.0 to 22.9 $MtCO_2$, somewhere between the amounts produced by the nations of Jordan and Sri Lanka [5]. However, not all blockchains have such a negative impact on the environment. Many blockchain solutions have been introduced that do not require computationally intensive tasks and instead operate based on other consensus algorithms such as proof-of-stake, delegated proof-of-stake, practical Byzantine fault tolerance, and many others. The blockchain used in the prototype of ChainFaaS is Hyperledger Fabric, which works based on practical Byzantine fault tolerance consensus algorithm. As a result, the use of Hyperledger Fabric-based blockchain is in tune with the design goals of ChainFaaS.

To sum up, ChainFaaS offers an open blockchain-based serverless platform with the following features:

- It is public in the sense that anyone can be either a developer, provider or both.
- It is open and transparent to everyone.
- It is based on the excess computing power available on personal computers.
- It is affordable for developers, especially compared to similar centralized (in terms of management) cloud solutions.
- It is user-friendly and easy to use.
- It embodies edge and fog computing by nature as ChainFaaS can get very close to the end-user.

In this paper, our objective is to design a public serverless platform with the above mentioned features, and to investigate the feasibility of the design. We first describe a high-level architecture of the platform and introduce the stakeholders. Then we assess the functional and non-functional properties of the platform and present a more detailed design. To examine the feasibility of the proposed platform, we follow a proof-of-concept approach by implementing a prototype of ChainFaaS. The implementation is based on a microservices architecture to better capture the needs of serverless computing. Finally, the performance of the prototype is evaluated in a series of experiments.

The rest of this article is organized as follows. Section II explains serverless computing and blockchain as the two main technologies used in ChainFaaS. It also outlines the related work done in this field. Section III presents the design details and architecture of ChainFaaS. Section IV explains the implementation and deployment details of the prototype of ChainFaaS. Section V discusses the functional and non-functional evaluation of this prototype. In Section VI, the potential threats to the validity of the design and evaluation of ChainFaaS are discussed. Section VII summarizes the conclusions of this paper.

II. BACKGROUND AND RELATED WORK

Before describing the design and implementation details of ChainFaaS, two main technologies used in this platform, namely serverless computing and blockchain, and the related work are explained in this section.

A. SERVERLESS COMPUTING

To understand serverless computing, we should first define cloud computing. Cloud computing is defined as a model in which a shared pool of configurable computing resources can be accessed conveniently and on-demand through the internet. These resources can be networks, servers, storage, applications, or services that are provisioned and released with minimal effort and service provider interaction [6].

Based on the developer's control level over the cloud infrastructure, a few service models are available for cloud computing. In the infrastructure-as-a-service (IaaS) model, the developer has control over both the application and the cloud infrastructure. The developer should manage the virtual machine provided by the cloud provider and the deployment of the application on top of the underlying virtual machine. In platform-as-a-service (PaaS), the developer manages the application code, and the provider maintains the infrastructure. In software-as-a-service, full applications are delivered over the internet to end-users via providers. Both the infrastructure and the application are managed by the provider [7].

Serverless computing (also known as function-as-a-service or FaaS for short) is the latest paradigm in cloud computing in which the developer deploys functions to the cloud and delegates the operational and management tasks of servers to the provider [8]. The developer can focus on designing and implementing the application instead of spending time on the management, operation, and maintenance of the infrastructure. This characteristic makes serverless similar to PaaS solutions. However, in serverless, costs are truly event-based, unlike in PaaS. This means that the developers only pay for the execution time of their program. In other cloud computing solutions, the developers are billed for the resource allocated to their tasks, even if they are not used.

Another feature of serverless is autoscaling. When the developer deploys a function to serverless computing, they do not need to worry about how their code should scale. No matter how many times concurrent events to the function are triggered, the serverless provider will serve them by running new instances.

Moreover, in serverless computing, everything is based on containers or similar concepts, i.e., lightweight, isolated environments. Since containers are usually small in size and fast to start, every time an event is triggered in serverless computing, the cloud provider can spin up a new container on a virtual machine. In a serverless platform, it can take a few hundred milliseconds to a few seconds to serve a request. For some use cases, such as consumer-facing applications, this time is not acceptable. For instance, imagine a website that wants to use serverless to serve requests. For each request, the users should wait at least a few hundred milliseconds for the container to spin up and then a few more for the function to run. Based on these characteristics, serverless computing cannot currently be used for all types of workloads. Serverless providers advise developers not to submit functions that are time-sensitive since they cannot guarantee a fast response.

The above-mentioned features make serverless computing a great solution for tasks that are not time-critical. Currently, cloud giants are the main providers of serverless computing. Some examples of public serverless platforms are AWS Lambda [9], Azure Functions [10], Google Cloud Functions [11], and IBM Cloud Functions [12].

B. BLOCKCHAIN

For a public platform that anyone can join, there needs to be a management system that everyone trusts. In most cases, there is a company or entity that controls the transactions, and people trust that company (for example, a bank). In the case of ChainFaaS, blockchain is used instead of a trusted entity. Blockchain is a type of distributed ledger technology that has recently become extremely popular. It is mainly known for its use in cryptocurrencies such as Bitcoin [13], Ethereum [14], Ripple [15]. However, blockchains, and distributed ledger technologies in general, are more than just cryptocurrencies. In a recent study [16], CB Insights has reported 55 industries that can be transformed by the distributed ledger technology. Some examples are biomedical and health care [17]–[19], energy [20], Internet of Things (IoT) [21], [22], supply chain [23], [24], and cloud computing [25].

In terms of the data architecture, there are two main types of distributed ledger technologies: blockchain and directed acyclic graph (DAG). In a blockchain, transactions are bundled in blocks of data that are linearly chained to each other, just like a linked list. The blocks are connected in a chronological order, which is unalterable. On the other hand, in DAG, transactions are linked to each other in a graph. Usually, there is no block in DAG, and transactions are the components of the DAG [26]. In this paper, we mainly focus on the features and characteristics of blockchains, since they are used in ChainFaaS.

Distributed ledgers can be categorized as permissioned and permissionless, based on whether the identities of the participants are known. If the identity of everyone is known, the ledger is permissioned, while if everyone participates in the network anonymously, the ledger is permissionless. Also, based on who can participate in the network, distributed ledgers can be categorized as private and public. In private ledgers, only those who are approved can participate in the network. On the other hand, in public ledgers, anyone can participate without the need to be validated. For example, Bitcoin runs on a public permissionless blockchain.

Blockchain is a distributed database that is immutable, transparent, verifiable, and managed by a set of participants who do not necessarily trust each other. At any point in time, each transaction can be executed and verified by the participants based on a consensus algorithm. In public

permissionless blockchains, this consensus algorithm is usually a computationally expensive task that takes a lot of time and energy. This type of consensus algorithm is called Proof-of-Work (PoW), which limits the speed of the transactions. Although other solutions for public blockchains have been introduced, most of them are slow in terms of the number of transactions that they can verify per second. On the other hand, in permissioned blockchains, since the identities are known, the consensus algorithm can be less costly, which makes them dramatically faster. Usually, these blockchains are used between participants who have the same goal but do not necessarily trust each other. In a blockchain, each block consists of many transactions bundled together and a header, which consists of a hash. For each block, the hash is a unique value that is created based on the data inside that block and the previous block's hash. The hash is the element that makes blockchain an immutable chain of data. No one can change the transactions in the blocks that have already been verified because of this hash.

In blockchain, there is a concept called smart contract, which allows customization of the contract between the entities in the blockchain. Szabo first introduced the term smart contract in 1994 as a computerized transaction protocol that executes the terms of a contract [27]. Within blockchain, smart contracts are the pieces of code that contain the contract between entities and are stored on the blockchain. Participants can trigger these contracts to use their functionalities [28]. Using smart contracts on blockchains, trusted distributed applications can be created.

In ChainFaaS, there is no need for the participants to be anonymous. As a result, compared to a permissionless blockchain, a permissioned blockchain is a better solution since it is faster. Moreover, the goal of ChainFaaS is to help decrease the carbon emission of the information and communications technology ecosystem. Stoll *et al.* [5] recently found the carbon emission corresponding to the electricity consumption of Bitcoin to be 22.0 to 22.9 $MtCO_2$. This means the emission level of Bitcoin is somewhere between the amounts produced by the nations of Jordan and Sri Lanka [5]. Therefore, blockchains that use PoW, such as Bitcoin, cannot be used for ChainFaaS. In addition, the blockchain must support smart contracts since we need to implement customized contracts for ChainFaaS. To sum up, for ChainFaaS, we need a permissioned blockchain with a small carbon footprint that supports smart contracts.

Hyperledger Fabric [29] has all the required features and more. In Hyperledger Fabric, participants are identifiable, the transaction throughput is high, transaction confirmation is fast, smart contracts are supported, and all network details are highly configurable [30].

C. RELATED WORK

1) PUBLIC-RESOURCE COMPUTING

The idea of using the excess resources of personal computers in a distributed computing platform is not new. The first public-resource computing projects started in the mid-1990s. Great Internet Mersenne Prime Search (GIMPS) [31] and distributed.net [32] were two of the very first project in this area. Both projects started to work on solving research questions on personal computers and they are still actively running. In 1999, SETI@home [33] was released, attracting millions of volunteers. These are just a few examples of many projects running on personal computers. However, there are only a few frameworks that enable the creation of such projects, and most of them only focus on research projects.

Public-resource computing (also known as global computing) refers to any distributed computing platform that uses the idle processing power of personal devices. Most known public-resource computing platforms are based on volunteer computing, a type of distributed computing in which volunteers contribute their idle computational power to perform computationally expensive research tasks [34]. Because of the heterogeneity of the computers in such a network, a task scheduler is needed to properly distribute the chunks of a research project on personal computers. This task distribution allows a peer-to-peer network to be formed that enables users to share their resources [35]. It is worth mentioning that volunteer computing is different from grid computing. In grid computing, the computing resources are managed and owned by organizations, whereas, in volunteer computing, the resources are highly unreliable and are managed by non-expert individuals.

One of the most well-known volunteer computing frameworks is the Berkeley Open Infrastructure for Network Computing (BOINC) [36], [37], which was first introduced in 2002. Currently, more than 700,000 devices participate in the BOINC network, clearly demonstrating the potential that personal computers hold. This framework consists of a central server and clients run on the volunteers' computers. Any project that wants to use BOINC has to host their own server and run the central BOINC server. This makes it extremely hard for developers to use this platform. Moreover, volunteers can choose the projects they would like to contribute to. As a result, the developers have to ensure that their projects gain enough visibility to attract volunteers. This makes it hard to predict if the project receives enough resources.

