Dynamic Scheduling Method for Cooperative Resource Sharing in Mobile Cloud Computing Environments

Kyunglag Kwon[1], Hansaem Park[1], Sungwoo Jung[1], Jeungmin Lee[1] and In-Jeong Chung[1,*]
[1] Department of Computer and Information Science, Korea University
#302, 2nd Science and Technology Building, Sejong-ro 2511, Sejong City 339-700, Republic of Korea
[e-mail: helpnara, park11232000, sigran0, wjdals543, chung@korea.ac.kr]
*Corresponding author: In-Jeong Chung

Received July 27, 2015; revised October 2, 2015; revised November 16, 2015; accepted December 20, 2015; published February 29, 2016

Abstract

Mobile cloud computing has recently become a new paradigm for the utilization of a variety of shared mobile resources via wireless network environments. However, due to the inherent characteristics of mobile devices, a limited battery life, and a network access requirement, it is necessary for mobile servers to provide a dynamic approach for managing mobile resources efficiently in mobile cloud computing environments. Since on-demand job requests occur frequently and the number of mobile devices is drastically increased in mobile cloud computing environments, a different mobile resource management method is required to maximize the computational power. In this paper, we therefore propose a cooperative, mobile resource sharing method that considers both the inherent properties and the number of mobile devices in mobile cloud environments. The proposed method is composed of four main components: mobile resource monitor, job handler, resource handler, and results consolidator. In contrast with conventional mobile cloud computing, each mobile device under the proposed method can be either a service consumer or a service provider in the cloud. Even though each device is resource-poor when a job is processed independently, the computational power is dramatically increased under the proposed method, as the devices cooperate simultaneously for a job. Therefore, the mobile computing power throughput is dynamically increased, while the computation time for a given job is reduced. We conduct case-based experiments to validate the proposed method, whereby the feasibility of the method for the purpose of cooperative computation is shown.

Keywords: Mobile resource management and sharing, mobile computing, job scheduling, cooperative computing

This research is supported by Korea University Grant, and Brain Korea 21 Program for Leading Universities and Students (BK21 PLUS) Program.

http://dx.doi.org/10.3837/tiis.2016.02.003 **ISSN : 1976-7277**

1. Introduction

Cloud computing is a model for sharing a large number of on-demand services and configured computing resources via a heterogeneous, broad network access. The model eases the burdens of a rich set of computational requirements such as infrastructure, flexibility, and resources [1]. According to a Forrester research report, the broad market is expected to reach $241 billion in 2020, while in 2010 it was $40.7 billion [2]. Cloud computing is a fast-growing technology, making it possible for cloud service consumers to share varieties of software applications, hardware platforms, and infrastructures in the form of services such as Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) that are delivered over the Internet [3]. Cloud service providers allow their customers to perform numerical computations with a high time complexity such as big data processing for scientific or business purposes. They also provide their customers with large-scale online storage, databases, and applications for on-demand consumer requests in real time, enabling their consumers to concentrate on their core tasks instead of concerns regarding infrastructure, flexibility, or resource availability [4].

However, each mobile resource management server should be able to provide dynamic load balancing for efficient mobile resource sharing and job scheduling in real time because of the characteristics that are inherent to mobile devices such as the large quantity in existence, a limited computational power and battery life, and the need for network access. In addition, on-demand job requests in mobile cloud computing environments are frequently received by mobile cloud service providers. Furthermore, mobile resources need to be managed with a method that is different from those that are used in conventional cloud computing environments, since the number of mobile devices that are controlled by a service provider is drastically increased to maximize the computational power.

Our main goal is to design a cooperative, mobile resource sharing method that enables powerful computation in mobile cloudlets. We therefore considered both the intrinsic properties and the number of mobile devices to propose a cooperative mobile resource sharing method. In the proposed method, we divided participants into the following two groups: service providers and service consumers. In contrast to the conventional monitoring and allocation approaches for resources in server-based cloud computing environments [4, 5], a service's consumers can also participate as service providers. This means that the idle resources in their mobile devices may be utilized for jobs that are requested by fellow consumers to improve the computational power that is available for problem solving. Next, we describe the scenario-based experiments that we conducted in a performance study using the suggested method. We then demonstrate how the method works by measuring the execution times for the given jobs and show the way in which a great amount of resources can be managed by the cooperative computational model.

The remainder of this paper is organized as follows. Section 2 provides background and an overview of related work, and Section 3 gives the problem statement. Section 4 presents the cooperative mobile resource sharing method in detail. The performance study, including data, experiments, and validation of the suggested method, is described in Section 5. Finally, we conclude our paper with implications and further research directions in Section 6.

2. Related Works

Recently, the increasing popularity of mobile devices and the rapid development of wireless network technologies have shown the feasibility of applying the previous cloud computing paradigm to mobile environments, leading to the emergence of mobile cloud computing. Mobile cloud computing is a combination of cloud computing and mobile computing that allows mobile users to use rich computational resources via wireless networks in heterogeneous environments [6-8]. In contrast with conventional mobile cloud computing, each mobile device in this paper can be either a service consumer or a part of a service provider in the cloud, since the computational power of each mobile device is drastically increased. Each device has a low computational power when a job is processed independently, but the computational power can increase dramatically when numerous mobile devices cooperate to process a job simultaneously. This cooperation provides a rich throughput of computation and a decrease of the computational time for a problem with a given time complexity.

Cloud computing technology is composed of a variety of service models, including the previously mentioned SaaS, PaaS, and IaaS [1], as well as Desktop-as-a-Service (DaaS) and Resource-as-a-Service (RaaS) [9]. In addition, as the number of mobile devices increases greatly, a new hierarchical cloud computing architecture has emerged [10]. The corresponding issues, however, include pricing and lease duration, resource granularity, market-driven resource pricing, and a tiered service provision [9]. To address these issues, resource monitoring and allocation within the cloud need to be properly conducted. It allows resource-poor mobile devices to leverage elastic, powerful resources in heterogeneous cloud environments toward unlimited functionality (i.e., computational power), storage, and mobility [6, 8]. Moreover, there are models specified for mobile cloud computing paradigm such as Mobile as a Service consumer (MaaSC), Mobile as a Service Provider (MaaSP), Mobile as a Service Broker (MaaSB), and Mobile as a Representor (MaaR) [11].

