DOI:10.3969/j.issn.1673-5188.2017.04.001
http://kns.cnki.net/kcms/detail/34.1294.TN.20171025.1001.002.html, published online October 25, 2017

A Transparent and User-Centric Approach to Unify Resource Management and Code Scheduling of Local, Edge, and Cloud

ZHOU Yuezhi[1], ZHANG Di[1], and ZHANG Yaoxue[2]
(1. Tsinghua University, Beijing 100084, China;
2. Central South University, Changsha 410083, China)

Abstract

Recently, several novel computing paradigms are proposed, e.g., fog computing and edge computing. In such more decentralized computing paradigms, the location and resource for code execution and data storage of end applications could also be optionally distributed among different places or machines. In this paper, we position that this situation requires a new transparent and user-centric approach to unify the resource management and code scheduling from the perspective of end users. We elaborate our vision and propose a software-defined code scheduling framework. The proposed framework allows the code execution or data storage of end applications to be adaptively done at appropriate machines under the help of a performance and capacity monitoring facility, intelligently improving application performance for end users. A pilot system and preliminary results show the advantage of the framework and thus the advocated vision for end users.

Keywords

cloud computing; fog computing; edge computing; mobile edge computing; resource management and code scheduling

1 Introduction

In the past several years, cloud computing has made a significant achievement by penetrating into our daily work and life through pervasive smart devices and fixed or mobile networks. Large or small companies increasingly adopt commercial clouds (e.g., EC_2 by Amazon, G Suite by Google, and Azure by Microsoft) or private clouds to host their computing or storage services. A cloud typically consists of hundreds or thousands of servers, facilitating flexible and rapid access to a shared pool of dynamically configured and nearly unlimited computational and storage resources. The two major advantages of cloud computing (i.e., the elasticity of resource provisioning and the pay-as-you-go pricing model) enable users to use and pay only for their actually needed resources, inspiring most companies to transfer their traditional computing facilities to cloud platforms.

In the meanwhile, it has witnessed the pervasive usage of smart mobile devices, such as smart phones, pads, and wearables to access thousands of services (e.g., Facebook, Twitter, Wechat) hosted on different cloud platforms. As of 2011, the global shipments of smart mobile devices had exceeded PCs. At the end of 2016, the global mobile users reached up to 7 billion [1]. End users of mobile devices often use the proliferated high-speed wireless access technologies (e.g. long term evolution (LTE), 4G, worldwide interoperability for microwave access (WiMax)) to access services from remote cloud platforms, making cloud computing into a new developing stage. However, even with more powerful computational and storage equipment, the process and energy of mobile devices are still limited due to the fact that mobile applications are now becoming more sophisticated than ever in terms of computing and storage requirements. To alleviate this resource bottleneck of mobile devices, the computation, storage, and networking functions are often offloaded to cloud, giving rise to a new mobile cloud computing (MCC) [2], [3].

However, the relatively poor performance of long-haul wide-area and wireless networks and the unpredictable contention for the shared system resources at consolidated cloud servers make end-user experiences far away from satisfaction. On one hand, limited network bandwidth, time-varying network quality, and long roundtrip time of long-haul wide-area or wireless networks [4], [5] would impair the code/data transmission and responsiveness of users' demands, especially degrading the us-

This work was supported in part by Initiative Scientific Research Program in Tsinghua University under Grant No. 20161080066, and in part by International Science & Technology Cooperation Program of China under Grant No. 2013DFB10070.

er experience of interactive applications. On the other hand, the computation consolidation at the shared resources (e.g. CPU, disks, and I/O) at cloud servers would bring with increased and unpredictable contention, which may lead to significant performance degradation in terms of code execution time and result-return latency. This problem of noisy-neighbors or multitenancy is an outstanding issue especially in public or commercial cloud [6]. Unstable and unpredictable performance due to multitenancy has also been observed when executing computation-intensive scientific tasks on commercial cloud platforms [7] and has even led companies to give up the usage of cloud [8].

Recently, many researchers have proposed that in order to address these challenges faced by traditional cloud computing in various application environments, new computing paradigms have to be brought forward under the principal of "bringing cloud closer to end user". Typical computing paradigms are proposed including fog computing [9], mobile edge computing [10], edge computing [11], and dew computing [12]. This means there is a paradigm shift from the centralization of cloud computing to the decentralization of fog/edge computing similar to the shift from Mainframe computing to PC computing. Considering that the resource management and code scheduling in cloud computing has to be changed due to the paradigm shift from PC computing, we position that the resource management and code scheduling in the post-cloud computing era would also call for new mechanisms and approaches. Obviously, how to manage resources and schedule codes in an efficient and/or optimal manner is very challenging in today's computing environments, considering the co-existence of local, fog, and cloud computing. In this paper, we advocate for a transparent and user-centric approach as a novel mechanism to manage the resources and codes among a local device, its peer devices, nearby fog/edge servers, and cloud servers in a unified manner from an end user viewpoint, and then schedule and distribute the code to execute on the appropriate machines in a transparent and intelligent way under the help of new facilities or tools. We position that this management and scheduling approach will retain the core advantages of using Micro clouds (formed with one or several fog/edge servers) and/or clouds as a support infrastructure but will put back the control and trust decisions to end users in an easy and hales-free way, allowing for novel mobile applications.