XtremWeb [38] is another popular volunteer computing framework, which is the base of XtremWeb-HEP. As explained in Section II-C2, XtremWeb-HEP is used in the iExec project, which is a blockchain-based cloud computing. In XtremWeb, collaborators can host their own volunteer computing platform that can cooperate with the main Xtremweb server. This framework has three main components: the workers, the client, and the coordinator. The workers are volunteer computers that are responsible for executing the tasks. The client is the entity that submits the tasks to the network. Finally, the coordinator distributes the tasks, assigns workers to the tasks, and keeps track of them.

As another popular distributed computing framework, Cosm [39] provides a set of tools and libraries for distributed applications. This framework was introduced in 1995 and

research projects such as Folding@Home [40], [41], and Genome@Home [42] used this platform. Since 1999, Folding@Home project has been running protein folding on personal computers to find how proteins work and to find cures for diseases. Genome@Home designed genes that match existing proteins from 2000 to 2004.

In the recent years, the development of cryptocurrencies has led to some interesting projects that enable payment to volunteers in public-resource computing platforms. Some research projects may choose to reward volunteers for participation in their project. FoldingCoin [43] and CureCoin [44] provide reward tokens for participants of the Folding@Home network. GridCoin [45] offers reward to volunteers in the BOINC platfrom.

Although the mentioned solutions are similar to ChainFaaS, there are some fundamental differences between them. Most of the current solutions are based on volunteer computing. In volunteer computing, the providers do not expect any income for the computations they execute. Moreover, the computing providers trust that the research tasks running on their computer would not tamper with their files and programs. This is mainly due to the fact that the research owners are known universities. On the other hand, in ChainFaaS, there is no trust in the developers, and the computing providers expect the income to be worth their time and effort. Also, in most volunteer computing platforms, the developers need to host their own servers and promote their projects to gain popularity. This makes it hard for developers to adopt such solutions. Nevertheless, since ChainFaaS is a serverless computing platform, the developers only need to submit their functions to the network through the web application, and the network handles the rest. These characteristics make the design and implementation of serverless platforms, such as ChainFaaS, more challenging. Due to these fundamental differences, these platforms cannot be compared to ChainFaaS directly, especially in terms of performance.

The mentioned drawbacks of volunteer computing platforms have prevented their large-scale adoption. Despite the great potential that platforms such as BOINC have, they are still only being used by a small group of people. As mentioned earlier, the volunteers do not have any economic incentive to participate in the network and the developers have to spend much time modifying their program (i.e. high degree of customization) to be suitable to run on these platforms. Since these problems have been addressed in our design, ChainFaaS has a good potential to receive attention from both developers and providers.

2) BLOCKCHAIN-BASED CLOUD COMPUTING

Recently, there have been a number of research activities on distributed cloud computing platforms that run on personal computers. These distributed systems have many management barriers that can be solved using blockchain. In this section, we discuss some of the well-known blockchain-based cloud computing platforms.

The iExec [46] platform is a distributed cloud computing infrastructure, which is based on XtremWeb-HEP [47] and Ethereum smart contracts. XtremWeb-HEP is an open-source desktop grid software that is developed by the same team. The Ethereum blockchain provides the support for distributed applications. For better performance, a new consensus algorithm called Proof-of-Contribution (PoC) is proposed for use in this platform. In iExec, there are three different types of providers: application providers, computing providers, and data providers. Application providers are developers who want to use the platform to run their distributed applications. Similar to ChainFaaS, computing providers rent out their unused CPU cycles. Data providers can make their datasets available to others and charge them for each use. In this platform, the developers should divide their application into tasks and send them to the scheduler.

Golem project [48] aims to create a decentralized supercomputer. Golem is designed to use excess computational power on PCs to run heavy tasks such as deep learning or video rendering. However, currently, the only tasks that are accepted in this network are rendering tasks. In Golem, developers specify their price suggestions, and providers bid on the tasks. One of the limitations in the Golem network is the fact that computing providers are required to have public IPs, which prevents many from participating in the network.

Another project in this area is SONM [49], which is a decentralized fog computing platform. Similar to the previous two projects, in SONM, people in need of computational power are connected to those with excess computational capacity through a blockchain network. The main focus of this project is on IaaS, and the main applications are machine learning and video rendering.

CloudAgora [50], [51] is an academic project that proposes a blockchain-based platform providing access to storage and computing infrastructure. In this platform, providers bid on any incoming request in an auction-styled manner. The provider that offers the lowest price will be selected to execute the task. The current prototype of CloudAgora is implemented on Ethereum blockchain.

In their paper, Uriarte and De Nicola [52] have surveyed blockchain-based cloud computing solutions and explained the projects in detail. They have also studied the challenges and standards in this field. In another recent research, Yang *et al.* [53] have studied the challenges and research issues of blockchain and edge computing integration. Westerlund and Kratzke [54] surveyed the advantages and disadvantages of both centralized cloud computing platforms and distributed ledger technologies. They then identified the principles that a distributed cloud platform should follow to use the advantages of both traditional clouds and DLTs. ChainFaaS's design is in accordance with these proposed principles.

All the projects mentioned above are similar to ChainFaaS in the sense that they use personal computers to run developers' tasks in a blockchain-based network. However, none of them are serverless platforms. The developers have

to spend a lot of time modifying their applications for these platforms. In some platforms, developers even have to spend time selecting the provider for their job, which can be inconvenient. Moreover, in most of these platforms, it is hard for the providers to connect to the network. For instance, in Golem, they need to have a public IP, which may not be possible for most providers. In ChainFaaS, these problems are addressed, making the new platform easy to use for everyone. Unfortunately, none of the blockchain-based cloud computing solution providers have released performance analyses of their platform, and they cannot be directly compared to ChainFaaS.

III. SYSTEM DESIGN

A. HIGH-LEVEL ARCHITECTURE

Fig. 1 shows a high-level architecture of ChainFaaS. This platform has three main parts. The blockchain network consists of all the blockchain peers who are responsible for keeping track of the transactions on the platform. The serverless controller manages the cloud portal and the job scheduling task. The computing resource providers cooperate to shape the execution network.

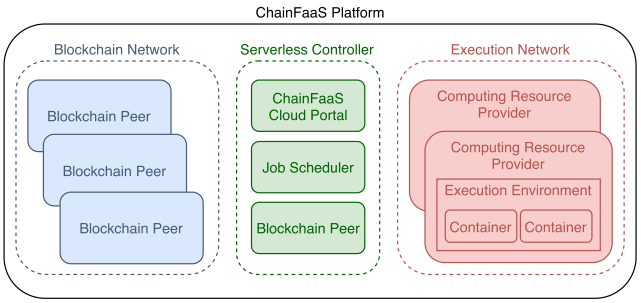

FIGURE 1. High-level architecture of ChainFaaS. The blockchain network stores and confirms all the transactions. The serverless controller is a blockchain peer itself and handles the cloud portal and scheduling the jobs. The execution network consists of all the computing resource providers.

1) BLOCKCHAIN NETWORK

The blockchain network has two main tasks: keeping records of all transactions and managing payments. From now on, we will refer to the record-keeping ledger and payment management ledger as *monitoring ledger* and *monetary ledger*, respectively. Every single transaction is stored and kept by all blockchain peers on the monitoring ledger. Any user can easily access the information of a job and see how it has evolved over time. Information such as the owner of the job, the computing resource provider who has been responsible for the job, the time it has taken to run the job, and its cost, are accessible through the monitoring ledger. The transparency feature of ChainFaaS is achieved through the monitoring ledger. The monetary ledger stores the financial account information and account balances of the users. Every time a job is successfully executed by a provider (i.e., a personal computer owner), this monetary ledger automatically transfers the cost from the developer's account to the provider's account.

The blockchain network comprises many peers that can be owned by different proprietors. Each peer keeps records of all the transactions occurring in the network. To add a new transaction to the ledger, the peers should reach a consensus on whether to accept the transaction or not. The peers receive a commission for each transaction because of their contribution to the network. It is worth mentioning that a blockchain peer can also act as a computing resource provider in the execution network, which results in having two sources of income from ChainFaaS. The execution network is described in Section III-A3.

2) SERVERLESS CONTROLLER

The serverless controller acts as the gateway and is responsible for providing the portal of ChainFaaS, publicly available here.[1] This portal is designed to help all users easily interact with the platform. Based on their enrolment in the system, users can submit new jobs, observe the status of their job, and monitor their contribution to the network. Moreover, the controller handles the job scheduling by finding a computing resource provider for each job. The job scheduling is based on a selection algorithm that may take into account many criteria to choose the provider, including the provider's availability and the computational power needed for the job. The serverless controller also acts as one of the blockchain peers and, just like any other blockchain peer, it receives a commission for the transaction.

3) EXECUTION NETWORK

The execution network consists of all computing resource providers. Anyone can easily connect their extra computational resources or their underutilized online computers to this network. This includes but is not limited to, personal computers, servers, and cloud resources. These providers get paid based on the time they spend on running the job to which they are assigned. Each job runs in an isolated execution environment on the provider's computer to ensure that it does not interfere with the provider's programs. This is also required to prevent different jobs from interfering with each other.

B. STAKEHOLDERS OF ChainFaaS

1) DEVELOPER

A developer, or software owner, is an individual or a company that wants to submit a function to the serverless platform. In this role, the user wants to use an affordable function-as-a-service system to decrease their infrastructure costs and server management overhead.

2) COMPUTING PROVIDER

A provider is someone who wants to rent out their computer's idle CPU cycles and memory to earn money. The goal of this user is to serve as many jobs as possible to increase their income.

[1] https://chainfaas.com

3) BLOCKCHAIN PEER

Any computer with public IP address could act as a blockchain peer. The peers are responsible for running and storing the blockchain on their computers. Other blockchain users send requests to these peers to interact with the blockchain. As an incentive to participate in the network, these peers receive a portion of the transactions when they serve requests. In addition, the blockchain peers can also participate in ChainFaaS as computing providers to increase their income from the platform.

C. FUNCTIONAL PROPERTIES

A detailed architecture of ChainFaaS with a complete description of the process of serving a request is shown in Fig. 2. The developer is the owner of the function who can have clients sending requests to their function. These clients can be the developer themselves, a program owned by the developer, or anyone else. The developer can also choose to store the result of their function in a separate storage. This can be specified in the container they upload to the system. The result storage block in Fig. 2 represents this storage unit.