A large amount of research has been conducted on mobile resource sharing methods for the efficient allocation or reallocation of resources in mobile cloud computing environments [6-8, 12]. Mobile cloud computing comprises the following two different models: the client–server communication model and the peer-to-peer communication model [12]. In the first model, the remote cloud provides data storage and computing services while the mobile devices are clients that access the service through wireless networks. The vast computational resources of remote cloud servers can enable computation-intensive applications on mobile devices. In contrast to the first model, the second model has emerged due to the increasing memory and computational power of mobile devices. A group of neighboring mobile devices can thus connect via a wireless network to form a mobile cloudlet, enabling them to provide their resources, as service providers, for other mobile devices. In [12], the authors introduce terms such as “initiator,” “cloudlet properties” (cloudlet size, cloudlet node’s lifetime, and reachable time), “upper” and “lower bounds” on computing capacity, and “long-term computing speed” of a mobile cloudlet, and they define them in a mathematical way. However, the methods that are proposed in [12] and [13] make many assumptions. In [13], the authors propose a mobile resource allocation method that is based on priority and time consumption, whereby the ALSALAM algorithm that subdivides jobs and represents them according to a graphical presentation method is used. In this paper, the proposed method uses a different architecture that allocates mobile resources based on priority, the performances of mobile resources, and the demanded resources. Since our method consists of a resource handler, it considers dynamicity and practicality in terms of resource reallocation, whereas the method in [13] assumes that the job must be processed to avoid its reallocation. Moreover, we focus on comparing the performance measurement with optimal and heuristic solutions. For the

proposed method, we demonstrate the effectiveness of its resource management by examining the usage of the available mobile resources and the allocation of these resources to tasks, whereas the method of [13] focuses on the results of the analyses of the relationships among the major obstruction factors in mobile cloud environments.

In this paper, we describe a mobile resource allocation problem in mobile cloud computing environments and then formulate the problem by specifying the mobile resource availability constraints. We do not consider mobile devices to be resource-poor inasmuch as large manufacturers have announced the ample resources of their devices such as multicore CPUs (even quad- or octal-core CPUs in 2015), and RAM and data storage that exceed 2 GB and 16 GB, respectively, which is in contrast with previous devices [10, 11]. Such developments suggest that the combined computing power may be sufficient when many mobile devices cooperate with each other in mobile cloudlets via wireless networks in a cooperative computation approach. A group of mobile resources (called a "mobile cloudlet") can therefore process a huge job that is composed of many small, independent sub-tasks. Building on these assumptions, we take full advantage of these mobile devices by using them concurrently in a mobile cloudlet under a cooperative, mobile resource sharing method.

3. Problem Statement

Mobile cloud computing environments have the following accompanying requirements. First, many aperiodic, on-demand jobs are submitted to mobile cloud service providers in real time, and each job is composed of multiple independent tasks that are to be properly distributed to remote resources. In this study, we assume that a job can be partitioned into several tasks and that these tasks are not interdependent. To meet the requirement of a real-time capability, a mobile cloud service provider needs to manage a high number of increasing mobile resources with a low time-complexity method. This is because of the characteristics that are inherent in wireless network mobility such as network instability and energy resources that are in themselves limited; therefore, an appropriate resource scheduling algorithm is required.

Secondly, the status of mobile resources in the cloud should be monitored and managed dynamically. As the mobile environment is dependent on each user's preferences and is in constant flux because of sudden calls, texting, web browsing, etc., the resource availability status of each mobile device is too erratic to monitor consistently; therefore, the service provider should check the status when a job is submitted. In addition, basic mobile functionalities should be guaranteed for mobile participants while they are providing their resources for the cloud. We therefore suggest a cooperative mobile resource sharing method regarding the monitoring and management of resources. We also has been devised in consideration of the previously mentioned mobile device characteristics such as unstable network access and limited energy (battery) resources.

Since many problems in big data processing take a lot of time due to repetitive computations, we have employed the proposed, cooperative method to solve these problems. The characteristic of big data processing lies in that one big job is divided into many small tasks that require many repetitive computations. Thus, it takes a long time to solve one big problem in an entire system. For this reason, we propose a dynamic scheduling method for cooperative resource sharing in mobile cloud computing environments. The proposed method is applicable to a variety of fields in the big data processing. For example, such a variety of application ranges not only from social network analysis or graph theory in social web environments, but also to matrix multiplication problems as well as Time Projection Chamber (TPC) tracking algorithm [14, 15], etc. As demonstrated in **Fig. 1**, a job in TPC tracking in physics consists of

many events and each event consists of tasks such as sector tracker, cluster finder and TPC sector. Each task in the event can be processed in parallel, and is performed as the unit of sector independently.

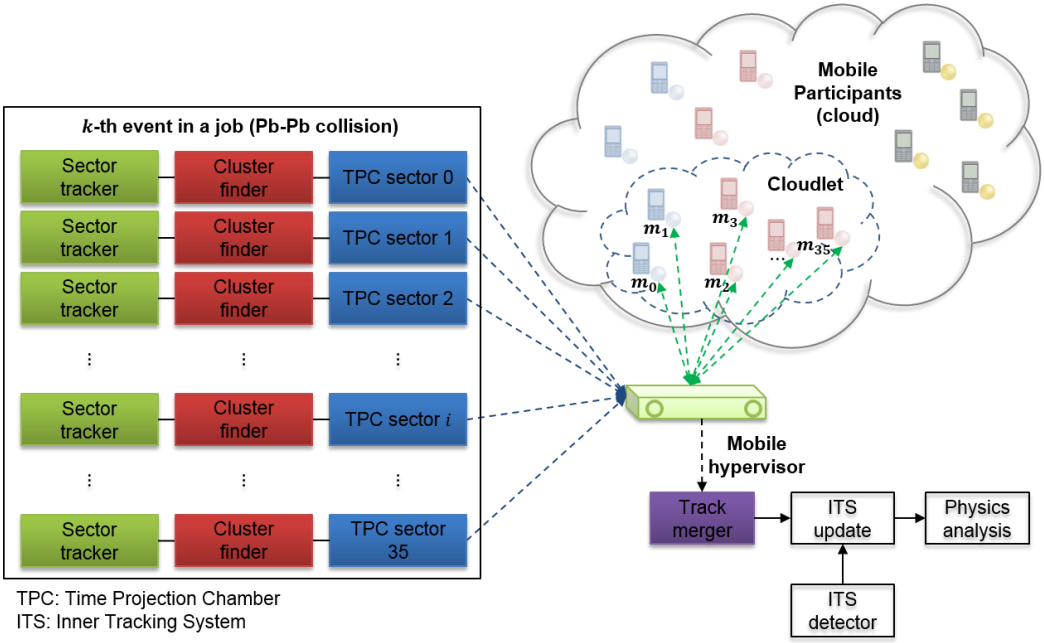


Fig. 1. An example of application scenario where a task can be divided into several independent tasks