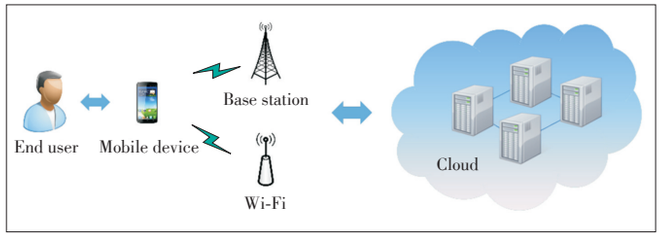
The remainder of this paper is organized as follows. In Section 2, we describe our idea and position of a transparent and user-centric mechanism for resource management and code scheduling in the virtually unified federal computing environment consisting of the local computing, fog/edge computing, and cloud computing. In Section 3, we present a software-defined code scheduling framework for illustration of our vision in details. In Section 4, we discuss a pilot system of the proposed code scheduling framework and evaluate the system through several experiments to demonstrate the effectiveness of the framework. Finally, we conclude the paper in Section 5.

2 Unifying Resource Management and Code Scheduling for Local, Edge, and Cloud

In a typical cloud computing environment, as shown in **Fig. 1**, there is a data center with logically unlimited compute and storage capacity. A lot of end-user devices surround the data center and connect to it with a wireless and/or long-haul wide-area network and access the services from the cloud. An end user manages the local resources of their devices (e.g., smartphones, iPads) and run their applications on the local machine. If needed, the end user requires to use the compute or storage resources from the data center. The cloud provider or the system administrator of the data center manages and schedules the resources of the cloud platform based on their benefits in a centralized manner and chooses the policy or strategy about where and how to execute the offloaded code from end users.

If the local compute resource is not enough for the end-user device, an end user can upload/offload these computation-intensive codes and related data to the cloud to execute there. Usually, two steps may occur during the period of the code execution. In the first placement (assignment) step, the uploaded codes is put on a specific server chosen in the data center by the code scheduler or dispatcher of the cloud platform based on current system status of resource usage. However, if the system status is changed due to some reasons (e.g., underutilization of server resources, conflicts of other resource consumers), the second reassignment (also referred to as migration in virtual machine (VM) based code offloading) step will occur and the code scheduler will transfer and schedule the code (accompanying with its data) to run on another more appropriate or optimal server in the data center. Note that this reassignment step to achieve an optimal performance or resource utilization may occur multiple time during the period of the code execution based on the tradeoff between the quality of experience (QoE) of end users and the resource operating costs/polices and relative benefits of the cloud provider.

Due to the natural ownership of the local end-user devices and the remote cloud servers, the resource management and code execution scheduling policies and strategy are made independently by end users and the cloud owner or provider based on their respective benefits and the service level agreement (SLA). Obviously, in the above centralized cloud computing



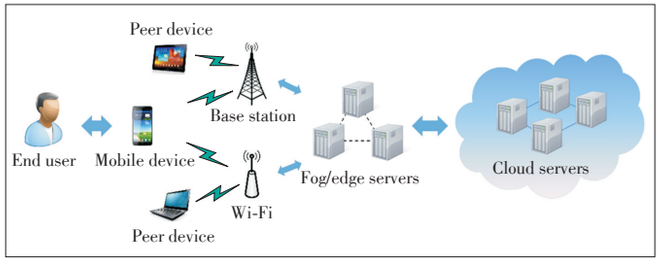
▲**Figure 1. Typical scenario of a cloud computing system.**

system, centralized resource management and code scheduling for offloaded codes can make usage of the resources of cloud platform by fully exploiting consolidation to improve the resource utilization and reduce the operating cost in the data center. It is also reasonable for end users to just require and demand the execution service based on the SLA. Therefore, the current centralized resource management and code scheduling without the involvement of end users is natural and suitable for cloud computing-based systems.

On the contrary, with the recently emerged fog/edge computing, there might be servers at nearby Nano data centers closer to end users in addition to servers located in remote data centers. Considering that fog/edge computing is just an extension of cloud computing at the edge of network, we believe that the traditional centralized cloud computing and the newly decentralized fog/edge computing will coexist for a long time. In such a situation, there would be several types of machines or places that end users can put their application codes to execute, including the local device, peer devices, nearby fog/edge servers, and remote cloud servers.

As shown in **Fig. 2**, we consider a general picture of mixed or federal computing environments in which end users can offload their computing tasks/codes to different computing devices, including local mobile devices, peer mobile devices, nearby fog/edge servers, and remote cloud servers in data centers. All of these computing devices or servers are interconnected through wireline or wireless links. Typically, the access link from the mobile devices to the fog/edge server is wireless. Generally, fog/edge servers are attached to the points of wireless access in a form of Nano data center or Micro cloud. Nearby Micro clouds of fog/edge are generally characterized by limited computation or storage resources, but are directly accessible by end users to avoid the additional communication delay of the Internet. Considering the cost of establishing a Micro cloud of fog/edge could be very little, even with only a single server, we believe that there could be multiple nearby Micro clouds around an end user and these clouds may be or not be owned and managed by the same provider.