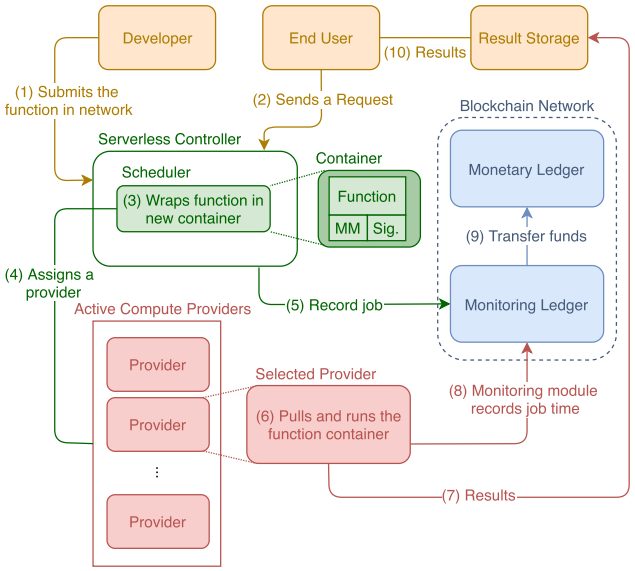

FIGURE 2. The detailed architecture of ChainFaaS and the way a request is served in the network (MM stands for monitoring module and Sig. stands for signature.)

The scheduler in the serverless controller is responsible for scheduling the jobs, i.e., functions, and computing providers. It creates a new container that includes the function, a monitoring module (MM), and the controller's signature (Sig.). The module is designed to send back the run-time metrics of the job to the monitoring ledger. The signature is used to verify the container's creator to be the controller and not a malicious entity. When an appropriate provider is found for the job, the controller asks the monitoring ledger to keep a record of the job. The job ID stored there should be the same as the one the monitoring module later uses to record the job service time. The assigned provider is the only one who can have the job's run time recorded on the ledger. A detailed explanation of each step in Fig. 2 follows:

1) For a function to be available in the system, the developer first needs to submit it to ChainFaaS. To do so, the developer sets the access link and characteristics of the function in the ChainFaaS cloud portal. From the moment the function is submitted, others can send requests for that function to the serverless controller.
2) As soon as the controller receives a request for a function, a job is created, and the process starts.
3) The scheduler then adds the monitoring module (MM) to the function and wraps it in a new container with the controller's signature. The monitoring module is responsible for sending the job processing time to the monitoring chain in step 8. The signature is used in the provider to verify that the job has been sent from the controller.
4) The next step is for the scheduler to assign an appropriate provider to the job. This assignment is done based on a selection algorithm that takes into account the computational capability of the provider and its availability.
5) When a computing provider is found for the function, the controller sends a record-request to the monitoring ledger. In this request, the controller asks the blockchain network to add a new job to their records. The information added to the record includes the job ID, the developer of the job, and the provider assigned to the job. The detailed explanation of the information stored about each job in our implementation can be found in Section IV-C.
6) The selected provider then pulls the container from the registry and runs it.
7) In the next step, the provider sends back the results to the target storage.
8) The monitoring module then uses the job ID received from the controller to ask the monitoring ledger to keep a record of the job's run time metrics.
9) In the next step, the monitoring chain charges the developer's account by transferring the amount of bill to the provider's account on the monetary ledger.
10) Finally, the result storage, which is managed and owned by the developer, sends back the results to the end-user. The developer can have the end-user manually get the result from the results storage or have the storage share the results with end-user whenever available.

D. NON-FUNCTIONAL PROPERTIES

This section evaluates the most important non-functional properties of ChainFaaS: performance, availability, security, and usability.

1) PERFORMANCE

One of the most important properties of a software system is performance. To understand the performance metrics of

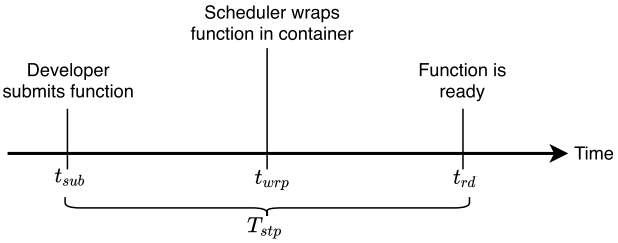

FIGURE 3. The timeline for submitting a function to ChainFaaS from the developer's point of view.

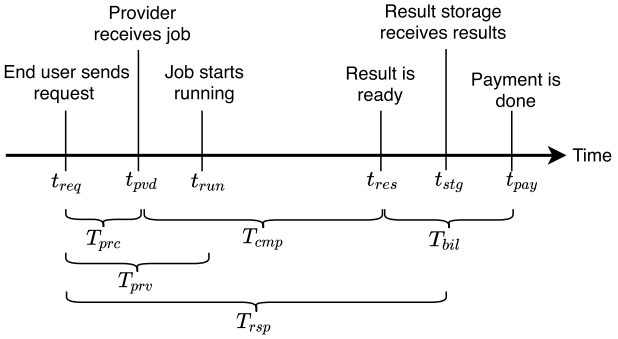

FIGURE 4. The lifetime of a request in ChainFaaS.

ChainFaaS, consider the timelines shown in the following two figures. Fig. 3 shows the timeline for submitting a function to ChainFaaS. From the developer's point of view, the time it takes for the function to become ready in the system after it has been submitted is a critical factor. It is called setup time $T_{stp} = t_{rd} - t_{sub}$.

A new job is created in the system as soon as a request for a function is received by the controller, as shown by t_{req} in Fig. 4. The time it takes for the result storage to receive the result of the request is called response time $T_{rsp} = t_{stg} - t_{req}$. This time depends on both the request processing time of ChainFaaS and the completion time. The request processing time, T_{prc}, is defined as the time it takes for the provider to receive the job after the controller receives a request for the function. It corresponds to the time the serverless controller requires to schedule the job. The time it takes for the provider to pull and run the function is called completion time, T_{cmp}. This time depends heavily on the size of the developer's container, the network delay, and the job itself. Since response time depends on the function, it can only be measured for a specific workload and not in general. Response times of sample workloads are shown in Section V.

Another important performance metric in this system is the provisioning time, T_{prv}. It is the time it takes for the job to start when a request is received. The provisioning time consists of the request processing time and the container pull time. Finally, from the provider's point of view, the time it takes for them to receive the payment after the job is finished, T_{bil}, is of great importance.

The most important performance metrics of ChainFaaS are summarized below:

- *Setup time (T_{stp})* is the time it takes for the function to become available in the system after it is submitted.
- *Request processing time (T_{prc})* is the time it takes for the provider to receive the job after the serverless controller receives a request.
- *Provisioning time (T_{prv})* is the time it takes for the job to start running after the serverless controller receives a request.
- *Completion time (T_{cmp})* is the time it takes for the provider to pull and run the job's container.
- *End-to-end response time (T_{rsp})* is the time it takes for the result to be available in the result storage after the serverless controller receives a request.
- *Billing time (T_{bil})* is the time it takes for the provider to receive payments after they are finished running the job.

2) AVAILABILITY

In a software system, availability is defined as the probability that the user receives a response for their request at a given time. There could be two main causes for jobs to be blocked in ChainFaaS: insufficient computing resources and job queue being full. When a new job is created in the network, there should be an available computing provider who has enough resources for the job. If the scheduler is unable to find a provider, the job request will be blocked.

In ChainFaaS, requests to functions will be queued until a provider is found; requests are served according to a queuing policy. When the queue is full, the serverless controller will block the new incoming requests. If the queue size is too big, some end-users may experience long response times. In such cases, the usual approach is to limit the queue size to prevent that from happening by immediate blocking.

3) SECURITY

Since ChainFaaS is open to the public, security is of paramount concern. As a computing provider, the user needs to be assured that the code running on their computer is not going to harm their system or access their personal data. An open platform such as ChainFaaS should run completely isolated from the rest of the programs on the provider's computer. For this isolated environment, access to data and information should be restricted. The way ChainFaaS achieves this quality is explained in detail in Section IV.

Moreover, all stakeholders are storing important information, such as their account balance, on this platform. Everyone should trust the system to be secure. ChainFaaS uses blockchain to ensure the security of shared information. Since blockchain is immutable, transparent, and secure, it can be used for this platform. No one, not even the controller, can change the information stored on the blockchain, and every single member can see all the transactions in the blockchain and verify their validity. All these characteristics make blockchain a great solution to open public platforms such as ChainFaaS.

4) USABILITY

ChainFaaS is designed to be used by anyone. Although it is a sophisticated platform based on the state of the art technology, the users are not necessarily computer professionals. Therefore, this system needs to be user-friendly. It should be designed in such a way that anyone, including users with no background in computer science, can easily become a computing provider or blockchain peer. To achieve this goal, ChainFaaS has a web portal along with easy-to-setup agents to be installed on computing nodes, as explained in Section IV.

E. BUSINESS MODEL

ChainFaaS has a few stakeholders that may benefit from this platform: the computing providers, the blockchain peers, and the serverless controller. When designing the system, their benefit should be considered an important factor. The computing providers get paid based on the computational power they contribute to the platform and the time they spend on running functions. Since the providers contribute their idle computing cycles, whatever they make is considered profit. Providers can easily connect their computers to ChainFaaS and rent out their excess cycle without any interference in their usual work. As the network extends in size, the providers' profit is likely to increase.

The blockchain peers get a portion of the transaction fees as an incentive for running the blockchain network. The reason blockchain peers are separate from the computing providers is the necessity to have a public IP for blockchain peers. All the peers should be accessible by a public IP so that everyone can send requests to them. Anyone can own a blockchain peer in the network and help keep the blockchain network secure and transparent.

IV. IMPLEMENTATION AND DEPLOYMENT

To demonstrate the feasibility of our design, we have implemented a prototype of ChainFaaS as a proof-of-concept. In this section, we present the details of the implementation and deployment of this prototype. Details of ChainFaaS implementation, including the complete code base, can be found on GitHub .[2] Fig. 5 shows the implementation and deployment view of ChainFaaS. The end-user represents the person who sends a request to a function. The end-users are FaaS users that can be the developers themselves, or their users who want to access the functions provided by the developer on ChainFaaS.

A. SERVERLESS CONTROLLER

As explained earlier in Fig. 1, the controller has three main parts: cloud portal, job scheduler, and blockchain peer. The cloud portal is responsible for all interactions between the controller and the users. At the back-end of the web application, the job scheduler is also implemented. We have implemented a simple scheduler that randomly selects a provider

[2]https://github.com/pacslab/ChainFaaS

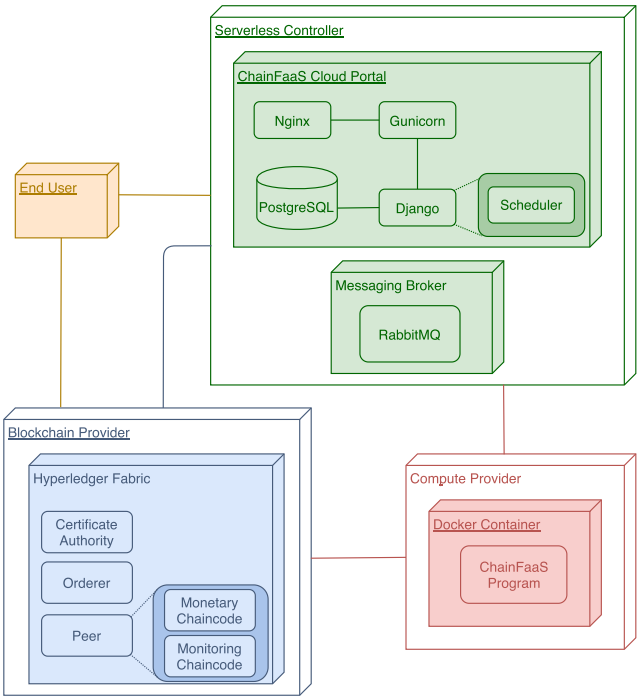

FIGURE 5. Deployment diagram of ChainFaaS showing the technologies and components used in different parts of the platform

from the available providers that can fit the request. The blockchain peer is just like other blockchain peers described in Section IV-C.