4. Proposed Methodology

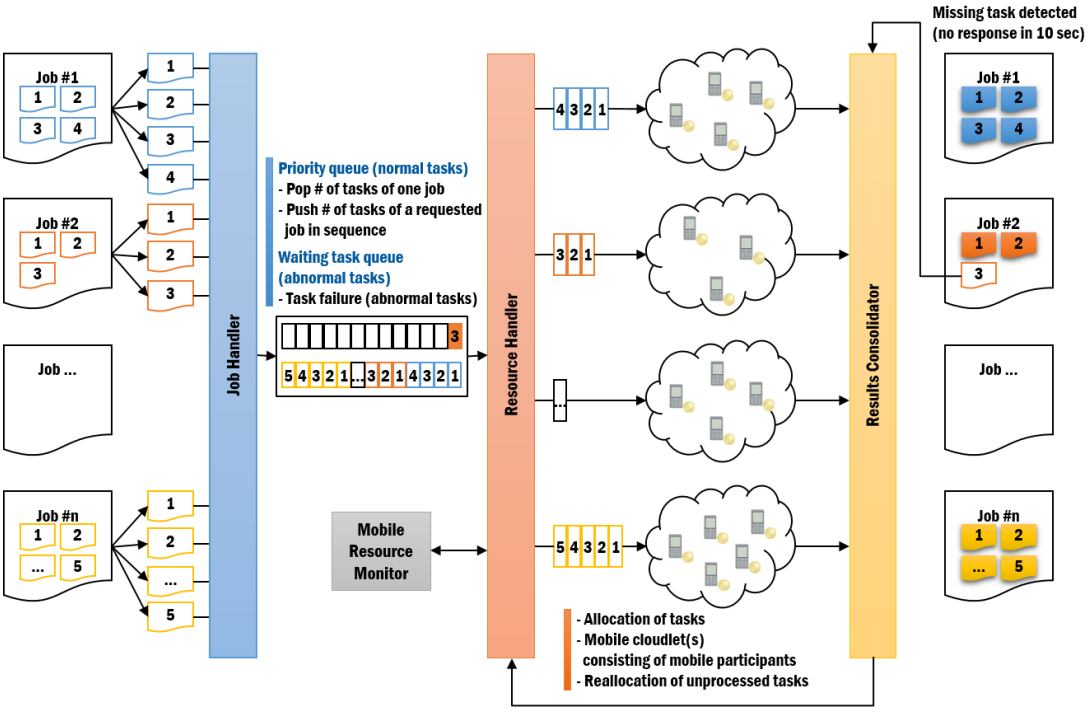


Fig. 2. Overall workflow of the proposed architecture

Fig. 2 shows the overall workflow of the proposed system architecture that consists of four main components such as mobile resource monitor, job handler, resource handler, and results consolidator. In this paper, we explain four modules, which execute tasks cooperatively to process a large job in mobile cloud computing environments. In **Fig. 2**, we get information on the available amount of resources and the amount required by each task from mobile resource monitor module and job handler. Job handler is responsible for dividing a big job into small tasks to collect amount of resource required by each task. The resource handler then assigns mobile resources to each task by using our proposed scheduling algorithm based on collected resource information. Lastly, there is the results consolidator to collect results for a given task. All these four modules perform each function efficiently to process a large job in mobile cloud computing environments. Mobile participants can be both consumers and providers of a cloud service. A mobile cloud service consumer can submit a new job request to a cloud service provider, and a mobile cloud service provider can provide consumers with its own resources for the processing of a given task(s) within the cloud service provider.

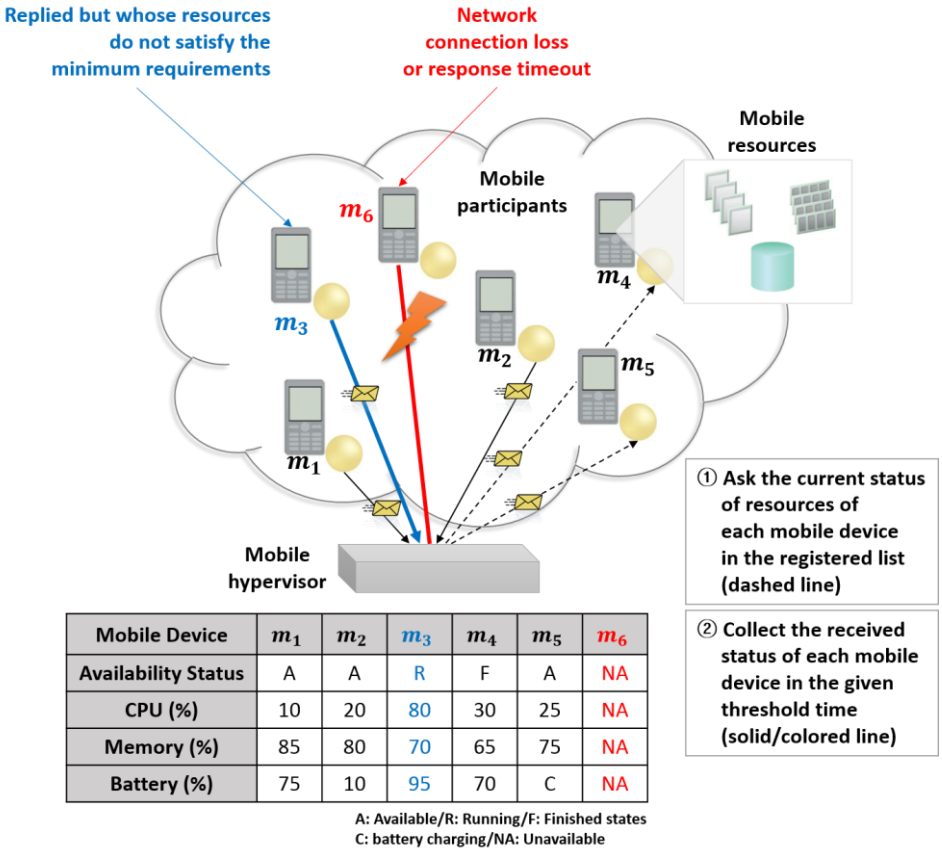
4.1 Mobile resource monitor

Fig. 3 demonstrates how the mobile resource monitor of the proposed method works and collects mobile resource statuses in the hypervisor under a variety of conditions. In **Fig. 3**, the mobile resource monitor periodically requests and receives data regarding the current network availability, remaining battery life, CPU, and memory before the allocation of resources and the assignment of requested jobs. The module asks for the current battery status for each mobile device since mobile device performance is inherently dependent on battery life, whereas cloud resources with a constant supply of power (e.g. servers) are not.

If there is no response from the registered mobile device within a predefined time limit (caused by, for example, link loss or timeout), the module considers it an unavailable resource at that time, as shown in the timeout case of the mobile device m_6 in **Fig. 3**; in this case, the network availability value is 0. Otherwise, the module dynamically adds the device to the available resource list (pool), or removes it from the list, and then marks its availability status, as shown in **Fig. 3** (in this case, the network availability value is 1). Another case is that of a mobile participant with enough CPU and memory resources, but with a low battery percentage such as the mobile device m_2 in **Fig. 3**; in this case, the monitoring module does not add the participant to the available resource pool. To be specific, the performance of each mobile device is highly dependent on the amount of remaining battery life. In general, most of mobile devices in real lives have a power saving mode to extend battery life by restricting applications in background, or reducing use of features such as brightness of display, vibration, sensors, etc. when battery is low (at 15-20% battery level). This is the reason why we say that the battery level affects other resources such as CPU and memory seriously, and is thus one of the important factors in mobile computing. Furthermore, the remaining battery life needs to be high enough to allow for additional demands, since a mobile participant may receive a sudden call or text message, making it difficult to provide cloud service consumers with sufficient resources. Regarding memory, the memory usage of mobile devices is quite stable, ranging from 70% to 90% (i.e., 10-30% free battery) regardless of the user's usage patterns; consequently, the available resource pool in this example contains the four mobile devices m_1, m_2, m_4, and m_5.