Obviously, a mixed computing environment consisting of local devices, peer devices, edge servers, and cloud servers is much more complex compared to a single cloud environment of local devices and cloud servers. It is natural that the resource management and code scheduling mechanism in a mixed scenario would be much more complicated to achieve a global optimization. Currently, a cloud provider only commits to availability of their services, through SLAs with their clients. Therefore, there are numerous inherent factors that introduce uncertainty regarding the actual performance of an offloaded task by end users in a cloud computing infrastructure [13]. However, in a mixed computing environment, if the performance of an offloaded task degrades due to the uncertainty of a cloud server, an ideal code scheduler or dispatcher can reassign its code to execute on another fog/edge server in the whole federal computing system to exploit better resources than that of any server in the original cloud to furtherly reduce the execution time and mitigate the effects of uncertainty than before. This is the basic reason that we advocate for a new and unified resource management and code scheduling mechanism for efficiently and optimally utilizing the whole system resources across different computing devices and different Micro clouds or cloud servers in the post-cloud era. This will bring end users with better performance and much more satisfactory experience.



▲**Figure 2. Typical scenario of a mixed computing environment.**

In this section, we elaborate the resource management and code scheduling mechanism that enables end users to leverage the resources that across all of the computing devices and servers that may be owned and managed by different individuals or organizations. We positon that end users can select and choose where and how to execute their offloaded applications (e.g., code, data, networking) at their will and benefits. In contrast to the centralized approach where the cloud provider selects the cloud-wide optimal scheduling strategy, we advocate for transparent and user-centric approaches where the decisions are made independently by each end user or task. First, we delineate the performance benefits that arise for mobile end users in emerging computing paradigms and identify the opportunities to logically unify the resource management and code scheduling across different computing devices, Micro clouds and cloud servers to improve the performance in a user-centric viewpoint. Second, we explain the difficulties and challenges for end users to manage and schedule their tasks/codes across different computing devices, Micro clouds and/or cloud servers, and identify the requirement that an intelligent facility or tool is called for to help end users and even Micro cloud or cloud providers to achieve an optimal scheduling for them under several constraints to achieve a more efficient resource utilization among the whole mixed and federal system.

2.1 Unifying Resource Management and Code Scheduling from User-Centric Viewpoint

As stated above, the resource management and code scheduling mechanism in a cloud is a centralized one where the cloud provider selects the scheduling strategy for the cloud based on the benefits of their own. Obviously, this mechanism is a cloud-centric or server-centric approach. Because the cloud is a dynamically changing environment, its performance depends on numerous uncontrollable and unpredictable param-

eters. The performance of an offloaded task/code cannot be guaranteed and it may upgrade or degrade with the available resources on the hosted server. However, in a mixed computing environment of local, edge, and cloud, the current cloud-centric approach cannot utilize the resources across different computing devices, Micro clouds or cloud servers as a whole and thus cannot globally achieve an optimal scheduling strategy from the perspective of an end user. To elaborate the benefits of a new code scheduling mechanism, we give two hypothetical application scenarios based on code offloading of mobile applications.

In Scenario 1, because of his smart phone's limited computing power, Bob offloads a computation-intensive task into a server (S1) in a remote cloud owned and managed by the provider P1 to efficiently execute the code according to the SLA. Afterwards, he moves to another place where there is a nearby fog server (F1) that is free and its available computing power is much more than the host server S1 that is now competitive for several tasks of other end users.

In Scenario 2, Alice wants to offload her computation-intensive task into a nearby edge server due to her limited smart phone's computing power. There are two edge severs (E1 and E2) owned and managed by two providers (P2 and P3) respectively. Without further information, she chooses the cheaper E1 to execute the task. Afterwards, another woman also chooses the E1 to execute her code, reducing the computing power shared by Alice. At the same time, E2 is available with a much higher computing capacity.

With the current cloud-centric or server-centric code scheduling mechanism, there is not a facility to help Bob and Alice to schedule their codes across Micro clouds or clouds, thus what Bob and Alice can do is only waiting for his or her task to finish without any control of the execution of their offloaded tasks.

If we imagine that there is an ideal global resource management and code scheduling facility or tool that can obtain the information online about dynamics of the computing device, Micro cloud and cloud server, with the help of such a facility, Bob could move his task to F1 and Alice could reassign her task at E2 to leverage the more powerful computation resources to execute their tasks. Here we assume that the overhead of code and data transferring can be omitted compared with the achieved performance and the network link capacity and characteristics should be kept at a same level.

Based on the above observations and considerations, we advocate that novel resource management and code scheduling mechanisms are required to improve the performance of offloaded functions or tasks for end users and/or to achieve a more efficient global resource utilization. Here, we move to distributed approaches where the scheduling decisions are made independently by an end user or each offloaded function/task. To adhere to a realistic scenario, we assume that information about the dynamics of computing devices, Micro clouds and cloud servers is presented online, not available as a priori, just before a scheduling decision is made. Therefore, from the perspective of an end user, they can obtain these information to choose or select optimal or near-optimal machines to execute their functions/tasks. We also assume that the code/data uploading (transferring) or task migration process are standardized and can be carried out across different computing devices, Micro clouds, and clouds to form a virtually federal computing system. Considering the reality of industry standardization and industry alliance, we think this assumption is also realistic.