In our implementation of ChainFaaS, the serverless controller is running on an instance with 4 VCPUs, 15GB RAM, and 83GB disk with Ubuntu 18.04. The cloud portal is written in Python using the Django web framework [55]. The job scheduler is also implemented in the backend of the cloud portal. Gunicorn is used as the application server, and Nginx is used as a reverse proxy. Gunicorn is a Python Web Server Gateway Interface (WSGI) HTTP server that communicates with the Python application. Gunicorn optimally creates as many instances as needed from the web application, distributes the requests between them, and restarts them if necessary. Nginx is used as a reverse proxy to Gunicorn to handle all incoming requests.

New users can use the ChainFaaS cloud portal to register. Each user should select what role they want to play in the platform: developer, provider, or blockchain peer. After registration is complete, the user can start interacting with the platform. As a developer, the user can submit the link to their Docker container in the Docker registry and set its characteristics.

When the scheduler matches a provider with a job, there should be a way for the serverless controller to inform the provider of the link to the function and all its information. Since providers are personal computers, they are not directly accessible by the controller. As a result, in ChainFaaS, a messaging queue has been implemented to manage the interactions between the serverless controller and each provider. RabbitMQ has been chosen as the messaging broker for this

system since it is a lightweight open-source broker that can be customized for different applications. During the registration step, the providers are also registered in the messaging broker to be able to access the appropriate queue. Each provider has its own queue that only the serverless controller and the provider can access. The serverless controller puts the jobs in the provider's queue, and the provider fetches it from the queue.

B. COMPUTE PROVIDER

On providers' computers, ChainFaaS runs a program that is written in the Python programming language. As long as the program is running on the provider's computer, it receives new jobs from the serverless controller, runs them, and waits for other jobs. Jobs run in isolated environments using Docker containers to keep them from interfering with the provider's computer and accessing their files. In ChainFaaS, having isolated environments for the jobs is particularly important since the provider may not necessarily trust the source of the job. Moreover, there may be more than one job running on the provider's computer at the same time. Using sandboxing solutions, we can ensure there is no conflict between the dependencies and resources of the jobs.

A popular solution for isolating the execution environment of software is using virtual machines. In this solution, a guest operating system runs on top of a host operating system and has virtual access to the system's underlying hardware. Another solution is using containers, which also provide an isolated environment for running a software service. Unlike virtual machines that virtualize the hardware stack, containers provide the developers with a logically isolated operating system by virtualizing the computer resources at the OS-level. As a result, compared to virtual machines, containers are far more lightweight, faster to start, and use far less memory.

In the current implementation of ChainFaaS, the security and privacy concerns of providers are addressed using containers as the sandboxing solution. The reason for this choice is that the providers should be able to start using the platform with minimal effort and overhead. As discussed earlier, containers are fast, lightweight, and they provide the required isolated execution environment. Other open distributed computing platforms have different solutions to this security challenge. In BOINC [36], [37], the provider can choose to run the project applications under an unprivileged account on the operating system, which cannot access or modify data other than its own. This is called account-based sandboxing. This type of sandboxing is less scalable and secure compared to containers since it does not virtualize at the OS-level. This may not be a problem for volunteer computing platforms since there is some notion of trust between the researchers and the providers. However, in ChainFaaS, the providers do not necessarily trust the developers, and there could even exist a malicious entity in the network. In iExec [46], computing providers, which are called workers, can choose to run the program with a virtual machine, with a Docker container, or just running the script. Golem [48] uses a WebAssembly sandbox for this purpose. Both Golem and iExec are working on using Intel Software Guard Extensions (SGX) for isolation. Intel SGX offers hardware-level isolation, which is the most secure level. However, it is only supported on some modern Intel CPUs. Since this solution is hardware-specific and still experimental, it cannot be extended to all computers. Although beyond the scope of this paper, the security and privacy of providers and developers in an open distributed computing platform are important topics for future research.

C. BLOCKCHAIN PEER

The blockchain used in ChainFaaS is implemented using Hyperledger Fabric V1.4. When choosing the blockchain solution, one of the most important criteria was its energy consumption. As mentioned in Section I, one of the initial motivations for developing this platform was to decrease the carbon emission of the ICT ecosystem by increasing the usage of personal computers. However, blockchains that use proof of work as their consensus mechanism, such as Bitcoin, require computationally powerful computers to solve meaningless puzzles to get rewards. On the other hand, in Hyperledger Fabric, there is no need for solving such problems since the consensus algorithm is not based on proof of work. Moreover, Hyperledger Fabric has been designed explicitly for enterprises and takes into account their needs. In Hyperledger Fabric, everything is highly configurable and can be customized for different use cases. These features make it an excellent choice for ChainFaaS. The reason for selecting V1.4 is its stability and long term support. It is the first version with long term support, while V2.0 is still under development.

Hyperledger Fabric uses an execute-order-validate architecture for transactions. In this architecture, transactions are first executed and checked for correctness. Then, via a pluggable consensus algorithm, transactions are ordered. Finally, the transactions are validated against an application-specific endorsement policy and added to the ledger. Other blockchains that support smart contracts, such as Ethereum, Tendermint, and Quorom, use an order-execute architecture in which the transactions are validated and ordered first and only then executed by all peers. In order-execute based blockchains, smart contracts must be deterministic. As a result, these blockchains require smart contracts to be written in a domain-specific language to ensure that their operations adhere to this requirement. On the other hand, in Hyperledger Fabric smart contracts can be written in standard programming languages such as Go or Node.js. Also, since in order-execute blockchains all nodes execute the transactions, these blockchains face performance and scalability issues. Hyperledger Fabric's architecture enables applications to specify which peers and how many of them need to execute the transaction. As a result, only a subset of peers that are specified by the endorsement policy execute the transaction. This feature

enables parallel execution of transactions which increases the performance and scalability of the network [30].

The Hyperledger Fabric's network is managed by a collection of organizations that come together to form a blockchain. As a permissioned blockchain, Hyperledger Fabric needs a membership component to overlook the participants in the network. A trusted Membership Service Provider (MSP) is used for this purpose. Similar to other components in Hyperledger Fabric, the MSP is configurable by the network designer. In ChainFaaS, the default Fabric Certificate Authority (CA) is used as the MSP. Peers are important components of the blockchain network that are responsible for hosting the ledgers and smart contracts. They receive transactions, invoke corresponding smart contracts, and endorse the results. Each peer belongs to an organization and each organization can have many peers in the network. In Fabric, an ordering service is required which is responsible for collecting the endorsed transactions from the peers, ordering them and creating the blocks. The ordering service consists of one or more orderer nodes which reach consensus among each other based on a pluggable consensus algorithm. Currently, Hyperledger Fabric supports Raft, Kafka, and Solo consensus algorithms [30]. The implemented prototype of ChainFaaS uses one Solo orderer.

In the current implementation of ChainFaaS, all components of the Hyperledger Fabric network have been deployed on an instance with 8 VCPUs, 16GB RAM, and 160GB disk with Ubuntu 18.04. There is one Fabric Certificate Authority (CA), one Solo orderer, and two organizations, each with two peers. In the future version of ChainFaaS, there can be included more of these components, each running on different computers. Anyone with a public IP can run the blockchain peers. Each peer stores all the latest information about the ledgers and verifies the new transactions.

We are using chaincodes, which are Hyperledger Fabric's smart contracts, to implement the functionalities needed for ChainFaaS in the blockchain. There are two main chaincodes: monitoring and monetary. The monitoring chaincode is responsible for keeping track of every job that has been served in ChainFaaS. Fig. 6 shows the monitoring ledger with an example. Anything that is stored on Hyperledger Fabric blockchain is shown by a key-value pair. In the case of monitoring ledger, the key is the job ID, and the value is its developer, provider, function ID, time, cost, received, and payment-is-done information. The time shows the time the provider spends on running the function, received shows whether the end-user has received the result, and payment-done shows whether the payment has successfully taken place. In Fig. 6, monitoring ledger keeps track of all changes that happen to each job. Consider JOB7 as an example. Right now, the job has finished, but the end-user has not yet received the result. As soon as the end-user receives it, a new transaction is created that consists of the change of JOB7's received value from False to True. The world state database is part of the Hyperledger Fabric structure and stores the latest version

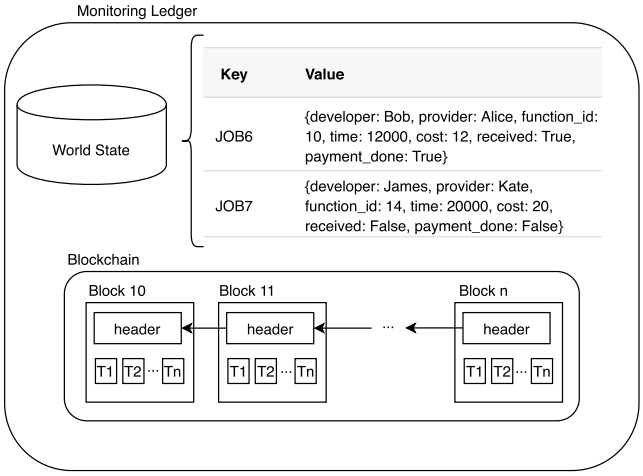

FIGURE 6. Details of the monitoring ledger. Blockchain stores changes to each job in terms of transactions. World state stores the latest version of the jobs.