To determine the minimum clustering criteria for the addition of an available mobile resource to a specific cloudlet, we measure the devices' CPUs and memory usage every five seconds for a week under various conditions such as idle state, normal use, and video streaming or downloading, as shown in **Fig. 4**. We do not measure the battery usage of each

device because the device is changed into the power saving mode when the remaining battery falls below 15-20 percent. Most of mobile devices in the power saving mode are restricted of their resources. Therefore, we directly use this percentage (e.g. 15 or 20) as minimum clustering criteria.



Mobile Device	m_1	m_2	m_3	m_4	m_5	m_6
Availability Status	A	A	R	F	A	NA
CPU (%)	10	20	80	30	25	NA
Memory (%)	85	80	70	65	75	NA
Battery (%)	75	10	95	70	C	NA

A: Available/R: Running/F: Finished states
C: battery charging/NA: Unavailable

Fig. 3. Collection of received mobile resource statuses in the hypervisor under a variety of conditions

In this paper, we regard the status information from mobile devices' manufacturers as correct. To be specific, the remaining resource information such as CPU, memory, and battery, provided by native libraries on each mobile device, is correct unless the device is out of order. Therefore, if the given information proves incorrect, we consider the mobile device out of order. For example, we trust that the information (e.g. current time and today's date) provided by a mobile device is always correct with no doubt. Another example is that when we drive a car, we also implicitly believe that the gauge of the car displays a correct driving speed. In the resource monitoring module, we consider devices inoperable if their network status is unstable or disconnected. The reason is that we cannot reliably receive the status information about the mobile device periodically while processing a job. For these reasons, we establish the minimum clustering criteria for mobile cloudlet membership, as follows:

- Network availability = 1 (connected) or 0 (disconnected)
- Remaining battery > 15
- Remaining CPU > 30
- Remaining memory > 10

Using these constraints, we define the factors of both mobile availability and resource throughput.

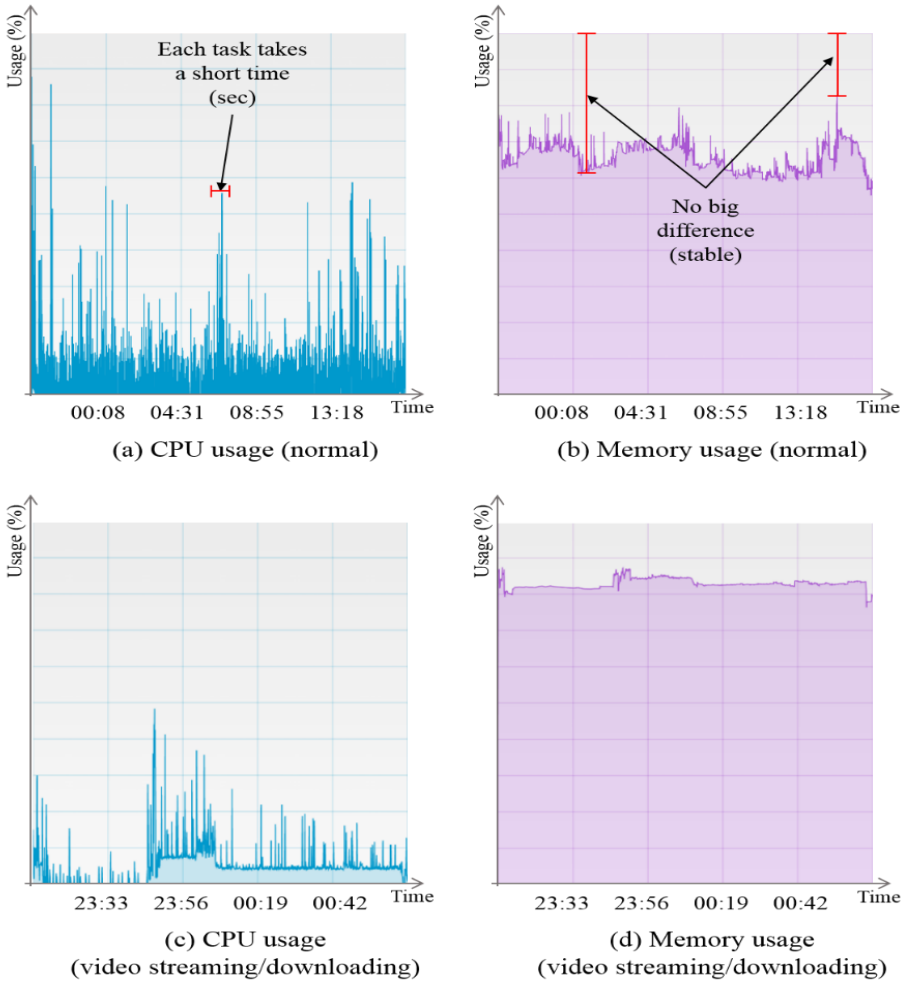


Fig. 4. Measurement of mobile device resources under a variety of conditions

Definition 1: Mobile availability. Mobile availability is given as a decimal value to indicate whether a mobile resource in a mobile cloudlet meets the minimum clustering criteria on demand or not. If one of the conditions does not meet one of the predefined minimum clustering criteria, the mobile availability is set to zero; otherwise, it is the product of all four values. It can be computed using the ternary conditional operator that is found in computer languages. A ternary conditional operator is defined as follows:

expression1 ? expression2 : expression3

- The first operand *expression1* is implicitly converted to its Boolean value (i.e., true or false).
- If *expression1* evaluates to true (1), the second operand *expression2* is evaluated; otherwise, the third operand *expression3* is evaluated.

In **Table 1**, for example, for the mobile resource m_3, the value is ((94 > 15) ? 94 : 0) × ((4 > 30) ? 4 : 0) × ((85 > 10) ? 85 : 0) = 94 × 0 × 85 = 0, indicating that the mobile resource does not meet one of the minimum clustering criteria (in this case, remaining CPU). Alternatively, for

the mobile resource m_4, the value is computed as ((29 > 15) ? 29 : 0) × ((59 > 30) ? 59 : 0) × ((35 > 10) ? 35 : 0) = 29 × 59 × 35 = 59,885, indicating the resource meets all of the minimum clustering criteria for being an available resource in the mobile cloudlet. If the value is greater than zero, the mobile resource can be a mobile resource provider in the resource pool; otherwise, the mobile resource is unavailable.