Therefore, we take a user-centric approach to schedule the tasks/codes to run at the selected optimal places or machines based on the benefits of end users from time to time, not just of cloud providers. In this way, the resource management and code scheduling of the whole system consisting of local, edge, and cloud is logically unified from the viewpoint of an end user. Note that the computing devices, Micro clouds or cloud servers are still owned and managed by their owners and they just make a logically federal computing system by interactions and agreements.

From a user-centric view, the virtually unified Internet computing environment consisting of data centers and clouds is at the core, Nano data centers (consisting of standard servers or routers, Wi-Fi points, and base stations with available computing capacities) and Micro clouds at the edge, and peer computing devices such as desktop PCs, tablets, and smart phones at the leaf. Based on this view, the resource management and code scheduling mechanism helps end users to leverage the resources of such a virtual system as a whole and achieve the most benefits. In summary, a user-centric approach to resource management and code scheduling encompasses the following elements:

1) Human-centered design: Human-centered design considers humans as an important factor in designing and developing a mechanism. First, humans should naturally interact with the system and impact the system behavior. Second, humans should be put in the control loop, so that users can take or retake control of their information. This leads to the design of novel crowdsourced and informed scheme where users control the information flows. Finally, the human-centered design also calls for novel and innovative forms of human-centered applications.
2) User-controlled decisions: The decision-making about the resource management and code scheduling is under the full control of end users. First, end users can choose and obtain all needed information about their nearby peer devices, fog/edge servers, and/or cloud servers. Second, end users are able to freely coordinate with computing devices or servers surrounding them to assign or delegate computation synchronization or storage to other computing devices or servers selectively. Finally, end users can get the status of their offloaded tasks and decide where and how to reassign their tasks to another computing device or server based on their

own intentions.

3) Local or edge proximity: It is more efficient to communicate and distribute data among nearby computing devices or servers than to use resources far away from end users. Here, of course, nearby can be understood both in a physical sense and a logical sense.
4) Local or edge trust: Personal and social sensitive data is clearly located in the local or the edge under the control of end users. Therefore, the control of trust relation and the management of sensitive information flows in a secure and private way must also belong to the end user.

2.2 Unifying Resource Management and Code Scheduling in a Transparent Manner

In Section 2.1, we advocate for a novel resource management and code scheduling mechanism that virtually unify and schedule the resources across the local, peer, edge, and cloud and leverage these resources to execute functions or tasks for an end user. However, even with the information about Micro cloud dynamic or cloud dynamic is acquired, it is still very hard for a common end user to choose and select where to upload and execute their codes. Therefore, an intelligent facility or tool should be developed to automate the assignment or reassignment process of offloaded tasks on behalf of end users, in order to make an easy and transparent manner to interact with such a virtual and complex system. Several research efforts have been done to enable an easy and transparent manner to use emerging complex computing systems [14], [15]. Here, we follow the concept of transparent computing as a basis for developing the unified resource management and code scheduling mechanism. To sum up, a transparent approach to resource management and code scheduling encompasses the following elements:

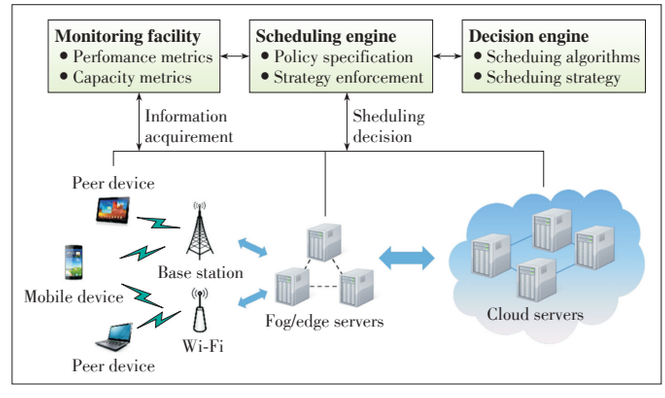
1) Easy-to-use interface: The interface for end users should be easy to use, at least keeping the same experience of offloading tasks to a cloud. Novel and easy-to-use interface are called for to make the code scheduling across different types of computing devices and servers much easier.
2) Hassle-free management: End users should just focus on their tasks, not the techniques about the installation, maintenance, and management of such a scheduling mechanism. This means that the techniques need to be transparent to end users and make the complex system of systems as a single system.
3) Intelligent decision-making: As it is very hard for humans to handle the information obtained from time to time and make a decision where and how to schedule the codes to execute, the novel mechanism or approach could semi-autonomously or autonomously help end users to make decisions of placement or reassignment according to their requirements. Of course, end users can intervene the process by some means.
4) Adaptive reaction: Due to the continuous changing characteristics of cloud servers, the new mechanism should also adaptively react to the changed environment. First, it should sense the changes of the hosted servers and the candidate servers that could be reassigned. Second, it should make a decision weather a reassignment is needed. Finally, it should select and reassign the code to another appropriate server to finish the remaining task.