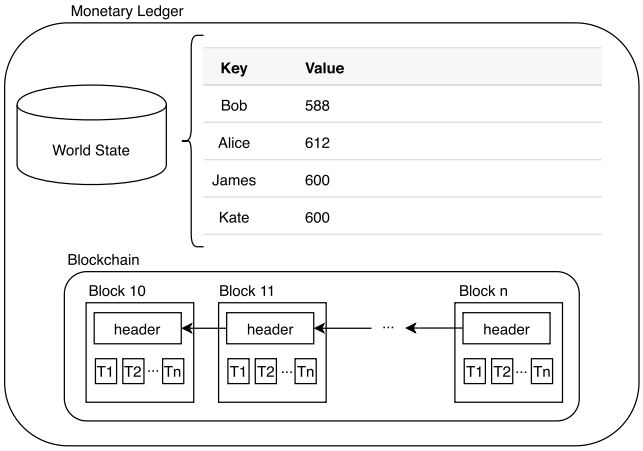

FIGURE 7. Details of the monetary ledger. Blockchain stores changes in each account in terms of transactions. World state stores the latest version of every account.

of every job. In other words, if anyone follows all the changes in the blocks from genesis to the last block, they will reach the value inside the world state. This implementation makes it easy to query the latest values very fast.

The monetary chaincode is responsible for keeping track of user account balances. The key in this ledger is the username of the user, and the value is how much they own in ChainFaaS. Fig. 7 shows the details of the monetary ledger. Similar to the monitoring ledger, the blockchain keeps track of changes that happen in accounts, and the world state stores the latest version of account balances. To better understand the functionality of the two ledgers, consider the examples shown in Fig. 6 and Fig. 7. Imagine that before JOB6 and JOB7, Bob, Alice, James, and Kate all had 600 units of money in their accounts. As soon as JOB6 is finished and received by the end-user, the cost (12 units) is deducted from Bob's account and added to Alice's account. Since the end-user has not yet received JOB7, the account balance of James and Kate has not changed.

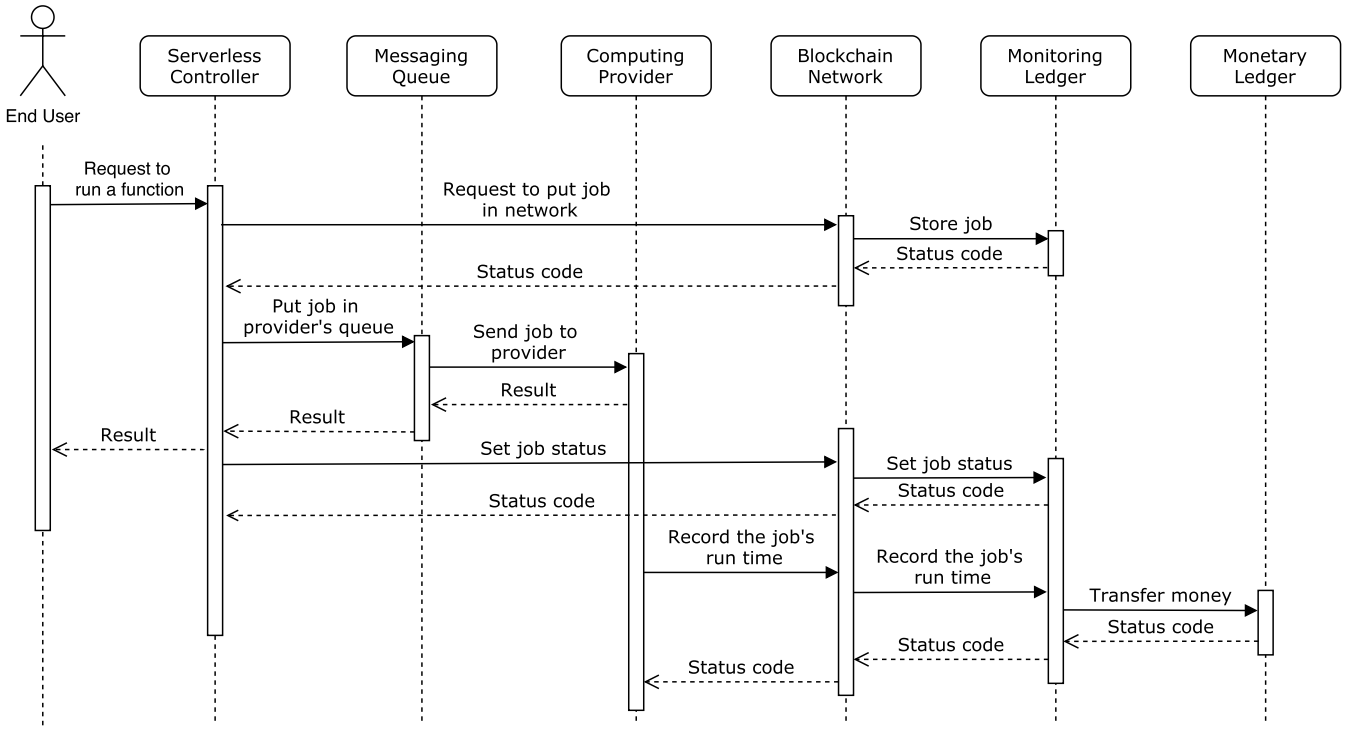

FIGURE 8. Sequence diagram showing the steps of processing a request in the prototype of ChainFaaS

V. EXPERIMENTAL EVALUATION

In this section, the functional and non-functional properties of the implemented prototype of ChainFaaS are evaluated. This prototype can be accessed through ChainFaaS's website.[3]

A. FUNCTIONAL

We have discussed how each unit in ChainFaaS is implemented. In this part, we will show how all the units work together to achieve the functional properties of the platform, described in Section III-C. The sequence diagram shown in Fig. 8 describes the process in which the prototype of ChainFaaS serves a request from an end-user. In the current implementation, the result storage is the serverless controller.

When the serverless controller receives a request to run a function, the scheduler first finds an active provider capable of serving the request. When a provider is found, the controller asks the blockchain network to store the job on the monitoring ledger. In this step, the job is created in the ledger and information such as the developer, the provider who is supposed to execute the job, and the function ID is set. The controller then puts the job in the provider's messaging queue. The provider fetches the job from the queue, runs the Docker container, and sends back the results to the controller to be accessed by the end-user. As mentioned earlier, the serverless controller is set as the result storage for the prototype implementation. In the future versions of ChainFaaS, the result storage will be chosen by the developer. After receiving the result, the controller confirms that the result has been received by setting the status of the job in the blockchain network. In the meantime, the provider also sets the time it took to run the function. The provider is only able to set the run time if the job has already been created by the controller in the blockchain network. Moreover, only the provider that has been assigned to the job can set the run time metrics. This ensures that others cannot tamper with the job's information in the blockchain network. When the monitoring chaincode receives both the confirmation from the controller and the run time from the provider, the service fee will be transferred to the provider's account from the developer's wallet.

B. NON-FUNCTIONAL

In this section, we evaluate the non-functional properties of the prototype, most importantly, its performance. Our goal is to shed some light on the system response time for various use cases. This baseline evaluation will help to enhance the platform's performance in future versions.

To evaluate the system, we have selected a sample function to run on the prototype of ChainFaaS. We have created a Docker container that gets the node's information, such as its operating system, number of CPUs, and uptime. Two sets of experiments have been conducted. One focuses on virtual machines running on clouds as providers, and another focuses on personal computers as providers. From now on, we will refer to the first and second experiments as the cloud deployment and the personal computers deployment, respectively.

[3]https://chainfaas.com

In the first experimental evaluation, the serverless controller, along with four computing power providers, run on virtual machines in the Compute Canada cloud. The blockchain network runs on a virtual machine on the SAVI testbed cloud. In the second set of experiments, the same serverless controller and blockchain network are used. However, the providers are two different personal computers. Table 1 summarizes the characteristics of the ChainFaaS components in the cloud deployment.

TABLE 1. Experimental evaluation setup in cloud deployment.

Component	Size	CPU	RAM	Disk
Serverless Controller	1	4 vCPU	15 GB	20 GB
Computing Network	4	2 vCPU	7.5 GB	20 GB
Blockchain Network	1	8 vCPU	16 GB	160 GB

The workload generated for the cloud deployment is shown in Fig. 9a. Based on a recent research by Shahrad *et al.* [56], 81% of applications running on Microsoft Azure Functions are invoked, on average, once per minute or less. This information shows that most applications are invoked infrequently, and the generated workload selection, i.e. the number of requests sent to the function, is reasonable for serverless platforms. During a one-hour period, exponentially distributed requests are sent to the serverless controller to invoke the sample function, and the performance metrics of the platform are recorded.

For all experiments, we use a client running on an instance in the Compute Canada cloud with 8 vCPUs, 30 GB of memory, and 186 GB of disk, with less than 10 milliseconds latency to the controller server. A Python 3.7 script sends exponentially distributed requests to a function on ChainFaaS. The results, the request time, and the response time are stored in CSV files, and later processed to extract the response time. On the serverless controller, for each request, two times are stored: the time that the provider has received the job and the time the provider has spent to finish it. Finally, the blockchain server records the billing time for each job.

The most important metric for the end-user is the end-to-end response time, T_{rsp}, shown as response time in Fig. 9b. During this time, the scheduler finds a provider for the job, the provider pulls the Docker container, runs it, sends back the result to the end-user, and the blockchain network keeps track of every change in the job's status. As expected, T_{rsp} increases when the number of requests to the function is increased.

As can be seen in Fig. 9b, the response time (T_{rsp}) and provisioning time (T_{prv}) follow the same pattern as the processing time (T_{prc}). This behaviour is expected since T_{prc} is included in both other metrics. The completion time (T_{cmp}) contains the pull and run time of the Docker container. Since all four compute providers are in the same network with the same computational capacity, and they run the same function, completion time should remain nearly constant. Fig. 9b confirms this assumption.

From the provider's point of view, it is crucial to know the billing time (T_{bil}). Fig. 9c shows the average value of T_{bil} in a given minute in the cloud deployment. The billing time usually falls between 5-10 seconds and remains below 20 seconds.

Finally, for developers, it is crucial to know how fast they can deploy a new function. Initially, a developer who wants to submit a function to ChainFaaS needs to register on the platform. During the registration process, an account is created for the user in the blockchain. This process takes about 5 seconds, which is a one-time-only wait. After that, the developer can create and submit a function in a matter of a few milliseconds by providing the controller with the Docker registry path.

In the second set of experiments, we focus on personal computers as providers. In these experiments, the serverless controller and the blockchain network are the same as the cloud deployment, but the providers are personal computers instead of virtual machines. Table 2 shows the hardware specifications of the PCs used in the personal computers deployment.

TABLE 2. Hardware specifications of providers in personal computers deployment.

Component	CPU	RAM	Disk
Provider 1	Core i7-6700HQ	12GB	1TB
Provider 2	Core i7-6700HQ	16GB	1TB

The results of this experiment are shown in Fig. 10. It is interesting to see how the two experiments differ from each other. The completion time is almost doubled in the second set of experiments. This is mainly due to networking delays. In the cloud deployment, both the controller and provider were running on the same network (Compute Canada cloud), which makes communications much faster. Moreover, virtual machines running on cloud generally have more stable networks compared to personal computers. The provisioning time and response time are also influenced by the networking delay. The average response time in the cloud deployment is 10.2 seconds. This value is 16.8 seconds in the personal computers deployment.