Table 1. Examples of collected statuses for each mobile resource

List (M) of resources in the cloud	Network availability	Remaining battery	Remaining CPU	Remaining memory	Mobile availability	Mobile resource throughput $\tau(r_i)$	$\|R\|$
m_1	-	-	-	-	-	-	
m_2	-	-	-	-	-	-	
m_3	1	94	4	85	-	-	
m_4	1	29	59	35	59,885	42.2	r_1
m_5	1	95	7	39	-	-	
m_6	1	36	25	88	-	-	
m_7	-	-	-	-	-	-	
m_8	-	-	-	-	-	-	
m_9	1	68	90	91	556,920	81.4	r_2
m_{10}	-	-	-	-	-	-	
m_{11}	1	58	98	38	215,992	70.0	r_3
m_{12}	-	-	-	-	-	-	
m_{13}	1	75	87	51	332,775	75.0	r_4
m_{14}	1	53	48	91	231,504	58.6	r_5
m_{15}	1	78	62	40	193,440	64.0	r_6
⋮							
m_{20}	1	92	44	92	372,416	72.8	r_8
Total						**525.8**	**8**

Definition 2: Resource throughput $\tau(r_i)$. Resource throughput $\tau(r_i)$ is the resource amount that can be provided by a mobile device r_i in a mobile cloudlet. The resource throughput $\tau(r_i)$ is computed according to the following equation:

$$\tau(r_i) = \boldsymbol{w}^T \cdot \boldsymbol{r}, \text{ where } \boldsymbol{r} = \begin{bmatrix} \text{Remaining battery(\%) of } r_i \\ \text{Remaining CPU(\%) of } r_i \\ \text{Remaining memory(\%) of } r_i \end{bmatrix} \quad (1)$$

where $\boldsymbol{w}^T$ is the transpose of a weighting vector for the set of constraints (remaining battery, remaining CPU, and remaining memory), and $\boldsymbol{r}$ is the resource status vector for the constraints. The weighting vector is determined by an observation-based heuristic approach. Based on the observations of usage of each resource in real mobile devices in **Fig. 4**, we can see that each usage variation of CPU and battery resources are relatively higher (more fluctuated) than that of memory resource. In determining weight values on each resource, we set more weight values on each CPU and battery compared to that of memory based on the empirical observations from **Fig. 4**. As this weight value changes, the number of available mobile resources is determined, and it can influence the entire scheduling function.

We use the weight vector to represent the characteristics of problems. To be specific, by adjusting the weight vector, we differentiate the problems that require resources with high

throughput of CPU from other resources with high throughput of memory. Since we focus on the problems with high throughput of CPU resources, we add more weights on CPU resources in this paper. We also add more weights on battery resources because the performance of CPU resources closely depends on the remaining battery power. In addition, by putting more weight vectors, we can collect available resource information from each mobile device by changing three different values (CPU, memory, and battery) of each mobile into one scalar value. These values of each mobile device are used to order mobile devices by its performance in the mobile resource monitor. With these values, the proposed system can remove the extreme or abnormal mobile resources such as a mobile device with high throughput of CPU (or memory) and very low remaining battery. As mentioned in previous section, because the battery level affects other resources seriously, it is an important factor in mobile computing. Since the goal is to solve problems that require many repetitive computations by using CPUs, we thus set higher weight values on CPU and battery than that of memory in experiments. Each value of weighted vector can be changed, depending on the problems to be solved.

In this paper, we thus set the weights for the remaining battery life and remaining CPU to 0.4, which are twice the value of the remaining memory weight. This is because the memory usage is much more stable than usage of the other constraints, regardless of a user's usage pattern. This indicates that memory usage is less important than the other factors in the processing of a job. These two vectors are then multiplied together to indicate the amount of available mobile resource in a range from 0 to 100; for instance, in **Table 1**, for mobile resource m_4, the resource throughput is computed as the following equation (2):

$$\tau(r_i) = \boldsymbol{w}^T \cdot \boldsymbol{r} = \begin{bmatrix} 0.4 \\ 0.4 \\ 0.2 \end{bmatrix}^T \cdot \begin{bmatrix} 29 \\ 59 \\ 35 \end{bmatrix} = 42.2 \tag{2}$$

In the same manner, we collect all of the statuses of the registered mobile resources and added them to the mobile resource pool so that $R = \{r_1, r_2, \cdots, r_x, \cdots, r_m\}$, thereby enabling us to easily monitor the status of the mobile resources in a cloud. From the example data in **Table 1**, the total number of available mobile resources and their total throughput for the mobile cloudlet are 8 and 525.8, respectively. We then divide the possible states of the mobile participants (resources) into the following: available, running, and finished. First, a mobile device in an available state refers to the device that meets the minimum clustering criteria for participating in job processing through the initial mobile resource monitor. Secondly, the mobile device moves to a running state to process a received task. Lastly, a mobile device in a finished state refers to the device in which a job handler is ready to reallocate an unprocessed task such as unexceptional situations (e.g. task miss or task fail). The job handler does not have to monitor available mobile devices again after a task has been allocated and successfully processed by the mobile resource. It enables the job handler to reallocate the task directly by reducing unnecessary steps such as monitoring of new mobile resources in the mobile resource monitor module and job scheduling in the resource handler.

4.2 Job handler

When a mobile cloud customer submits a job to the proposed system (mobile cloud service provider), the job handler partitions the job into several tasks (job partitioning) and creates the mobile cloudlet (cloudlet creation). The job handler then allocates each task to the appropriate mobile resource (mobile resource allocation) using the following assumptions:

- A job can be partitioned into several tasks, and the tasks are independent of each other.

- A brief duration is required to process each task, but the number of tasks is very large.
- We already know the amount of resources that each tasks needs.
- No more than two tasks can be allocated to one resource to guarantee the intrinsic functionalities of the mobile devices.
- When a job failure occurs due to an unknown resource problem, we reallocate the task rather than trying to model incoming or outgoing mobile resources in the mobile cloudlet.

For example, a factorial computation problem can be separated into sub-factorial problems, and a matrix multiplication problem can be composed of a set of small multiplication problems with respect to its row and column indices. In the job partition step, a set of jobs is represented by $J = \{j_1, j_2, \cdots, j_i, \cdots, j_n\}$. Each job j_i is composed of k tasks: $S(j_i) = \{s_1, s_2, \cdots, s_j, \cdots, s_k\}$. Based on the following criteria, each task s_j in i-th job j_i is pushed into one of two queues, as shown in **Fig. 5**: The waiting task queue is in charge of abnormally processed or unprocessed tasks, and tasks with a throughput higher than that of a mobile resource. The priority task queue is responsible for normal tasks that should be processed.