3 Software-Defined Code Scheduling Framework with Capacity and Performance Monitoring

In this section, we illustrate our vision of a novel transparent and user-centric approach to resource management and code scheduling by presenting a software-defined code scheduling framework with capacity and performance monitoring facilities as a case study. As shown in **Fig. 3**, the framework has three parts: the monitoring facilities (MF), scheduling engine (SE), and decision engine (DE). The MF periodically acquires the information about task performance and capacity of any machines in the concerned scheduling domain. If an end user wants to initiate a scheduling process, the SE gets the needed inforamtion from the MF and sends the predefined scheduling policy and the related information to the intellegent DE. The DE makes a scheduling strategy by using various intelligent scheduling algorithms. An end user can initiate a scheduling process periodically or by giving an instruction through defining the scheduling policy for her/his tasks, e.g., the task performance degrading to an unsatisfying level.

To meet diverse scheduling requirements of various kinds of end users, end applications, and usage scenarios, the scheduling polices for different end users or tasks could be defined and specified in advance by a graphical tool or a specific police-defining language.

The framework provides various selections and places for computation and storage of end applications. This characteristic can be used to improve the performance of offloaded task and thus the user experience of end users by assigning computation and storage at appropriate machines. For example, the



▲**Figure 3. Software-defined scheduling framework with monitoring facility.**

computation can be carried out at four kinds of machines, i.e., *L*, *P*, *E*, *C*, which denote the local device, peer device, fog/edge server, and cloud server, respectively.

Given a computing task, the computation time of the task would be the sum of time consumed by different stages. Typically, there are five stages of the life of a task as follows.

1) Preparation: Once a new task is generated by an end user, the scheduler or dispatcher will choose one or several destination peer devices, Micro clouds, or clouds for her/him. Then the needed code and any input data required for the initialization of the task are transferred to the destinations. Finally, the task will be executed at the selected peer devices, Micro clouds, or clouds.
2) Assignment: Once the required code and data have been uploaded to the destination, a local scheduler or dispatcher is responsible for the assignment of the task to a specific process at the peer devices, or a specific fog/edge or cloud server.
3) Execution: This is the actual processing of the offloaded task/code. A new execution entity, such as a concrete process or a virtual machine is created for the specific task at the selected machine, and it starts to execute immediately.
4) Reassignment: The instance of the executed task/code may be transferred from its current machine to a new one to continue execution there for various reasons. As mentioned before, this reassignment stage may occur multiple times during the task execution. For the further execution of the task, the accompanying related data required for the initialization of the new execution instance has to be transferred to the new destination machine. Note that the reassignment, though optional, may occur several times during the lifetime of a task and may cross different computing devices, Micro clouds or clouds.
5) Outcome: At this stage the mobile end user retrieves the final results. We assume that once the whole processing is completed, the final data are immediately downloaded by the user through the current access technology or forwarded to another place to store.

The total time of finishing the offloaded task is the sum of the time spent at these different stages. Obviously, it depends on several factors. First, the selected machine at the assignment or reassignment stage mainly affects the execution performance because different computing devices or servers have different compute and storage capacity. Second, the constantly changing capacity of the selected machine due to the execution of other offloaded tasks also has an impact on the considered tasks. Third, the scheduled reassignment stage itself may significantly affect the execution due to the needed transferring of codes or data. Thus, the scheduling policy of reassignment, the times of the reassignment, and the selected reassigned machine would have an impact. Finally, the mobility of the end user may make the original scheduling decision inefficient or invalid. All of these factors, even if possible, are very hard to accurately predict and make an optimal scheduling decision. Therefore, it is very challenging to ensure that a scheduling scheme for an offloaded computation task is an optimal scheduling. Considering the situations in the real world, a realistic solution must be given on time to respond to the scheduling demand. We call for further researches to deal with these new challenges in the post-cloud era.

3.1 Monitoring Facilities of Capacity and Performance

If an end user or task initiates a scheduling periodically or due to the observation of performance degradation, the first thing to do for the framework is to learn about the current status of the entire system.

To collect performance metrics of the computation task and the capacity of the peer devices, nearby fog/edge servers, or cloud severs, the MF employs a proxy that resides inside the local and peer devices, nearby fog/edge servers or cloud servers. These proxies periodically collect the performance metrics of computation tasks and the capacity metrics of related candidate destination machines (e.g., CPU and memory capacity and utilization) and then send all the needed information to the SE to form a scheduling decision.

The selection of metrics is critical for the usability of a monitoring facility. The low-level metrics of the physical machines, such as CPU and memory utilization, are easy to collect, but they do not meet the requirements of some types of applications. The high-level metrics of software applications, such as click response, are difficult to collect and specific to certain types of applications. To explore, we develop a tool to choose the appropriate metrics by using a machine learning method based on historical usage data or conducting real-world experiments under different usage scenarios. These metrics can be specified automatically or manually by system administrators.