Since the processing time is not influenced by the network delays, it is expected to be similar in both experiments when running on the same request rate. This expectation is shown to be true in Fig. 9b and Fig. 10b. The average processing time in personal computer deployment is 2.8 seconds. In the cloud deployment, during the time the request rate is less than five requests per minute, the processing time is around the same value (2.5 seconds). Similar to the cloud deployment, the billing time, which is shown in Fig. 10c, for the personal computer deployment, is still between 5-10 seconds.

VI. DISCUSSIONS AND THREATS TO VALIDITY

In this section, we discuss potential threats to the validity of the design implementation, and evaluation of ChainFaaS,

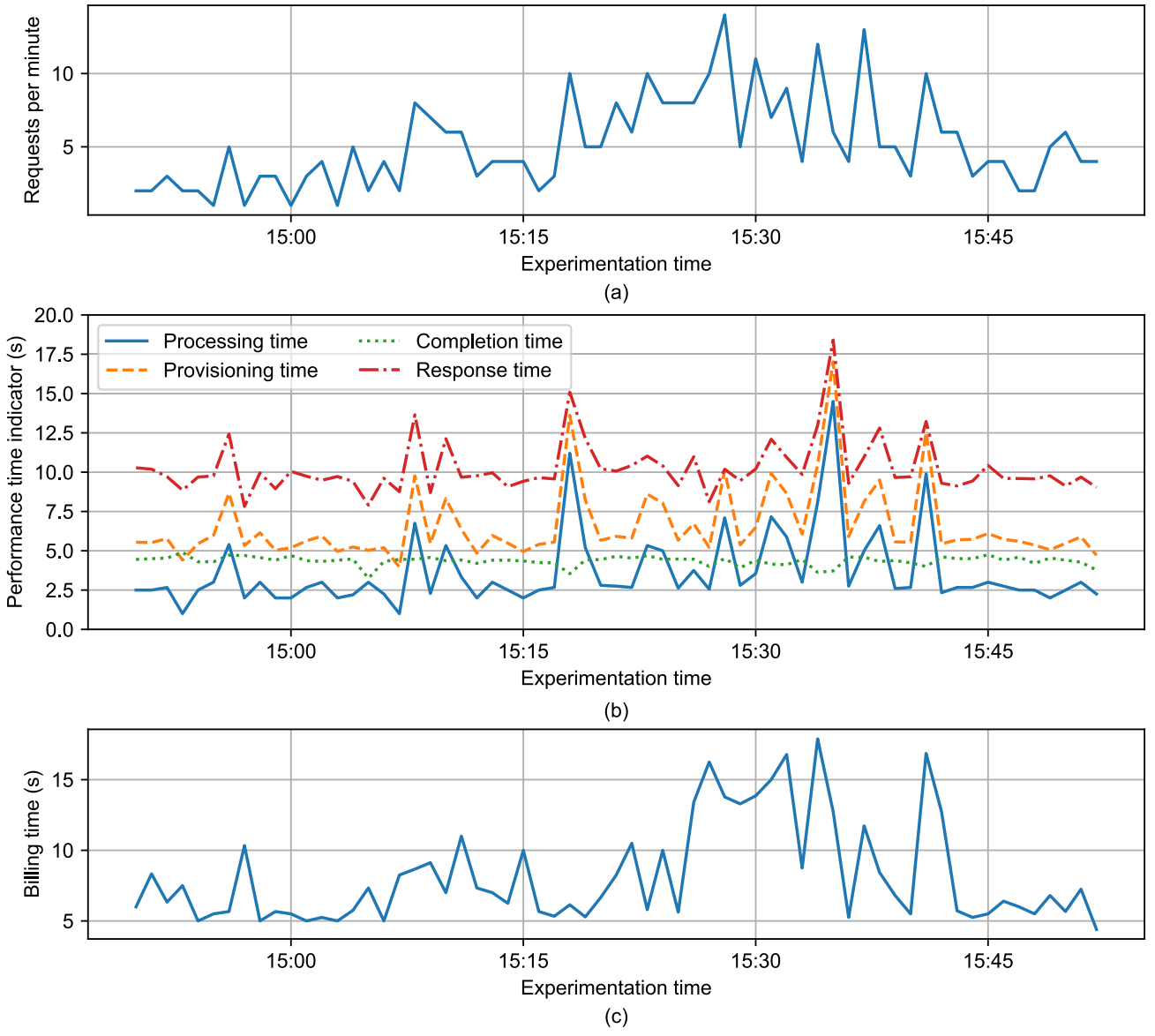

FIGURE 9. The trend of various performance metrics of ChainFaaS in the cloud deployment. (a) Workload generated to evaluate ChainFaaS over time. (b) Average processing time, completion time, provisioning time, and response time; averaged over a minute. (c) Average billing time in a given minute.

as well as possible improvements of the platform. Although the implementation and evaluation of ChainFaaS confirm that the proposed platform is feasible, there is still much potential for its improvement.

From the computing providers' and the blockchain peers' point of view, their income from this platform should be worth their time and effort. They also need to consider the increased electricity consumption of their computers resulting from their participation in the network. Moreover, to motivate the developers to switch from public cloud providers to ChainFaaS, the platform should be reliable and cost-efficient. While the initial design of ChainFaaS indicates a sustainable income for the providers and the peers and a low-cost for the developers, a more in-depth cost analysis is needed to confirm this. It is necessary to take into account the cost efficiency of the platform for the developers as well as any possible costs for the providers and the blockchain peers.

Based on qualitative analysis, we have concluded that the use of ChainFaaS can have a positive impact on the environment by reusing the available computational capacity of personal computers. Although this statement appears reasonable, a quantitative analysis of power consumption is needed. This analysis should compare the consumption of public serverless platforms with the prototype of ChainFaaS using different scenarios and report the consumption values. However, this evaluation would be difficult to conduct since we do not have access to the underlying infrastructure of public serverless platforms.

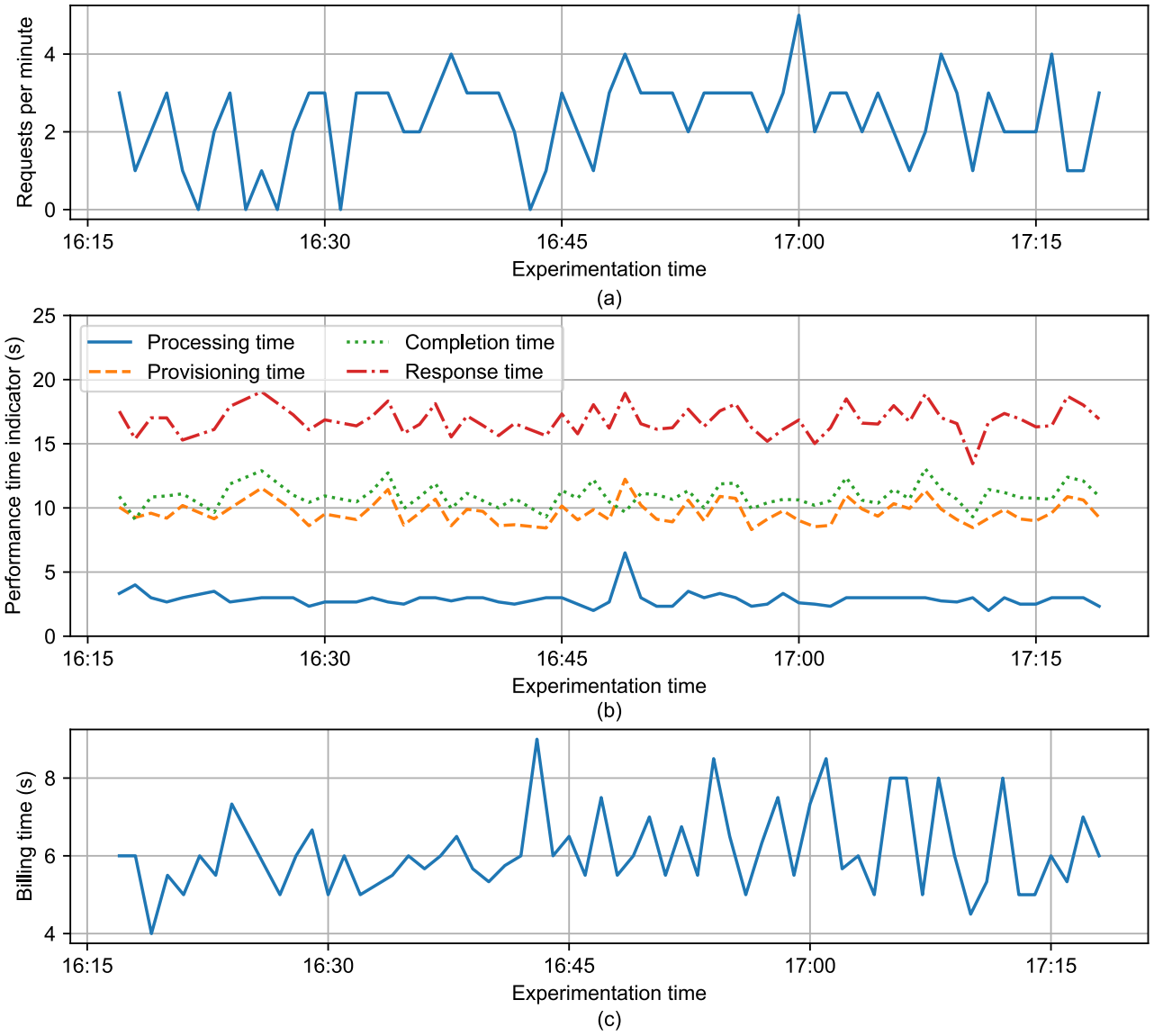

FIGURE 10. The trend of various performance metrics of ChainFaaS in the personal computers deployment. (a) Workload generated to evaluate ChainFaaS over time. (b) Average processing time, completion time, provisioning time, and response time; averaged over a minute. (c) Average billing time in a given minute.

In the current prototype of ChainFaaS, the scheduler randomly selects one of the available providers with enough CPU and RAM for the request. The scheduler can be improved to take into account other relevant factors. For instance, it can consider the reputation of the providers: those with a higher successful job completion score may get a higher priority in scheduling compared to others. Moreover, instead of having the providers delete the Docker images that they have run, they could cache the images of recent requests they served. The scheduler can then take into account the images that each provider has stored when distributing new tasks. This way, the impact of cold starts will be minimized.

The compute provider's agent has been deployed as a Docker container to prevent the code from tampering with the provider's programs and files. However, there are still some concerns about the security of Docker containers [57]. An interesting research direction on open platforms could be to increase the security for providers while maintaining the performance and usability of the platform.

