A Full Mirror Computation Model for Edge-Cloud Computing

Yuanda Wang*
yuandawang@ufl.edu
Department of Computer and Information Science and Engineering
Gainesville, Florida, USA

Ye Xia
Department of Computer and Information Science and Engineering
Gainesville, Florida, USA
yx1@ufl.edu

Youlin Zhang
Department of Computer and Information Science and Engineering
Gainesville, Florida, USA
ylzh10@ufl.edu

Dimitrios Melissourgos
Department of Computer and Information Science and Engineering
Gainesville, Florida, USA
dmelissourgos@ufl.edu

Olufemi O Odegbile
Department of Computer and Information Science and Engineering
Gainesville, Florida, USA
oodegbile@ufl.edu

Shigang Chen
Department of Computer and Information Science and Engineering
Gainesville, Florida, USA
sgchen@cise.ufl.edu

ABSTRACT

Edge computing has been gaining momentum lately as a means to complement cloud computing for shorter response time, better user experience, and improved data security. Traditional approaches of edge-cloud computing take two major forms: One is to offload the computation from an edge device to the cloud so as to take advantage of the virtually unlimited resources in the cloud and reduce the computation time. The other is to move selected computation to the edge devices where data are produced, actions are performed and users are located. However, in practice, it is often difficult to split the computation tasks of an application and decide which tasks should be performed in the cloud and which at the edge. The reason is that, for the same computation, it may sometimes be beneficial to execute it in the cloud while other times at the edge, depending on run-time conditions such as the data size, the type of computation, and the communication delay, which all varies from time to time. This paper proposes a new edge-cloud computing model, called the full mirror model, which provides a generic method to circumvent the problem of dynamic decisions on the execution location. With a two-thread implementation mechanism, the new model is able to achieve an execution completion time approximately equal to the smaller one between cloud execution and edge execution, regardless of what run-time conditions are. We test the new model by modifying an existing program for network traffic analysis so that it runs at both the edge and the cloud in a coordinated fashion. The experimental results demonstrate that the proposed model outperforms edge-alone computing and cloud-alone computing in reducing the execution time.

CCS CONCEPTS

• **Networks** → **Network architectures**.

KEYWORDS

edge-cloud computing, full mirror computation model, network activity viewer/analyzer.

ACM Reference Format:
Yuanda Wang, Ye Xia, Youlin Zhang, Dimitrios Melissourgos, Olufemi O Odegbile, and Shigang Chen. 2021. A Full Mirror Computation Model for Edge-Cloud Computing. In *2021 Thirteenth International Conference on Contemporary Computing (IC3-2021) (IC3 '21), August 5–7, 2021, Noida, India.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3474124.3474142

IC3 '21, August 5–7, 2021, Noida, India

ACM ISBN 978-1-4503-8920-4/21/08…$15.00
https://doi.org/10.1145/3474124.3474142

1 INTRODUCTION

Edge cloud computing is a paradigm, which can distribute the computing tasks to edge devices to utilize the increased computation resources at the network edge. Edge cloud computing also offloads computation tasks to the cloud server for higher processing power [7]. In edge computing, computation is offloaded to the edge devices, which are near to Internet users, so that the communication delay is short. As shown in Fig. 1, edge computing has played a remarkable role in many domains in our daily life, including vehicles, education, market, healthcare, etc. [3]. However, because the computing capability of edge devices is often limited, some computing-intensive tasks cannot be executed on edge devices. As such, we need to utilize the powerful computing power of cloud servers; but it takes longer time to transmit data to the cloud servers.

Although edge computing and cloud computing can collaborate to execute an application, the specific location needs be determined to execute each computation task. Considering edge computing and cloud computing each have their own advantages and limitations, it is often hard to determine where to place a computing task.

Recent years have witnessed great progress to improve the structure and efficiency of edge and cloud computing. In cloud computing, cloud server provides the computation services. As the cloud computing technology matures, many business service platforms (e.g., Amazon Web Service, GoGrid, Flexiscale, Mosso) have appeared based on cloud systems [19]. In edge computing, one needs to determine an edge device to execute computation tasks which has enough computing capacity. A resource allocation scheme assigns the available resources to the appropriate computational tasks. Al-Shuwaili et al. proposed an energy-efficient resource-allocation scheme for augmented reality applications [4], which can reduce