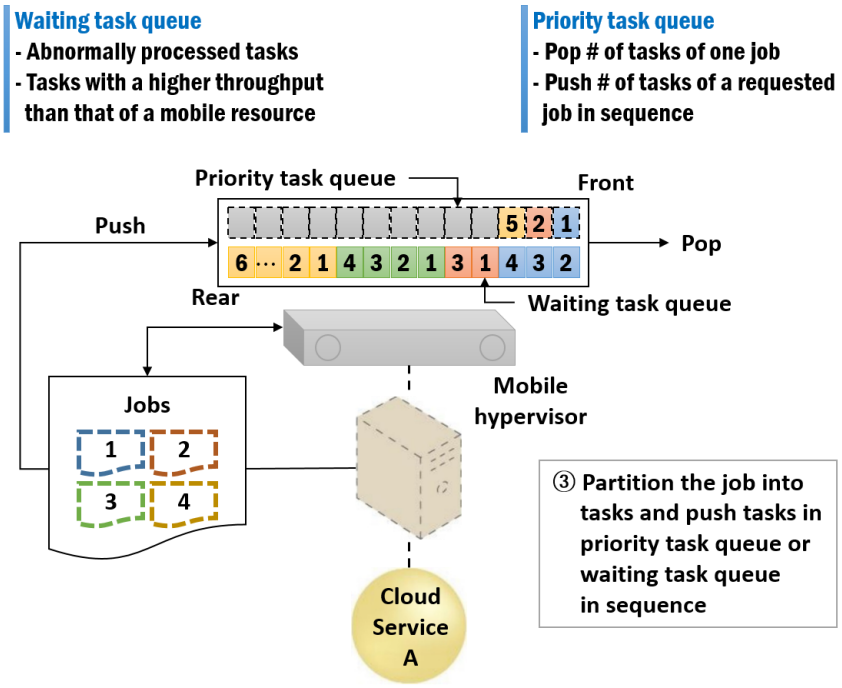


Fig. 5. Partitioning of a submitted job into several tasks in the job handler

Table 2. Example of a job submitted to the job handler

Tasks $S(j_1)$ in job j_1	1	2	3	4	5	6	7	8	9	⋯	30	**Total**
Required throughput	27	34	21	28	15	65	4	92	77	⋯	56	**1456.0**

Tasks are pushed into or popped from the queue(s), and a mobile cloudlet is formed for each job (after the processing is finished for a job, this mobile cloudlet is eliminated). Next, a job handler takes charge of the allocation of tasks to each service participant based on the list of available resources in the mobile resource monitor, as shown in **Fig. 6**. The job handler decides which mobile participant is appropriate for each task and processes the task based on the information from the mobile resource monitor at the current moment, as shown in **Fig. 6**. The job handler does not assign a new task to the participants if the predefined resource usage

limit is exceeded, since a user may abruptly use his or her mobile device for other purposes, such as calling, messaging, or gaming. This allows the cloud service provider to guarantee the basic functionalities of mobile devices for its users. The resource handler therefore has a predefined policy for the appropriate allocation of the requested jobs from the job handler; in **Fig. 5**, for instance, the first four tasks are allocated to four mobile participants in the mobile cloudlet. When job j_1 is submitted to the job handler, as shown in **Table 2**, the job handler allocates each task to an appropriate mobile resource(s). From **Table 1**, the available resource information from the mobile resource monitor is already known. From **Table 2**, the total number of tasks and their total required throughput are 30 and 1456.00, respectively.

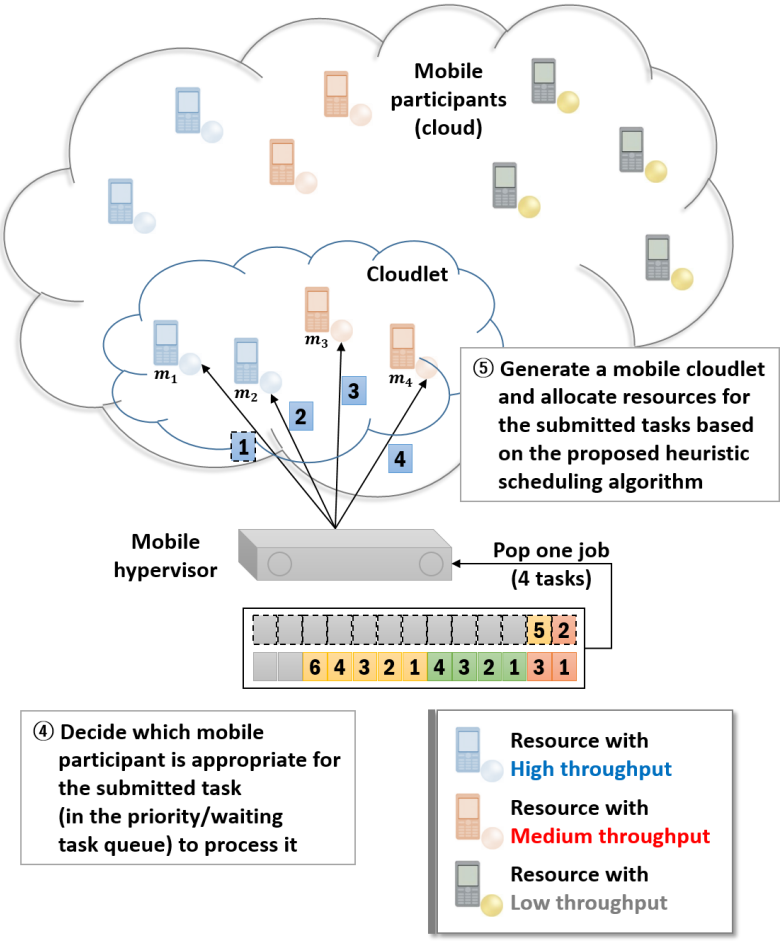


Fig. 6. Cloudlet creation and resource allocation workflow in the job handler

Since the required mobile resource throughput for a given task may be larger than the available mobile resource throughput of a mobile device, not all of the mobile resources are suitable for processing the given task. In this case, the processing time for the given task is longer than those of other cases (i.e., we assume that a mobile resource can process a task with a required throughput that is higher than the available throughput of the resource). For example, if the required throughput of the given task is 100 and the throughput of a mobile resource is 50 (e.g. the task 8), it will take twice as long for the mobile resource to process the task. We then construct a scheduling algorithm to allocate each task to an appropriate mobile resource in the mobile cloudlet. To construct the job scheduling algorithm, we formulate the

objective function for the efficient allocation of jobs under the monitored mobile resources. The following equation and inequalities express the objective function to minimize the total processing time with constraints:

$$\begin{aligned} &\min\left(\max_i \sum_{i=1}^{\delta} F(s_j, r_i)\right) = \min\left(\max_i \sum_{i=1}^{\delta} \frac{\tau(s_j)}{\tau(r_i)}\right), \\ &\text{subject to} \quad \sum_{j=1}^{|S(j_i)|} \tau(s_j) \le \delta \cdot \sum_{i=1}^{|R|} \tau(r_i), \\ &\qquad\qquad \forall i, \tau(r_i) > 0, \\ &\qquad\qquad \text{and } \forall j, \tau(s_j) > 0 \end{aligned} \tag{3}$$

where the performance value $F(s_j, r_i)$ indicates that the j-th task s_j is assigned to the i-th resource r_i. $F(s_j, r_i)$ is computed by dividing the required throughput $\tau(s_j)$ by the available resource throughput $\tau(r_i)$. The constant value δ is the minimum number of iterations for the given job and the mobile resources, and is computed by $\left\lceil \frac{|S(j_i)|}{|R|} \right\rceil$ (e.g. $\delta = \left\lceil \frac{|S(j_i)|}{|R|} \right\rceil = \lceil 30/8 \rceil = 4$ iterations in the cases of **Table 1** and **Table 2**). The index j is determined by the performance value and is based on the proposed job scheduling algorithm in **Table 3**.