3.2 Software-Defined and Adaptive Code Scheduling

With the assistance of the above monitoring facilities, we develop a software-defined scheduling engine that can schedule the code in a predefined way or adapt to the changing computing environment based on the monitoring of the task performance in a timely manner. For example, if the performance metric of the monitored task is below a threshold, then the scheduling engine could be triggered, and the scheduling policy predefined by end users would be read and analyzed. Then, all the needed information is sent to the decision engine to form a scheduling strategy, which includes where and how to execute the computation or storage. The decision engine runs a scheduling algorithm and then obtains a scheduling strategy based on the current system status according to the predefined scheduling policy or adaptively learns a scheduling policy based on artificial intelligent rules from the changes occurred in the whole system status before. After getting a scheduling strategy from the decision engine, the scheduling engine takes actions to enforce the scheduling strategy by interacting with

the selected peer devices, Micro clouds, or clouds. The whole scheduling process is illustrated by **Algorithm 1**.

Algorithm 1: Software-defined code scheduling process

Input:
The set of performance metrics: M_1, M_2, ..., M_n
The specification file of scheduling policy: F;
Output:
Scheduling strategy S;

Step 1: Retrieve and analyze monitored metrics
Initialize performance metrics: M_1, M_2, ..., $M_n \Leftarrow$ NULL;
for each i $\in$[1, n] **do**
if M_i is defined **then**
Obtain M_i from MF;
end if
end for

Step 2: Retrieve and analyze scheduling policy

initialize scheduling condition: $Con \Leftarrow$ NULL;
retrieve F;
for each i $\in$[1, n] **do**
if M_i is defined in F **then**
Con = Con $\cup$ M_i;
end if
end for

Step 3: Judge and select the predefined scheduling algorithm
if $Con \neq$ NULL **then**
retrieve scheduling algorithm A from F;
return A;
else
return NULL;
end if

Step 4: Form a scheduling strategy
if $Con \neq$ NULL and $A \neq$ NULL **then**
run scheduling algorithm A;
return S;
else
return NULL;
end if

Algorithm 2 gives a general example of scheduling algorithm for the proposed framework. Considering that obtaining the optimal placement is time consuming and unnecessary in the real world, several heuristic algorithms are designed for the decision engine to run under different usage scenarios. The end user can give a rule to run any scheduling algorithm by specifying the scheduling policy in the scheduling framework. Therefore, the scheduling is software-defined and controlled

Algorithm 2: A general code scheduling algorithm

Input:
The performance metric: M_n
The threshold of the performance metric: Tre;
Output:
Place to execute the code: D;

Step 1: Estimate the performance at different machines
initialize the set of place candidates: $Pls \Leftarrow$ NULL;
for each $i \in L, P, N, C$ **do**
for each $j \in N, C$ **do**
Estimate the code preparation performance T_R^j;
Estimate the code transportation performance $T_T^{j,i}$;
Estimate the code execution performance T_E^j;
Calculate the whole performance $T^{i,j}=T_R^j+T_T^{j,i}+T_E^i$
if $T^{i,j} \geqslant Tre$ **then** $Pls = Pls \cup \{i, j\}$
end if
end for
end for

Step 2: Choose the best place to execute the code
if $Pls \neq$ NULL **then**
Choose the best performance candidate from Pls;
return $\{i, j\}$;
else
return NULL;
end if

by the current system status and the enforced scheduling policy. **Algorithm 3** gives a heuristic example of scheduling algorithm. Here, the algorithm just uses the network performance to predict the actual execution performance of the scheduled

Algorithm 3: A simple heuristic code scheduling algorithm

Input:
The network performance metric: M_{net}
The threshold of the metric: Tre;
Output:
Place to run the code: D;

Step 1: Estimate the performance at different machines
initialize the set of place candidates: $Places \Leftarrow$ NULL;
for each $i \in L, P, N, C$ **do**
Estimate the value of M net to Net^i;
if $Net^i \geqslant Tre$ **then** $Places = Places \cup i$
end if
end for

Step 2: Choose the best place to run the code

if $Pls \neq$ NULL **then**
  Choose the best performance candidate from Places;
  return i;
else
  return NULL;
end if

code, significantly simplifying the implementation complexity. Note that the network performance in the exemplary scheduling policy and the scheduling algorithm (Algorithm 3) is just for illustration and not specified. If the network performance means network bandwidth, the condition judgement of $Net^i \geqslant Tre$ is established. However, if it means network delay, the condition may be set as $Net^i \leqslant Tre$.

4 Pilot System and Preliminary Results

4.1 Pilot System

We developed a pilot system using commercial off-the-shelf desktop machines and servers to demonstrate the feasibility and effectiveness of the proposed software-defined code scheduling framework based on the transparent computing [15].

The pilot system uses VM technology to encapsulate the codes of OSes and their applications. The computation or storage can be scheduled and executed on three types of machines or servers, namely, local machines, nearby fog/edge servers, and remote cloud servers.

To simplify the implementation, a heuristic threshold-based scheduling algorithm is designed to run when the performance is below a predefined threshold. More details about the pilot system can be found in [16].