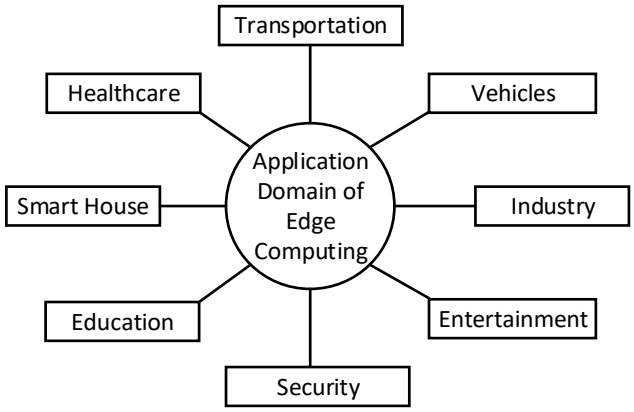


Figure 1: The Applications of Edge Computing

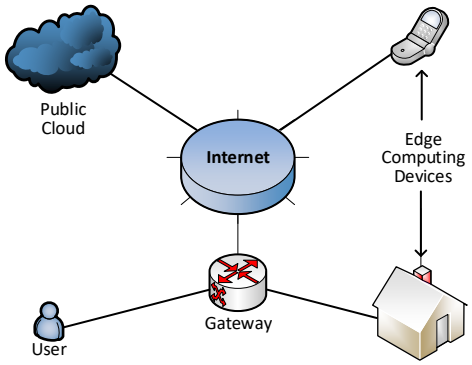


Figure 2: The Framework of Edge-Cloud Computing

mobile energy consumption compared with conventional offloading methods. Such a scheme increases the performance of edge computing. However, either a cloud or edge device is selected to execute a specific program [24]. The collaboration between the cloud and edge devices is missing. This leads to three problems. First, it takes extra time to evaluate the performance, delay and resource efficiency over the edge and cloud devices. Second, such a scheme can not fully utilize the computing resources on both ends [16]. Lastly, determining whether a specific application should be executed at the edge or in the cloud is difficult since the response time can depend on both the processing capacity of devices and data transmission delay over the Internet [11].

To solve these problems, in this paper we propose a *full mirror computation model*, which adopts a *joint execution mechanism* to execute an application program. The full mirror computation model (for brevity, the full model) represents a new computing paradigm called edge-cloud computing. Think about the high-level structure of an edge-cloud computing system. Fig. 2 is the framework of edge-cloud computing. In general, an edge-cloud computing system consists of four components: users, gateways, backend cloud platform (e.g, servers and datacenters) and edge devices (e.g., smart mobiles and smart houses) [29]. In edge computing system, the gateway is usually under the control of users. The users communicate with cloud servers and edge devices via gateways, through which they can request computational tasks. Under the edge-cloud computing model, computational tasks are assigned to both the edge devices and a centralized server (cloud server), and both the cloud server and edge devices are responsible for computation. Meanwhile, all the computing devices will communicate with each other for resource allocation and to share execution results.

Our full model presents a view of edge computing from a new angle: An application not only has its own integrated environment, but also has a scalable virtual computing environment [17]. To fully utilize the computation resources on both ends, in the full model, an application program is divided into multiple independent components, which are executed on both the cloud server and edge devices. Each component will be executed simultaneously at the edge and the cloud, and is synchronized through Internet communications. As a result, computation resources on both the edge devices and the cloud server will be consumed and the system will always get the execution results from the device that finishes the first.

In general, the edge devices' processing capability is limited, but the communication delay is very short. On the opposite, the remote cloud server is much more powerful, but the communication delay may be longer. Our full model adopts a distributed computing structure and integrates the computation resources from both the cloud server and the edge devices, which broadens the choice of computation resources. Moreover, because of the parallel execution of the application on both the edge and cloud devices, we always get the result from the device that finishes earlier [23].

The contributions of our paper are multifold:

- We design a full mirror computation model, which does not require any pre-computation to determine the execution location of an application in advance.
- We integrate the computation resources from both the edge devices and cloud server. By parallel execution of the application on both the edge and cloud devices, the execution time is guaranteed to be the shorter one.
- Rather than generating the output of an application program from a designated device that is determined in advance, our system executes an application on both ends to fully utilize the computation resources.

The rest of this paper is organized as follows: Section 2 presents the system model and problem statement. Section 3 presents the design of full model. In section 4, we introduce joint execution mechanism of our full model in detail. We evaluate and compare the performance of our full mirror computation model with the conventional edge computing model and cloud computing model in Section 5. Section 6 discusses additional related works. Section 7 draws conclusions for this paper.

2 SYSTEM MODEL AND PROBLEM STATEMENT

The system model consists of edge devices and the cloud (e.g., a data center), which together execute an application for better performance. In this paper, out focus is on the response time of computation tasks issued by a user. We may either execute the

application on an edge device. When facing computational intensive tasks, the edge device may offload these tasks to the cloud in order to reduce the completion time. We may also execute the application in the cloud. When facing user-interactive or sensor-interactive low computation tasks, the cloud may push these tasks to the edge device for faster response.