Table 3. Proposed job scheduling algorithm

Algorithm (Job scheduling)	
Input	A set of mobile resources R, a set of tasks $S(j_i)$ for the i-th job j_i
Output	$\min\left(\max_i \sum_{i=1}^{\delta} F(s_j, r_i)\right)$ // total job processing time
Method	

```
01: sumOfTotalCost = 0;
02: compute δ = ⌈|S(j_i)|/|R|⌉;
03: construct a |R| × |S(j_i)| matrix O and fill in F(s_j, r_i) values;
04: numOfTasks = |S(j_i)|;
05: while (δ > 0)
06:     numOfAvailableR = |R|;
07:     maxF = max_j F(s_j, r_i);
08:     sumOfTotalCost += maxF;
09:     while (numOfAvailableR > 0 && numOfTasks > 0)
10:         find max_j F(s_j, r_i) in matrix O, where r_i is not in use and F(s_j, r_i) ≤ maxF;
11:         assign a task s_j to a resource r_i;
12:         mark '0' on all elements in j-th row;
13:         numOfAvailableR--;
14:         numOfTasks--;
15:     end while
16:     δ--;
17: end while
18: return sumOfTotalCost;
```

In **Table 3**, we use a set of mobile resources R and a set of tasks in the i-th job j_i $S(j_i)$ as input parameters, and a total job processing time is given as an output; for example, R and $S(j_i)$ can be represented by **Table 1** and **Table 2**. In line 2, δ is computed by $\left\lceil \frac{|S(j_i)|}{|R|} \right\rceil = 4$, indicating that each resource can be allocated a maximum of four tasks. The algorithm then constructs a $|R| \times |S(j_i)|$ matrix O whose elements are composed of the throughput values $F(s_j, r_i)$ with respect to the j-th task and i-th mobile resource, as shown in **Table 4**. Next, the

algorithm finds the $\max_j F(s_j, r_i)$ for each task s_j, because the task s_j requires the longest processing time even though the task is allocated to the mobile resource r_i that has the highest throughput (e.g., resource r_2 in **Table 4**). We regard the processing time as 1 when the requested resource is equal to the throughput of the mobile resource. For example, if mobile device r_4 processes task s_8 in **Table 4**, the processing time is 1.227 because the throughput of mobile device r_4 is 75 which is less than the required throughput of the given task s_8. In the proposed method, the mobile resource r_i with the highest throughput processes the most time-consuming task to minimize each iteration time. Since several mobile resources work in parallel on the given tasks, the most time-consuming task should be processed in as short a time as possible. As the time that a task consumes increases, the overall performance of the system is reduced more drastically, because other processed tasks must wait for the task to be processed. For this reason, the algorithm chooses another task and a suitable resource with a value very close to, but not greater than *maxF* in the next sub-while loop (line 10). From line 11 to line 15, the algorithm repeats in the same manner until all of the given resources are allocated to tasks, representing one iteration of the overall workflow. The iteration is performed until all of the tasks are assigned to mobile resources. For example, in **Table 4**, the eight tasks $s_{27}, s_{26}, s_{15}, s_8, s_{19}, s_{18}, s_{29}$, and s_{20} are simultaneously assigned to appropriate mobile resources (marked as yellow-colored cells in **Table 4**) at the first iteration, and it is likely that the processing times for all of them are very similar, ranging from 1.161 to 1.229.

Table 4. Example of constructed matrix and the resources allocated at the first iteration

	Resource	$\max_j F(s_j, r_i)$	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
Task	$\tau(r_i)$ / $\tau(s_i)$	-	42.2	81.4	70.0	75.0	58.6	64.0	61.8	72.8
s_1	27	0.332	0.640	0.332	0.386	0.360	0.461	0.422	0.437	0.371
s_2	34	0.418	0.806	0.418	0.486	0.453	0.580	0.531	0.550	0.467
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
s_8	92	1.130	2.180	1.130	1.314	1.227	1.570	1.438	1.489	1.264
s_9	77	0.946	1.825	0.946	1.100	1.027	1.314	1.203	1.246	1.058
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
s_{14}	85	1.044	2.014	1.044	1.214	1.133	1.451	1.328	1.375	1.168
s_{15}	72	0.885	1.706	0.885	1.029	0.960	1.229	1.125	1.165	0.989
s_{16}	4	0.049	0.095	0.049	0.057	0.053	0.068	0.063	0.065	0.055
s_{17}	79	0.971	1.872	0.971	1.129	1.053	1.348	1.234	1.278	1.085
s_{18}	75	0.921	1.777	0.921	1.071	1.000	1.280	1.172	1.214	1.030
s_{19}	78	0.958	1.848	0.958	1.114	1.040	1.331	1.219	1.262	1.071
s_{20}	49	0.602	1.161	0.602	0.700	0.653	0.836	0.766	0.793	0.673
s_{21}	28	0.344	0.664	0.344	0.400	0.373	0.478	0.438	0.453	0.385
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
s_{25}	7	0.086	0.166	0.086	0.100	0.093	0.119	0.109	0.113	0.096
s_{26}	86	1.057	2.038	1.057	1.229	1.147	1.468	1.344	1.392	1.181
s_{27}	100	1.229	2.370	1.229	1.429	1.333	1.706	1.563	1.618	1.374
s_{28}	8	0.098	0.190	0.098	0.114	0.107	0.137	0.125	0.129	0.110
s_{29}	86	1.057	2.038	1.057	1.229	1.147	1.468	1.344	1.392	1.181
s_{30}	56	0.688	1.327	0.688	0.800	0.747	0.956	0.875	0.906	0.769

Table 5 shows the final task allocation results after four iterations have been performed. In **Table 5**, the yellow, green, blue, and gray cells represent the allocation results for the first,

second, third, and fourth iterations, respectively. After all of the tasks have been assigned to mobile resources, we compute the total job-processing time by adding together the longest times that were consumed by each iteration. In the example, the total job processing time is $\max_i \sum_{i=1}^{\delta} \frac{\tau(s_j)}{\tau(r_i)} = 1.229 + 1.192 + 1.155 + 0.160 = 3.736$. The proposed algorithm takes 1.349 times as long as the expected optimal time $\left(\sum_j \tau(s_j)/\sum_i \tau(r_i) = 2.769\right)$.