Any blockchain-based platform faces performance barriers, and ChainFaaS is not an exception. Hyperledger Fabric performs faster than most blockchains, especially those with a proof-of-work consensus algorithm. Nevertheless, as can be seen from the experiments, sometimes it can take up to 20 seconds to complete a task. The blockchain network could be optimized to enhance the overall performance of the platform.

Current implementation of ChainFaaS lacks a policy management component that defines different policies for the system such as fault tolerance, update and change, as well as workload aggregation policies. Since each provider node is an unreliable personal computer, the system's reliability

per node is low. On the other hand, sine a large number of providers participate in the network, the system achieves reliability by number. A fault tolerance policy is needed to specify what the system should do in case a provider node is unable to respond to a request. Also, an update policy is needed to specify the update and upgrade mechanisms of the system. This policy should include how the nodes are notified of the updates and what they should do if an update is received in the middle of a task. Finally, a workload aggregation policy would be helpful. With such policy, the network can accept large tasks and divide them into smaller subtasks that can run on individual personal computers. In the end, using the aggregation policy, it could merge the results from different nodes.

In the current experiments, the number of providers in the network, and the number of blockchain peers are fixed. Changing the scale of the system may influence the performance. A possible future research direction is to identify the bottlenecks of the system by changing the parameters and conducting different experiments. Knowing the bottlenecks would help scale the system in the future versions.

VII. CONCLUSION

Personal computers around the world are highly underutilized, and their computational power is being wasted every day. We have conducted a survey to gather more details and quantify this information. The results show that the typical CPU utilization of a personal computer is only 24.5% and, on average, a personal computer is only used 4.5 hours per day. Motivated by this, we have designed, implemented and evaluated an open blockchain-based serverless platform called ChainFaaS that uses the untapped computational power of personal computers. This platform can be used by developers to run tasks in a scalable environment with minimal infrastructure management overhead and a reasonable price. Any individual can rent out their extra computational power on ChainFaaS to make a profit. As proof of concept, we have implemented and evaluated a prototype of the proposed platform which is publicly available.[4] Also, the source code, documentations, and user guides are available on GitHub.[5] The prototype uses Hyperledger Fabric as the blockchain solution which enables decentralized and transparent management of the platform. Moreover, to ensure the security of computing providers, this platform runs jobs in isolated environments using containerization techniques. The evaluation process indicates the feasibility of the idea with satisfactory performance.

APPENDIX
PERSONAL COMPUTER SURVEY

The ChainFaaS platform is based on the assumption that personal computers are highly underutilized. However, there are no recent studies to verify this premise. In 2005,

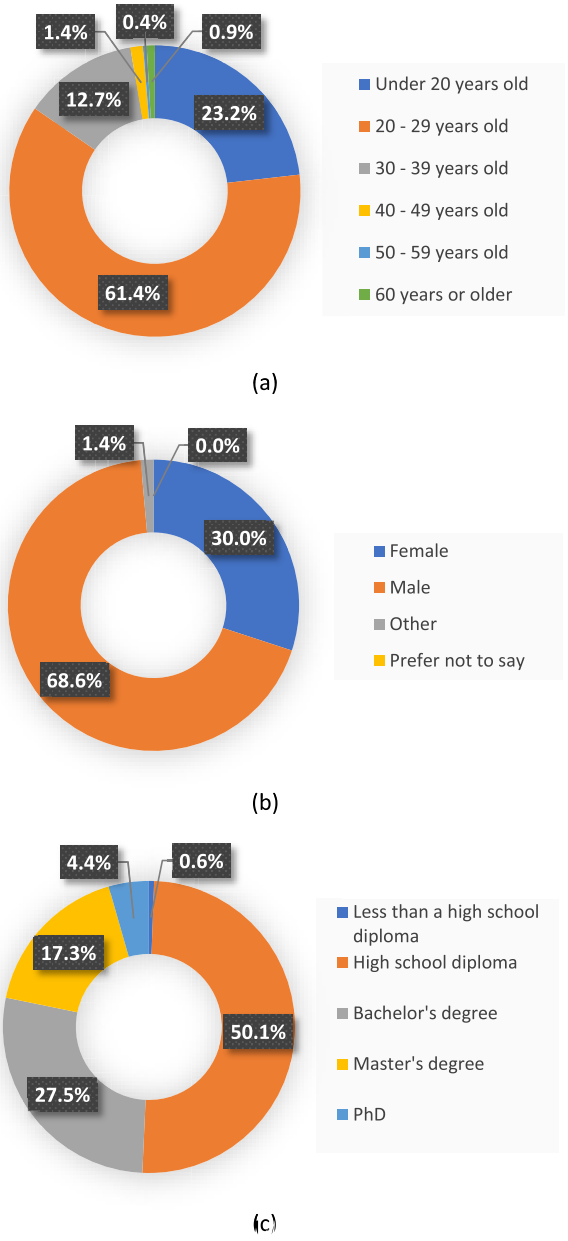

FIGURE 11. Demographic information of the participants in the personal computer survey. (a) Age (b) Gender (c) Highest level of education

Domingues *et al.* [58] studied the resource usage of Windows 10 machines from classroom laboratories. The results show an average CPU idleness of 97.9%. Although this study shows much wasted computational capacity, it focuses only on laboratory computers, and it was conducted about 15 years ago. Therefore, to provide stronger support of the idleness assumption, we conducted a survey and asked participants to report their computer's CPU utilization and unused memory when running their regular programs.

In this survey, we first gathered some demographic information about the participant, such as their age, gender, and level of education. The rest of the survey questions were aimed to find the amount of computational power that is not

[4]https://chainfaas.com
[5]https://github.com/pacslab/ChainFaaS

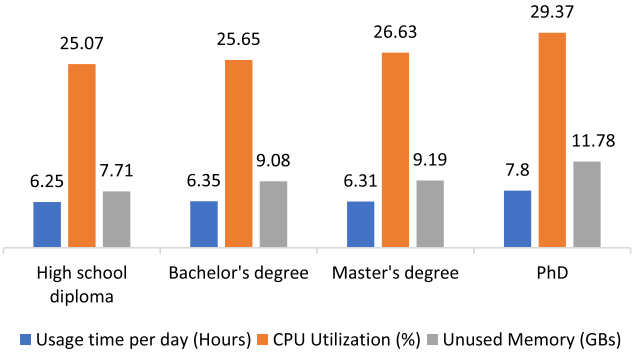

FIGURE 12. Average main computer's usage time per day, CPU utilization, and unused memory for each of the education groups.

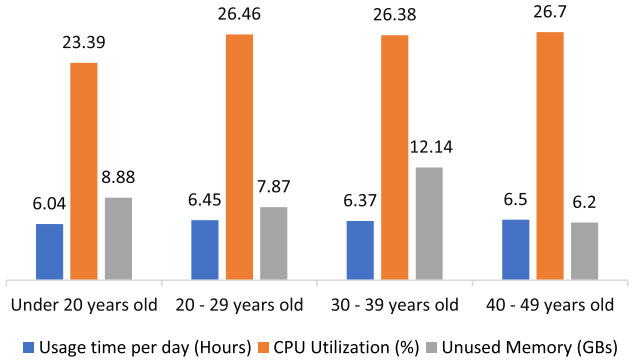

FIGURE 13. Average main computer's usage time per day, CPU utilization, and unused memory for each of the age groups.

being used. The participants first state the number of personal computers that they own. Then they answer the following questions for each of their computers:

- On average, how many hours per day do you work with your personal computer?
- What is the primary operating system that you use on your computer?
- What is your computer's average CPU utilization when running regular programs?
- What is the amount of unused memory, in gigabytes, on your computer when running regular programs?

About 700 people participated in this survey, mostly university students of computer science or computer engineering. Fig. 7 shows the demographic information of the participants. Most of them are 20 - 29 years old, male, with a high school diploma. In total, we gathered information about the utilization of about 1150 computers. The gathered data in this survey is available publicly on GitHub.[6]

To get a better understanding of the gathered data, the average usage time per day, CPU utilization, and unused memory of the participants' main computer for the education groups and age groups are shown in Fig. 12 and Fig. 13, respectively. Since there were less than 10 participants with an education level of less than a high school diploma, it was not possible to calculate a reliable average value for this group. The situation

[6]https://github.com/pacslab/PC-Survey

TABLE 3. Average usage time, CPU utilization, and unused memory for the main computers and all the computers in the survey.

	Time (Hours)	CPU (%)	Memory (GBs)
Main Computers	6.34	25.72	8.6
All Computers	4.53	24.54	7.91

is similar for the age groups of 50-59 years old, and 60 years and older.

It is also interesting to see the popularity of different operating systems among the participants. About 73.8% of the computers are running on Windows, which shows the popularity of this operating system. 16.6% run on macOS, 8.4% on Linux, and the remaining 1.2% run on other operating systems.

Finally, Table 3 shows the average usage time per day, CPU utilization, and unused memory for the main computers and all the computers in the survey. It can be seen that the average CPU utilization for all the personal computers in this survey is 24.54%. Since most participants are computer engineering and computer science students, the average CPU utilization for a more general audience is expected to be even less than this number. This result confirms our initial assumption that personal computers are highly underutilized.

REFERENCES

[1] R. Danilak, "Why energy is a big and rapidly growing problem for data centers," *Forbes*, vol. 15, pp. 12–17, 2017.
[2] N. Jones, "How to stop data centres from gobbling up the world's electricity," *Nature*, vol. 561, no. 7722, pp. 163–167, 2018.
[3] L. Belkhir and A. Elmeligi, "Assessing ICT global emissions footprint: Trends to 2040 & recommendations," *J. Cleaner Prod.*, vol. 177, pp. 448–463, Mar. 2018.
[4] IDC. (2019). *Personal Computing Device Shipments Forecast to Continue Their Slow Decline with a Five-Year Compound Annual Growth Rate of -1.2%, According to IDC*. Accessed: May 3, 2020. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS44908319
[5] C. Stoll, L. Klaaßen, and U. Gallersdörfer, "The carbon footprint of bitcoin," *Joule*, vol. 3, no. 7, pp. 1647–1661, Jul. 2019.
[6] P. Mell, "The nist definition of cloud computing," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. 800-145, 2011.
[7] A. Fox, "Above the clouds: A Berkeley view of cloud computing," Dept. Electr. Eng. Comput. Sci., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS, p. 2009, 2009, vol. 28, no. 13.
[8] I. Baldini, "Serverless computing: Current trends and open problems," in *Research Advances in Cloud Computing*. Singapore: Springer, 2017, pp. 1–20.
[9] (2019). Amazon Web Services. *Aws Lambda*. Accessed: Jan. 23, 2020. [Online]. Available: https://aws.amazon.com/lambda/
[10] Microsoft. (2020). *Azure Functions*. Accessed: Feb. 21, 2020. [Online]. Available: https://azure.microsoft.com/en-us/services/functions/
[11] Google. *Google Cloud Functions*. Accessed: Feb. 21, 2020. [Online]. Available: https://cloud.google.com/functions
[12] *IBM Cloud Functions*. Accessed: Feb. 23, 2020. [Online]. Available: https://cloud.ibm.com/functions/
[13] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. Accessed: Jul. 17, 2020. [Online]. Available: https://bitcoin.org/bitcoin.pdf
[14] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, Apr. 2014.
[15] D. Schwartz, "The ripple protocol consensus algorithm," *Ripple Labs Inc White Paper*, vol. 5, no. 8, p. 151, 2014.
[16] C. Insights. (2019). *Banking Is Only The Beginning: 42 Big Industries Blockchain Could Transform*. Accessed: Feb. 24, 2020. [Online]. Available: https://www.cbinsights.com/research/industries-disrupted-blockchain//

[17] T.-T. Kuo, H.-E. Kim, and L. Ohno-Machado, "Blockchain distributed ledger technologies for biomedical and health care applications," *J. Amer. Med. Inform. Assoc.*, vol. 24, no. 6, pp. 1211–1220, Nov. 2017.