The problem is that it is often difficult to determine which way of executing a task will save time, in the cloud or by the edge device. The answer depends on a large number of run-time conditions such as data size, type of computation, communication delay, resource availability at the edge, resource at the cloud, other concurrent computations, etc. This paper attempts to solve the problem through a new edge-cloud computing model that circumvent the need for pre-determination of execution location. It is able to work under any run-time conditions.

3 DESIGN OF THE FULL MIRROR MODEL

The full model we propose in this paper is designed under a new computing paradigm, edge-cloud computing, based on the premise that we have enough computation resources on both edge devices and cloud server. The basic idea of our full model is that the same application program will always be executed on the edge devices and the cloud server simultaneously. Whenever either an edge device or a cloud server completes its assigned task, it will send out an execution cancellation signal to the other device. When receiving a cancellation signal, the other device stops the same task immediately. The user gets the final result from the device that completes the computation first. If the edge device is incapable of completing the computational task, it just drops current task and waits for the execution result from the cloud.

The full model is different from a virtual machine or cloud offloading. A virtual machine only processes data at the datacenter and returns the execution result to the edge computers at the user side [20]. As such, the virtual machine may require a high datacenter processing capacity. To solve the challenge of big data processing, today's virtual machine resorts to remote computer cluster servers, which have efficient and high-speed hardware. Cloud offloading is quite the opposite, where the data and code are both processed at the edge devices [8]. The full model is much more comprehensive. It combines the benefits of both the virtual machine approach and cloud offloading.

As mentioned in Section 1, determining the execution location of an application program consumes time and resources. Many factors need be considered, such as computation power, communication delay, etc. [27]. In practice, edge devices have less computing capability compared with cloud servers and thus usually take more time to complete a computational task. Executing the application program on a cloud server may suffer a long communication delay [14], especially when the network bandwidth is low. Therefore, edge devices are often selected to execute small tasks, while the cloud server is used to execute computation-intensive tasks. However, it is sometimes hard to estimate the computational workload.

Current research focuses on distributing computational tasks to appropriate edge devices. Some research focuses on determining whether to execute a task on an edge device or a cloud server. Factors of consideration in deciding the execution location include the response time, memory space, data availability, data privacy, etc. [15]. However, the research on selecting the execution location is very limited and one cannot guarantee the execution time of edge-cloud computing model. To guarantee the execution-time performance in the edge-cloud model, we propose the full model.

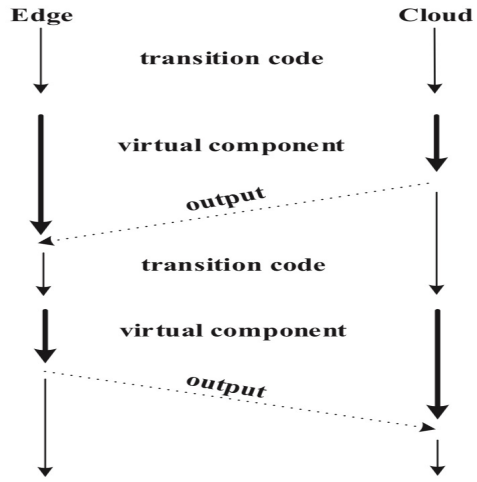


Figure 3: Joint Execution Mechanism in the Full Model

4 JOINT EXECUTION MECHANISM

4.1 Joint Execution Mechanism Overview

In our full model, an application program runs in parallel on both an edge device and a cloud server. We need to develop a mechanism that synchronizes the executions. Compared with previous computing models that adaptively decide the execution location (edge vs. cloud), the full model eliminates the need to analyze the prediction metrics as it executes the application on both the edge devices and the cloud server.

To execute an application program in the full model, we transform the computation tasks of the program into *virtual components*, as shown in Fig. 3. These virtual components can communicate with each other to share resources and synchronize the execution results between the edge devices and the cloud server. We will discuss the mechanism of how to transform a program into virtual components in the next section. When the system gets the result from the device that finishes the first, a cancellation signal is sent to the slower computing device. After receiving the cancellation signal, the slower device will stop execution immediately.