4.2 Preliminary Results

Two sets of experiments are conducted to show the effectiveness of the proposed framework. In all experiments, local machines use Intel Core i7-4790 (8 cores, 3.6 GHz) chips, with 12 GB DDR3 RAM, a 1TB Seagate 7200 rpm hard disk, and a 1 Gbps Realtek RTL8139C (plus Fast Ethernet) NIC. The nearby fog/edge servers use Intel Xeon e5-2620 (24 cores, 2.0 GHz) chips, with 24 GB DDR3 1333 RAM, one 1 TB Western Digital 7200 rpm RAID5 hard disk, and a 1 Gbps NetXtreme BCM 5720 Gigabit Ethernet network card. Local machines and nearby servers are connected by a TP-Link TLSG-1048 Ethernet switch with 48 1 Gbps ports. The OS and software code are encapsulated as VMs and scheduled to run at the local machines, nearby servers, and Amazon cloud servers, which are configured with one virtual CPU, with 2 GB memory and a 50 GB hard disk. The nearby servers use XenServer 6.0.2 OS. The local machines use Ubuntu 14.04, and a VM based on VirtualBox 4.3.36 runs on every machine. The VMs run Windows 7 SP1 with 1024 × 768 (32-bit color depth) resolution. The local machines access the VM running on the nearby server through SSH virtual network computing (VNC) 1.0.29 and access the VM running on the Amazon WorkSpace cloud server through Amazon WorkSpace 2.0.8205 client.

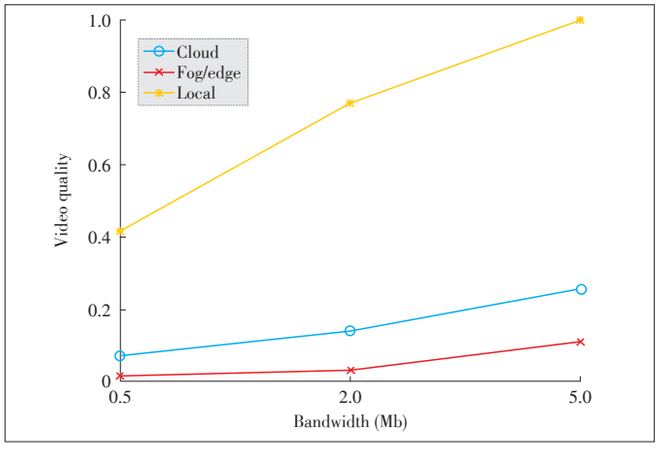
First, we evaluate the performance under different scenarios with standard 2D graphics test suite of PassMark Performance Test 8.0. This test suite includes operations of drawing lines, bitmaps, fonts, text, and GUI elements. To be more specific, eight tests are: simple vector, complex vector, fonts and text, Windows interface, image filters, image rendering, direct 2D, and overall 2D performance. We also compare the performance in three scenarios: scheduling code to run at the remote Amazon cloud server, the fog/edge server (accessing with VNC), and the local machine.

As shown in **Table 1**, the overall 2D performance of executing code on local machines outperforms that on the nearby fog/edge server and the fog/edge outperforms the cloud. This means that the performance is worse when the software codes are executed farther away from the end user. However, in the test of Windows interface, the fog/edge gets a slightly better performance than the other two, and in the test of image rendering, the cloud achieves the best performance. These two tests show that the performance will be better if the codes are scheduled to run at appropriate machines.

Second, we evaluate the video playback performance by playing a 21-second video clip, which has 320 × 240 pixels and 24 frames per second, in the 1024 × 768 resolution with Windows Media Player 12.0. The performance is measured with slow-motion benchmarking [17], which uses a packet monitor to capture network traffic for quantifing performance in a non-invasive manner. The video playback quality defined in the slow-motion benchmarking is taken as the performance metric. The results are shown in **Fig. 4**. We evaluate the performance under different scheduling strategies. In each scheduling strategy, the network bandwidth is changed to see how it affects the video quality. The video quality of the local machine with 2 Mb bandwidth (nearly 1) is better than that of the fog/edge server (nearly 0.11 with VNC) and the remote Cloud server (nearly 0.25) when the codes are scheduled to run at the local machine. This result indicates that adaptive scheduling can

▼**Table 1. Performance of 2D graphics (Scores)**

Test	Cloud	Fog/Edge	Local
Simple vector	29	28.6	38.8
Complex vector	109.1	171.6	191.5
Fronts and text	136.1	151.7	159.9
Windows interface	115.5	116.6	109
Image filters	608	672	725
Image rendering	575	513	507
Direct 2D	6.6	8.5	13.8
Overall 2D Performance	481.5	554	659



▲**Figure 4. Video quality performance.**

sharply enhance video playback quality, and such enhancement substantially improves the end-user experience. A comparison of the video quality with different network bandwidths shows that the video quality is highly sensitive to the bandwidth. This also explains that our adaptive scheduling framework can achieve better performance by scheduling the software codes to run at machines with higher network bandwidth to end users.