[18] J. Zhang, N. Xue, and X. Huang, "A secure system for pervasive social network-based healthcare," *IEEE Access*, vol. 4, pp. 9239–9250, 2016.

[19] Q. Xia, E. B. Sifah, K. O. Asamoah, J. Gao, X. Du, and M. Guizani, "MeDShare: Trust-less medical data sharing among cloud service providers via blockchain," *IEEE Access*, vol. 5, pp. 14757–14767, 2017.

[20] M. F. Zia, M. Benbouzid, E. Elbouchikhi, S. M. Muyeen, K. Techato, and J. M. Guerrero, "Microgrid transactive energy: Review, architectures, distributed ledger technologies, and market analysis," *IEEE Access*, vol. 8, pp. 19410–19432, 2020.

[21] T. M. Fernández-Caramés and P. Fraga-Lamas, "A review on the use of blockchain for the Internet of Things," *IEEE Access*, vol. 6, pp. 32979–33001, 2018.

[22] P. K. Sharma, M.-Y. Chen, and J. H. Park, "A software defined fog node based distributed blockchain cloud architecture for IoT," *IEEE Access*, vol. 6, pp. 115–124, 2018.

[23] K. Korpela, J. Hallikas, and T. Dahlberg, "Digital supply chain transformation toward blockchain integration," in *Proc. 50th Hawaii Int. Conf. Syst. Sci.*, Jan. 2017, p. 15.

[24] K. Toyoda, P. T. Mathiopoulos, I. Sasase, and T. Ohtsuki, "A novel blockchain-based product ownership management system (POMS) for anti-counterfeits in the post supply chain," *IEEE Access*, vol. 5, pp. 17465–17477, 2017.

[25] J. Park and J. Park, "Blockchain security in cloud computing: Use cases, challenges, and solutions," *Symmetry*, vol. 9, no. 8, p. 164, Aug. 2017.

[26] F. M. Benčić and I. P. Žarko, "Distributed ledger technology: Blockchain compared to directed acyclic graph," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Oct. 2018, pp. 1569–1570.

[27] N. Szabo. (1994). *Smart Contracts*. Accessed: Feb. 24, 2020. [Online]. Available: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literat%ure/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html

[28] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[29] E. Androulaki, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proc. 13th EuroSys Conf.*, Apr. 2018, pp. 1–15.

[30] Hyperledger. (2019). *Hyperledger Fabric Documentation*. Accessed: Feb. 24, 2020. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/whatis.html

[31] G. Woltman and S. Kurowski. (1996). *The Great Internet Mersenne Prime Search*. Accessed: May 12, 2020. [Online]. Available: http://www.mersenne.org

[32] E. Ady. (1997). *Distributed*. Accessed May 12, 2020. [Online]. Available: https://www.distributed.net/

[33] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI home: An experiment in public-resource computing," *Commun. ACM*, vol. 45, no. 11, pp. 56–61, Nov. 2002.

[34] K. Watanabe, M. Fukushi, and M. Kameyama, "Adaptive group-based job scheduling for high performance and reliable volunteer computing," *J. Inf. Process.*, vol. 19, pp. 39–51, 2011.

[35] P. Dharmapala, L. Koneshvaran, D. Sivasooriyathevan, I. Ismail, and D. Kasthurirathna, "Peer-to-peer distributed computing framework," in *Proc. 6th Nat. Conf. Technol. Manage. (NCTM)*, Jan. 2017, pp. 126–131.

[36] D. P. Anderson, "BOINC: A system for public-resource computing and storage," in *Proc. 5th IEEE/ACM Int. Workshop Grid Comput.*, 2004, pp. 4–10.

[37] D. P. Anderson, "Boinc: A platform for volunteer computing," *J. Grid Comput.*, pp. 1–24, 2019.

[38] G. Fedak, C. Germain, V. Neri, and F. Cappello, "XtremWeb: A generic global computing system," in *Proc. 1st IEEE/ACM Int. Symp. Cluster Comput. Grid*, 2001, pp. 582–587.

[39] (2020). *The Mithral (COSM)*. Accessed: Jun. 23, 2020. [Online]. Available: http://www.mithral.com

[40] The Folding home Consortium. (2020). *Folding Home*. Accessed: Jun. 23, 2020. [Online]. Available: https://foldingathome.org/

[41] A. L. Beberg, D. L. Ensign, G. Jayachandran, S. Khaliq, and V. S. Pande, "Folding home: Lessons from eight years of volunteer distributed computing," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, May 2009, pp. 1–8.

[42] S. M. Larson, C. D. Snow, M. Shirts, and V. S. Pande, "Folding home and genome home: Using distributed computing to tackle previously intractable problems in computational biology," 2009, *arXiv:0901.0866*. [Online]. Available: http://arxiv.org/abs/0901.0866

[43] FoldingCoin. (2018).*Foldingcoin Whitepaper*. Accessed: Jun. 23, 2020. [Online]. Available: https://foldingcoin.net/index.php/whitepaper

[44] C. Team. (2019). *Curecoin Whitepaper*. Accessed: Jun. 23, 2020. [Online]. Available: https://curecoin.net/white-paper/

[45] GridCoin. (2018). *Gridcoin Whitepaper*. Accessed: Jun. 23, 2020 [Online]. Available: https://gridcoin.us/assets/img/whitepaper.pdf

[46] (2018). *Iexec: Blockchain-Based Decentralized Cloud Computing V3.0*. Accessed: Mar. 24, 2020. [Online]. Available: https://iex.ec/wp-content/uploads/pdf/iExec-WPv3.0-English.pdf

[47] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proc. 26th Symp. Oper. Syst. Princ.*, Oct. 2017, pp. 153–167.

[48] (2016). *The Golem Project: Crowdfunding Whitepaper*. Accessed: Mar. 24, 2020. [Online]. Available: https://golem.network/crowdfunding/Golemwhitepaper.pdf

[49] S. P. Ltd. (2020). *Sonm: Decentralized fog Computing Platform*. Accessed: Mar. 24, 2020. [Online]. Available: https://sonm.com/

[50] K. Doka, T. Bakogiannis, I. Mytilinis, and G. Goumas, "Cloudagora: Democratizing the cloud," in *Proc. Int. Conf. Blockchain*. Cham, Switzerland: Springer, 2019, pp. 142–156.

[51] T. Bakogiannis, I. Mytilinis, K. Doka, and G. Goumas, "Leveraging blockchain technology to break the cloud computing market monopoly," *Computers*, vol. 9, no. 1, p. 9, Feb. 2020.

[52] R. B. Uriarte and R. De Nicola, "Blockchain-based decentralized cloud/fog solutions: Challenges, opportunities, and standards," *IEEE Commun. Standards Mag.*, vol. 2, no. 3, pp. 22–28, Sep. 2018.

[53] R. Yang, F. R. Yu, P. Si, Z. Yang, and Y. Zhang, "Integrated blockchain and edge computing systems: A survey, some research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1508–1532, 2nd Quart., 2019.

[54] M. Westerlund and N. Kratzke, "Towards distributed clouds: A review about the evolution of centralized cloud computing, distributed ledger technologies, and a foresight on unifying opportunities and security implications," in *Proc. Int. Conf. High Perform. Comput. Simul. (HPCS)*, Jul. 2018, pp. 655–663.

[55] Django Software Foundation. (2020). *Django*. Accessed: Feb. 13, 2020. [Online]. Available: https://www.djangoproject.com/

[56] M. Shahrad, R. Fonseca, Ì. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," 2020, *arXiv:2003.03423*. [Online]. Available: http://arxiv.org/abs/2003.03423

[57] T. Combe, A. Martin, and R. Di Pietro, "To docker or not to docker: A security perspective," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 54–62, Sep. 2016.

[58] P. Domingues, P. Marques, and L. Silva, "Resource usage of windows computer laboratories," in *Proc. Int. Conf. Parallel Process. Workshops*, 2005, pp. 469–476.

SARA GHAEMI (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering from the Amirkabir University of Technology, Tehran, Iran, in 2018. She is currently pursuing the M.S. degree in software engineering and intelligent systems with the University of Alberta, Edmonton, AB, Canada. She is a Research Assistant with the University of Alberta and a Visiting Research Assistant with the Performant and Available Computing Systems (PACS) Laboratory, York University, Toronto, ON, Canada. Her research interests include cloud computing, distributed ledger technologies, and distributed systems.

HAMZEH KHAZAEI (Member, IEEE) received the Ph.D. degree in computer science from the University of Manitoba, where he extended queuing theory and stochastic processes to accurately model the performance and availability of cloud computing systems. He was an Assistant Professor with the University of Alberta, a Research Associate with the University of Toronto, and a Research Scientist with IBM. He is currently an Assistant Professor with the Department of Electrical Engineering and Computer Science, York University. His research interests include performance modeling, cloud computing, and engineering distributed systems.

PETR MUSILEK (Senior Member, IEEE) received the Ing degree (Hons.) in electrical engineering and the Ph.D. degree in cybernetics from Military Academy, Brno, Czech Republic, in 1991 and 1995, respectively. In 1995, he was appointed the Head of the Computer Applications Group, Institute of Informatics, Military Medical Academy, Hradec Králové, Czech Republic. From 1997 to 1999, he was a NATO Science Fellow with the Intelligent Systems Research Laboratory, University of Saskatchewan, Canada. In 1999, he joined the Department of Electrical and Computer Engineering, University of Alberta, Canada, where he is currently a Full Professor. He was the Director of Computer Engineering Program, from 2016 to 2017. He serves as an Associate Chair (Research and Planning) with the ECE Department. His research interests include artificial intelligence and distributed, and renewable energy systems.