Fig. 3 illustrates the joint execution mechanism we implement for our full model. It consists of two parts. We first extract the computational tasks from application program and transform each computational task into a virtual component. We install a two-thread implementation mechanism on both the edge device and the cloud server. One thread is responsible for executing computational tasks and transmit execution results. The other thread uses a TCP connection for sending/receiving cancellation signals. When either the edge or cloud device completes the computation of a virtual component, a cancellation signal will be sent to the other end. Upon receiving such a signal, the device terminates the execution of the

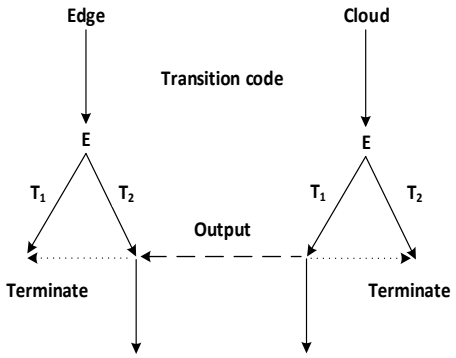


Figure 4: Two-thread Implementation (here, the cloud side finishes first).

current virtual component. After completing the current virtual component, the device will proceed to the next one until all virtual components are completed, which means the application program is also completed. We also use a synchronization mechanism between the edge and cloud devices to make sure that the virtual component are executed on both ends at the same time. The execution time on each of the devices is affected by the following factors:

- Data: The sizes of the input data and execution result can determine the transmission time. The transmission latency of the edge device is negligible compared with its execution time.
- Computing Capability: The computation speed is determined by the number and characteristics of the cores, the memory size and IO characteristics.
- Space: If the edge device cannot allocate more memory or the disk space is too small for the application program, the edge will terminate the current virtual component and wait for the result from the server.

4.2 Two-Thread Implementation for the Full Model

In this section, we describe our two-thread implementation mechanism. We use the application of network activity viewer/analyzer (NAVA) as an example to illustrate the implementation. In order to synchronize the execution of the NAVA software on both the edge device and cloud server, we create a method called *NetworkManager*, which has two threads. One thread uses a TCP connection to receive cancellation signals from the other end. The other thread controls the execution of the main program.

As shown in Fig. 4, the entrance to the *NetworkManager* method is denoted as E. Both the edge device and the cloud server share the same source program. Take the program at the edge as an example. At the start of the program, we create two threads. Thread T1 keeps on executing the original program C, and the other thread T2 sets up the TCP connection. If T1 completes the execution, it sends a cancellation signal to the cloud server to terminate the execution there. On the other hand, if T2 receives a cancellation signal, it will terminate the execution of T1 immediately. The same setup is also at the cloud server. Both the edge device and cloud server can return the execution result, which ever finishes first. Considering the limited computational resources of the edge device, if it cannot complete the computational task, T1 terminates immediately and T2 still waits for the cancellation signal from the cloud server.

4.3 Synchronization Mechanism between Edge and Cloud

We use cyclicbarrier to synchronize the application program's execution at the edge device and the cloud server [13]. The application program is controlled by threads. Cyclicbarrier is a mechanism by which all the threads can wait for each other to reach a common execution point. In cyclicbarrier, a counter is used to calculate the number of threads that have reached the common point. When that number is equal to a preset value, cyclicbarrier permits all the threads to execute at the same time.

In the program, we use the cyclicbarrier mechanism to make sure the connection between the edge device and the cloud server is set up first. After that, we begin to compute the virtual component at the same time.

4.4 Execution Time Analysis between Edge and Cloud

The execution time of a virtual component usually consists of transmission time and computing time [6]. The transmission time represents the time consumed to transmit the request and execution results between edge device and cloud server. Computing time is the time consumed to execute the computing instructions. We use D to denote the total data size transmitted between edge device and cloud server. We also use I to denote all the instructions executed on a computation device for a virtual component.

The computing time can be expressed as:

$$T_{calc} = \frac{I}{MIPS} \tag{1}$$

In equation 1, computing time equals to the total instructions to be executed divided by million instructions executed per second (MIPS). MIPS is a parameter to evaluate the performance of a CPU. It is determined by the number of cores and computer frequency.

$$\text{MIPS} = \frac{\text{Number of Cores} * \text{Computer Frequency}}{CPI * 1000000} \tag{2}$$

Normally, a computer with higher main frequency usually completes the computation task faster. CPI refers to the average number of clock cycles needed by per instruction. It is a definite number for each virtual component.

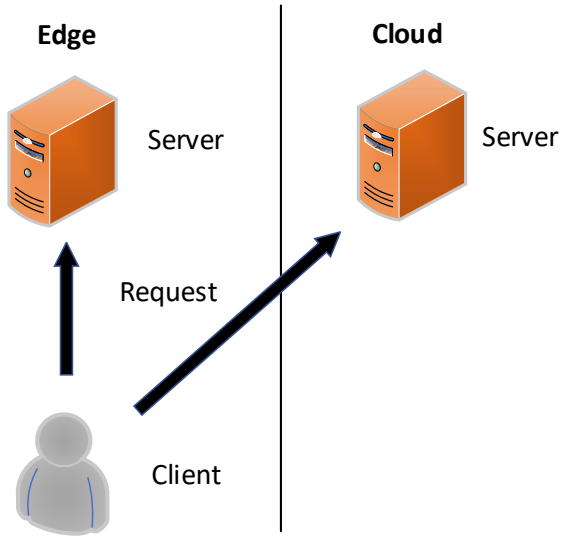
The transmission data usually consists of the request from the edge device to the cloud server and the execution results backhauled to the edge. As such, the transmission time can be expressed as:

$$T_{TX} = \frac{D}{B} \tag{3}$$

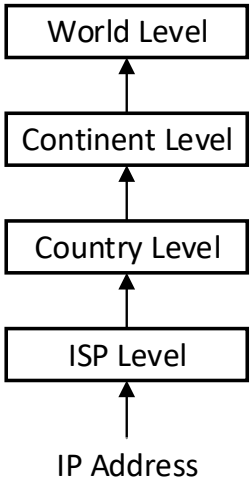
In equation 3, B represents the bandwidth of the Internet.

The response time of cloud server can be expressed as:

$$T_{server} = \frac{D}{B} + \frac{I}{\text{MIPS of Cloud Server}} \tag{4}$$



(a) NAVA under the Full Model



(b) Hierarchy in NAVA

The response time of edge device can be expressed as follow:

$$T_{edge} = \frac{I}{\text{MIPS of Edge Device}} \tag{5}$$

Actually, the transmission time of edge device is negligible compared with its computing time, such that we ignore the transmission time from the response time. The transmission time of edge device is mainly caused by hardware delay.

5 EXPERIMENTAL RESULTS AND PERFORMANCE EVALUATION

5.1 The Application: NAVA

We use the NAVA application to evaluate the performance of our full model. NAVA is a network analyzer that is useful for detecting and tracking malicious activities and for investigating potential threats in the network. In our case, it collects and analyzes the network flow information between the University of Florida (UF) campus and the users outside the campus. The network flow data contains information such as the number of packets, the number of connections, connection time, etc. The NAVA system is divided into two parts: the server and the client. The client can send a request to a server to obtain network information. In the full model, we set up the server part of NAVA on both an edge device and a cloud server. The client part of NAVA is at the edge. Whenever the client sends a request, the request is forwarded to both the edge device and the cloud server. As discussed earlier, we use threads to synchronize the executions at both ends, and we get the final result from the device that finishes the first.

NAVA is designed in a hierarchical structure [9]. The top level shows the general network information between the seven continents and our campus network. When we move the mouse to an icon that representing a continent, it shows the total number of network flows between that continent and our campus. After clicking at any of the continent icons, the next interface shows the network information between the countries in that continent and our campus routers. Each country usually has many Internet Service Providers (ISPs). The next level shows the network information between the ISPs from each country and our campus routers. The last level contains information between the outer IPs of each ISP and our campus routers. We can then find out specific information (e.g. the number of packets, connection time, the number of connections, etc.) between an individual IP and any campus router. The information of network flow can be used to analyze network behavior.

NAVA can be used to detect potential threats and attacks and help identify the vulnerable points in a network, by monitoring the activities between a specific outer IP address and our campus network. Abnormal behaviors can be distinguished from normal ones by analyzing the normal network flows. By summarizing the malicious network flows, we can also easily find the characteristics of the attackers. Tracing through the whole network, we can identify the victims and the attacking sources. Using the NAVA system, we have successfully identified worm attacks and DDos attacks. We can then reject the traffic from an attacking source.

5.2 Experimental Setting

For the experiments, we use a Dell Desktop and a HP Z840 server as an edge machine and a cloud server, respectively. The Dell desktop has a CPU of Intel(R) Core(TM) i7-2600, with 32GB RAM and a clock rate of 3.4GHz. It has four cores. We constrain the CPU utilization of edge device to 20% to increase the performance difference between edge device and cloud server. The HP Z840 server has an E5-2643v4 CPU (6-Core, 20M Cache, 3.4GHz), 256GB memory and disk space of total 9.6 TB. The bandwidth of the network between the machines is around 92.5Mbps to 94.4Mbps.

The NAVA application program was created based on the UF's campus network, which has two gateways and 16 core routers. The network flow data are saved in files. Each file records the network flow information connecting to a core router for a 5-minute period. The whole data set is 21.7GB, which covers a span of six months.

We compare the performance of our full model with the conventional edge computing model and conventional cloud computing model in executing the NAVA application. We use the same desktop and server to simulate edge computing and cloud computing, respectively.

5.3 Performance Evaluation

Tables I, II and III show the time to execute certain tasks using NAVA under the three computing models. The first column in the tables contains the computation tasks we use in the experiments. The second, third and fourth columns contain the execution time of the programs under the edge computing model, the cloud computing model and our full model (edge-cloud computing), respectively. The symbol '(C)' in the last column indicates that the result is obtained from a cloud device, and the symbol '(E)' indicates that the result is obtained from an edge device. For each computing device, we list the response time (R-time), transmission time (T-time) and computing time (C-time). These concrete displays can help evaluate the performance of edge device and cloud server. They also show how the transmission delay influences the final response time.