5 Conclusions

The recently emerging computing paradigms, such as fog/edge computing, have brought great changes to the computing environments for mobile end users and also numerous opportunities for creative applications. However, the current cloud-centric or server-centric resource management and code scheduling mechanisms are not suitable in the complex computing environment that consists of different Micro clouds or clouds. Thus, we advocate for new and novel resource management and code scheduling mechanism to deal with such a challenge and elaborate the details of this vision. To illustrate this vision, we proposed a software-defined scheduling framework to assign or reassign computation and/or storage to appropriate machines, including peer computing devices, nearby fog/edge servers and remote cloud servers. To demonstrate the effectiveness in real world, we also developed a pilot system. The pilot system shows that our vision and approach have good potential to further improve the performance of end applications and user experience in today's emerging computing environments by adaptively scheduling the codes to execute on appropriate machines. Further research and exploration are also called for realizing such a transparent and user-centric approach to globally achieve efficiency of resource utilization and optimal performance of code execution.

References

[1] B. Sanou. (2017, July 22). ICT Facts and Figures 2016 [Online]. Available: http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2016.pdf
[2] H. Flores, P. Hui, S. Tarkoma, et al., "Mobile code offloading: from concept to practice and beyond," *IEEE Communication Magazine*, vol. 53, no. 3, pp. 80–88, Mar. 2015. doi: 10.1109/MCOM.2015.7060486.
[3] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: a survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84–106, Jan. 2013. doi: 10.1016/j.future.2012.05.023.
[4] D. Chu, A. Kansal, J. Liu, and F. Zhao, "Mobile apps: it's time to move up to condos," in *Proc. 13th Usenix Conference on Hot Topics in Operating Systems*, California, USA, May 2011, pp. 16–16.
[5] N. Tolia, D. Andersen, and M. Satyanarayanan, "Quantifying interactive experience on thin clients," *Computer*, vol. 39, no. 3, pp. 46–52, Mar. 2006. doi: 10.1109/MC.2006.101.
[6] A. Williamson (2010, Jan. 12). Has Amazon EC_2 Become over Subscribed [Online]. Available: http://alan.blog-city.com/has_amazon_ec2_become_over_subscribed.htm
[7] A. Iosup, S. Ostermann, M. N. Yigitbasi, et al., "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 931–945, Jun. 2011. doi: 10.1109/TPDS.2011.66.
[8] Mixpanel Engineering. (2011, Oct. 27). Mixpanel: Why We Moved Off the Cloud [Online]. Available: https://code.mixpanel.com/2011/10/27/why-we-moved-off-the-cloud
[9] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. MCC' 2012*, New York, USA, Aug. 2012, pp. 13–16. doi:10.1145/2342509.2342513.
[10] H. Li, G. Shou, Y. Hu, and Z. Guo, "Mobile edge computing: progress and challenges," in *Proc. 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, Oxford, UK, Mar. 2016, pp. 83–84. doi: 10.1109/MobileCloud.2016.16.
[11] W. S. Shi, J. Cao, Q. Zhang, et al., "Edge computing: vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Jun. 2016. doi: 10.1109/JIOT.2016.2579198.
[12] Y. W. Wang. (2015, Nov. 10). The Initial Definition of Dew Computing [Online]. Available: http://www.dewcomputing.org/index.php/2015/11/10/the-initial-definition-of-dew-computing
[13] L. Gkatzikis and I. Koutsopoulos, "Migrate or not? exploiting dynamic task migration in mobile cloud computing systems," *IEEE Wireless Communications*, vol. 20, no. 3, pp. 24–32, Jun. 2013. doi: 10.1109/MWC.2013.6549280.
[14] R. Murch, *Autonomic Computing*. USA: IBM Press, 2004.
[15] Y. Zhang, K. Guo, J. Ren, et al., "Transparent computing: a promising network computing paradigm," *IEEE/AIP Computing in Science & Engineering*, vol. 19, no. 1, pp. 7–20, Jan. – Feb. 2017. doi: 10.1109/MCSE.2017.17.
[16] Y. Zhou, W. Tang, D. Zhang, and Y. Zhang, "Software-defined streaming-based code scheduling for transparent computing," in *Proc. 2016 International Conference on CBD*, Chengdu, China, Aug. 2016, pp. 296–303. doi: 10.1109/CBD.2016.058.
[17] J. Nieh, S. J. Yang, and N. Novik, "Measuring thin-client performance using slow-motion benchmarking," *ACM Transactions on Computer Systems*, vol. 21, no. 1, pp. 87–115, Feb. 2003. doi: 10.1145/592637.592640.

Manuscript received: 2017-05-10

Biographies

ZHOU Yuezhi (zhouyz@mail.tsinghua.edu.cn) is an associate professor in the Department of Computer Science and Technology, Tsinghua University, China. His research interests include distributed system, mobile network, and transparent computing system.

ZHANG Di (dizhang@tsinghua.edu.cn) is a postdoctoral researcher in the Department of Computer Science and Technology, Tsinghua University, China. His research interests include distributed system, mobile network, and transparent computing system.

ZHANG Yaoxue (zyx@csu.edu.cn) is a professor in the School of Information Science and Engineering, Central South University, China. His research interests include transparent computing, pervasive computing, and big data.