Table I shows the execution time of using NAVA to get the network flow information between UF and different countries. The workload of processing a country's information depends on the number of Internet service providers (ISPs) operating in that country. From the table, we see that in our full model, the execution results are obtained from the cloud server for some countries such as the United States. This is because U.S. has many ISPs and it requires more time for the edge device to calculate the network flow information, whereas the cloud device is much more powerful and the execution is faster. However, for the other countries, under our full model, the execution is faster on the edge device because the computation requirement is not high. If we compare the execution time of the three models, we see that our full model outperforms the cloud computing model on tasks with small computation requirement. The performance of the full model is better than the edge computing model on computation-intensive tasks.

Table II shows the execution time of using NAVA to get the network information between UF and different companies. We select several well-known large companies, which have many internal IPs connected to the UF routers. From this table, we see that our full model outperforms the edge computing model and it performs similarly to the cloud computing model. We also select several small companies with limited internal IPs connected to UF router. At this time, the full model performs similarly to edge computing model, which is better than cloud computing model.

Table III shows the execution time of scanning different anomaly activities including scanning, worm attacks and DDoS. There is a threshold that limits how frequent an outer IP can connect to the UF routers during a period of time. If an outer IP connects to the UF routers too many times (over the threshold), it is deemed as an attacker. We combine the properties of scanning, worm attacks and DDoS to classify the attacks. In this table, we set the threshold to be 10, which means an outer IP will be regraded as malicious if it connects to the UF routers more than 10 times over a predefined time interval. The cloud computing model shows superior performance in finding out all three types of attacks compared with the edge computing model. Our full model always maintains a good performance level similar to that of the cloud computing model.

When we analyze the transmission time and computing time of edge device and cloud server, we can detect that the transmission time of edge device is very trivial, which is mainly delayed by hardware response. The transmission time of cloud server accounts for an important proportion, such that the cloud server proves to be superior only when processing computing-intensive tasks. Since we reduce the performance of edge device, the computing time consumed by edge device is obviously longer than cloud server. This design is based on the premise that we usually select smart phones and IoT sensors as the edge devices which have limited computational resource.

We also notify that the computing time of full model is a little longer than either edge alone computing or cloud alone computing. We have added two-thread implementation mechanism to the full model, which spends time in synchronizing the execution of edge device and cloud server. The communication between edge device and cloud server also consumes some time. Through optimization, we have reduced the delay of two-thread implementation mechanism to a low point, such that the execution time of full model is close to the computing device completes faster.

To sum up, we observe that both the edge computing model and the cloud computing model can have bad performance under certain circumstances, while our full model is always in line with the better one of them, which demonstrates the superiority of our design.

6 RELATED WORK

Edge computing system has gained rapid and significant improvement due to technological development and research progress [26]. The increase in network bandwidth has greatly reduced the communication delay between computation resources [10]. The advancement of microprocessors and GPUs has increased the processing capacity of computers [5]. In addition to technological progress, new research results have also led to more efficient management and resource utilization of edge and cloud computing systems. These research results mainly concentrate on specifying data format, optimizing the edge computing structure and modifying the edge computing model [28].

Peng et al. [31] proposes a cloudlet mechanism to improve the end-to-end efficiency. Cloudlet is a small-scale cloud datacenter located at the edge of the Internet with powerful computing capability and lower latency. Current research on task placement mainly focuses on resource allocation. Hui et al. have designed a resource-allocation scheme for broadband network [12]. Tran et al. have designed a joint task-offloading and resource-allocation scheme for multi-server and edge computing networks [25]. Their algorithm has proved to be close to the optimal solution. In [21], the authors show that edge-cloud computing is beneficial for many applications, such as cloud offloading, video analytics, smart home/city, and collaborative edge.

Despite the progress, there remain some problems with the current edge-cloud computing models [22]. The current models always fix an execution location for a program in advance [1]. The edge

	Edge(R-Time T-Time C-Time)	Cloud	Full Model
UF-UNITED STATES	4793 4 4789	1941 1616 325	2085(C) 1754 331
UF-CANADA	395 2 393	193 170 23	214(C) 189 25
UF-UNITED KINGDOM	442 1 441	162 143 19	173(C) 153 20
UF-BAHAMAS	10 1 9	89 88 1	10(E) 1 9
UF-SWITZERLAND	83 1 82	134 127 7	89(E) 1 88
UF-GREECE	20 1 19	62 52 10	24(E) 1 23
UF-THAILAND	47 1 46	82 75 7	52(E) 1 51
UF-MYANMAR	10 1 9	90 89 1	11(E) 2 9
UF-AUSTRALIA	90 2 88	161 148 13	101(E) 2 99
UF-ARGENTINA	90 2 88	173 161 12	98(E) 2 96
UF-EGYPT	17 1 16	54 48 6	19(E) 1 18

Table 1: Time (in milliseconds) of producing statistics from network data between UF and individual countries under edge-alone computing, cloud-alone computing, and the full mirror model of edge-cloud computing. Under any runtime conditions, the full mirror model always performs similar to the better one of edge-alone and cloud-alone, with a slight increase in time for edge-cloud communication.

	Edge(R-Time T-Time C-Time)	Cloud	Full Model
UF-Google LLC	4633 2 4631	1452 1339 113	1656(C) 1538 118
UF-Facebook Inc.	379 3 376	170 154 16	176(C) 159 17
UF-Apple Inc.	1017 2 1015	369 340 29	375(C) 342 33
UF-Amazon Technologies Inc.	6784 2 6782	2565 2292 273	2730(C) 2455 275
UF-Shells	5 1 4	45 45 0	5(E) 1 4
UF-Delta Air Lines	6 1 5	44 43 1	8(E) 1 7
UF-Ford Motor Company	9 1 8	46 45 1	12(E) 1 11

Table 2: Time (in milliseconds) of producing statistics from network data between UF and individual companies under edge-alone computing, cloud-alone computing, and the full mirror model of edge-cloud computing. Under any runtime conditions, the full mirror model always performs similar to the better one of edge-alone and cloud-alone, with a slight increase in time for edge-cloud communication.

	Edge(R-Time T-Time C-Time)	Cloud	Full Model
Scanning	13390 1 13389	7428 589 6839	7517(C) 553 6964
WormAttack	29092 1 29091	11621 1060 10561	12108(C) 828 11280
DDoS	17220 1 17219	4571 564 4007	4798(C) 666 4132

Table 3: Time (in milliseconds) of UF Scanning Malicious Hosts. These tasks require scanning and analyzing a large amount of data, which makes cloud computing the winner.

and cloud devices lack enough cooperation [30]. Actually selecting the execution location is very time-consuming [18]. It needs to evaluate many factors: the performance of edge and cloud devices, the delay due to transmit data and the disk space on two ends [2]. On top of that, it is even impossible to know these factors in advance sometimes, such that the evaluation can not be done in advance. Fixing the execution location in advance can also make the edge and cloud devices work isolated from each other, which may waste many computation resources. Our proposed full mirror computation model can successfully solve the problem of selecting execution location and fully utilize the computation resources on both edge and cloud devices.

7 CONCLUSION

In this paper, we propose and design a full mirror computation model. In this model, the program is executed simultaneously on the edge device and cloud device, and the system always uses the result from the device that finishes the execution first. We use

the application of network activity viewer/analyzer to evaluate the new model. The full model is shown to always have excellent performance compared with running the application on either the edge alone or the cloud alone, since it incorporates the advantages of both. It is guaranteed to have the execution time comparable to the shorter one of the latter two approaches. Importantly, it achieves this performance level without an in-advance placement algorithm that must consider a set of complex factors such as the application's computing requirement, data requirement, characteristics of the available machines, and network bandwidth.

ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation of US under grant CNS-1909077 and a grant from Florida Center for Cybersecurity.

REFERENCES

[1] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie. 2018. Mobile Edge Computing: A Survey. *IEEE Internet of Things Journal* 5, 1 (2018), 450–465.

[2] Yuan Ai, Mugen Peng, and Kecheng Zhang. 2018. Edge computing technologies for Internet of Things: a primer. *Digital Communications and Networks* 4, 2 (2018), 77 – 86. http://www.sciencedirect.com/science/article/pii/S2352864817301335

[3] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. 2015. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys & Tutorials* 17 (June 2015), 2347 – 2376.

[4] A. Al-Shuwaili and O. Simeone. 2017. Energy-Efficient Resource Allocation for Mobile Edge Computing-Based Augmented Reality Applications. *IEEE Wireless Communications Letters* 6, 3 (2017), 398–401.

[5] A.Manz, Y.Miyahara, J.Miura, Y.Watanabe, H.Miyagi, and K.Sato. 1990. Design of an open-tubular column liquid chromatograph using silicon chip technology. *Sensors and Actuators B: Chemical* 1 (Jan. 1990), 249–255.

[6] F. Busch, W. Pfeffer, B. Kohlmeyer, D. Schüll, and F. Pühlhoffer. 1980. A position-sensitive transmission time detector. *Nuclear Instruments and Methods* 171, 1 (1980), 71 – 74. http://www.sciencedirect.com/science/article/pii/0029554X80900117

[7] X. Chen, L. Jiao, W. Li, and X. Fu. 2016. Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing. *IEEE/ACM Transactions on Networking* 24, 5 (2016), 2795–2808.

[8] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. 2016. Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing. *IEEE/ACM Transactions on Networking* 24 (October 2016), 2795–2808.

[9] CISCO. Online. *Cisco IOS NetFlow.* http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html

[10] Thinh Quang Dinh, Jianhua Tang, Quang Duy La, and Tony Q. S. Quek. 2017. Offloading in Mobile Edge Computing: Task Allocation and Computational Frequency Scaling. *IEEE Transactions on Communications* 65 (Aug. 2017), 3571–3584.

[11] Christian Esposito, Aniello Castiglione, Florin Pop, and Kim-Kwang Raymond Choo. 2017. Challenges of Connecting Edge and Cloud Computing: A Security and Forensic Perspective. *IEEE* 4 (March-April 2017), 13–17.

[12] J. Y. Hui. 1988. Resource allocation for broadband networks. *IEEE Journal on Selected Areas in Communications* 6, 9 (1988), 1598–1608.

[13] Java. Online. *Class CyclicBarrier.* https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CyclicBarrier.html

[14] Juan Liu, Yuyi Mao, Jun Zhang, and Khaled B. Letaief. 2016. Delay-optimal computation task scheduling for mobile-edge computing systems. *2016 IEEE International Symposium on Information Theory (ISIT)* (July 2016), 10–15.

[15] D. Melissourgos, S. Wang, S. Chen, Y. Zhang, O. Odegbile, and Y. Wang. 2020. An Experimental Study on the Impact of Execution Location in Edge-Cloud Computing. (2020), 145–151.

[16] Olufemi Odegbile, Shigang Chen, and Yuanda Wang. 2019. Dependable Policy Enforcement in Traditional Non-SDN Networks. *in Proc. of 39th IEEE International Conference on Distributed Computing (ICDCS)* (July 2019).

[17] O. Odegbile, S. Chen, and Y. Wang. 2019. Dependable Policy Enforcement in Traditional Non-SDN Networks. (2019), 545–554.

[18] G. Premsankar, M. Di Francesco, and T. Taleb. 2018. Edge Computing for the Internet of Things: A Case Study. *IEEE Internet of Things Journal* 5, 2 (2018), 1275–1284.

[19] B. P. Rimal, E. Choi, and I. Lumb. 2009. A Taxonomy and Survey of Cloud Computing Systems. (2009), 44–51.

[20] Tiago Gama Rodrigues, Katsuya Suto, Hiroki Nishiyama, Nei Kato, and Katsuhiro Temma. 2018. Cloudlets Activation Scheme for Scalable Mobile Edge Computing with Transmission Power Control and Virtual Machine Migration. *IEEE Trans. Comput.* 67 (Sept. 2018), 1287–1300.

[21] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3 (Oct. 2016), 637 – 646.

[22] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.

[23] Weisong Shi and Schahram Dustdar. 2016. The Promise of Edge Computing. *Computer* 49 (May 2016), 78–81.

[24] Syed Noorulhassan Shirazi, Antonios Gouglidis, Arsham Farshad, and David Hutchison. 2017. The Extended Cloud: Review and Analysis of Mobile Edge Computing and Fog From a Security and Resilience Perspective. *IEEE Journal on Selected Areas in Communications* 35 (Nov. 2017), 2586–2595.

[25] T. X. Tran and D. Pompili. 2019. Joint Task Offloading and Resource Allocation for Multi-Server Mobile-Edge Computing Networks. *IEEE Transactions on Vehicular Technology* 68, 1 (2019), 856–868.

[26] Blesson Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick, and Dimitrios S. Nikolopoulos. 2016. Challenges and Opportunities in Edge Computing. *2016 IEEE International Conference on Smart Cloud (SmartCloud)* (Nov. 2016), 18–20.

[27] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. 2015. Fog Computing: Platform and Applications. *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)* (Nov. 2015), 12–13.

[28] Wei Yu, Fan Liang, Xiaofei He, William Grant Hatcher, Chao Lu, Jie Lin, and Xinyu Yang. 2017. A Survey on the Edge Computing for the Internet of Things. *IEEE* (Nov. 2017), 6900–6919.

[29] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang. 2018. A Survey on the Edge Computing for the Internet of Things. *IEEE Access* 6 (2018), 6900–6919.

[30] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang. 2018. A Survey on the Edge Computing for the Internet of Things. *IEEE Access* 6 (2018), 6900–6919.

[31] YuanAi, Mugen Peng, and Kecheng Zhang. 2018. Edge computing technologies for Internet of Things: a primer. *Digital Communications and Networks* 4 (April 2018), 77 – 86.