Table 5. Final results of task allocation

	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
s_1							0.437	
s_2	0.806							
s_3			0.300					
s_4	0.664							
s_5				0.200				
s_6					1.109			
s_7	0.095							
s_8				1.227				
s_9				1.027				
s_{10}		1.155						
s_{11}			0.029					
s_{12}						0.344		
s_{13}						0.516		
s_{14}								1.168
s_{15}					1.229			
s_{16}						0.063		
s_{17}			1.129					
s_{18}							1.214	
s_{19}						1.219		
s_{20}	1.161							
s_{21}					0.478			
s_{22}		0.160						
s_{23}								0.261
s_{24}		1.192						
s_{25}							0.113	
s_{26}			1.229					
s_{27}		1.229						
s_{28}					0.137			
s_{29}								1.181
s_{30}							0.906	

From the theoretical viewpoint, an optimal solution is possible for given tasks and resources. However, in reality, it is not easy to find an optimal solution in a limited period since the amount of mobile resources varies greatly and each task requires a different amount of resource throughputs. For example, let the available resource from a mobile device be *x*. If the situation requires an exact *x* in a random task, an optimal job distribution is possible in such a circumstance. However, in reality, a possibility always exists for surplus resources being occurred since a majority of tasks requires a different amount of resources from that of *x*. In theory, it is ideal to maintain zero status for surplus resources when processing each task.

However, it is not easy to find an optimal solution for resource allocations and job scheduling in a reasonable time; sometimes, we cannot find the optimal solution at all. The reason is that most of cases happen under the condition that the required throughputs of each task are not equivalent to the available resource throughputs of each mobile. In real cases, the throughput of required resources is not always equal to that of available mobile resources. When the optimal method processes a job, it always needs at least more than one iteration. During the time of iterations, the optimal method should maximize the usage of mobile resources. Thus, it minimizes the total processing time. In this paper, we consider this case as the optimal one, and its performance can be computed by $\sum_j \tau(s_j)/\sum_i \tau(r_i)$. For example, if the amount of required resources for the given job is 100 and that of available mobile resources is 20, then the number of iterations and the maximum throughput of mobile resources in each iteration are 5 and 20, respectively. Therefore, we can compute the performance of optimal method as $\sum_j \tau(s_j)/\sum_i \tau(r_i) = 100/20 = 5$.

4.3 Resource handler

The resource handler is responsible for rerouting the current mobile participant to another mobile resource to handle unexpected situations, as demonstrated in **Fig. 7**.

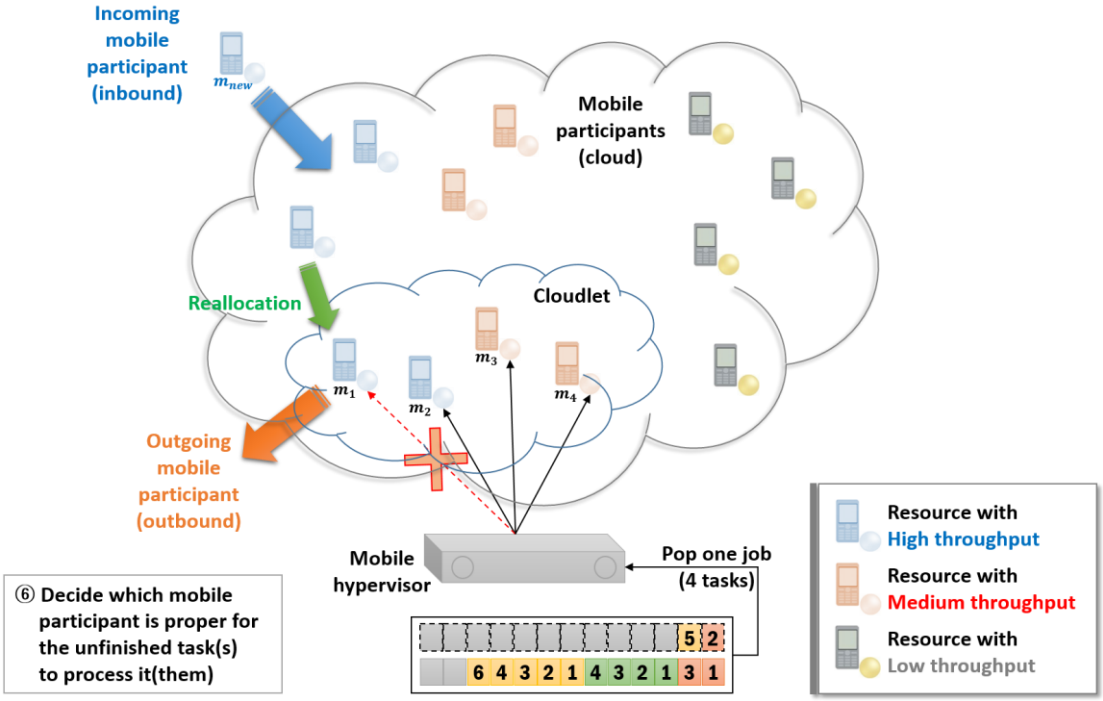


Fig. 7. Example of an unexpected occurrence situation in the cloud

The resource handler prevents job processing failures or delays that are caused by a resource issue. It notifies the mobile resource monitor and the job handler when the requested job is not properly processed for unknown reasons. For instance, in the processing of a task or a job, the processing time can be longer than the expected time for unknown reasons, such as connection loss, the reaching of a resource's throughput limit (CPU, memory, and/or battery), or a delay of the notification messages between the results consolidator and the participants in the mobile cloudlet. In addition, when mobile participants leave the range of the cloud or they are

disconnected due to a breakdown in wireless communications, the capacity of the cloud's resource availability can be dramatically decreased. To address the issue of fault tolerance and to prevent the worst possible cases, we are therefore suggesting a model for the calculation of the minimum threshold of the acceptable resource power for a mobile cloud in the mobile resource monitor. In that case, the task is rerouted to other mobile resources based on the proposed job scheduling algorithm. However, we do not try to model the event wherein a mobile resource moves into or out of range. This is because we assume that all of the mobile participants that are entering or leaving the cloudlet (or networks) follow a Poisson distribution [16, 17], and the probability that these cases will occur is quite low and they are therefore negligible (e.g., 14% for $|R| = 8$). To deal with an unexpected situation, the resource handler checks for the existence of a mobile participant that has already finished its requested task. If so, the resource handler reallocates the unfinished or unprocessed task to the other mobile resource. Otherwise, the resource handler asks the mobile resource monitor whether another resource exists and, in the case of an affirmative response, reallocates the task to it.

4.4 Results consolidator

The processed results from the job handler (or resource handler) are collected and combined in the results consolidator. Whenever a task in the mobile participant is successfully processed within the predefined time limit, the mobile participant sends a notification message to the results consolidator. Otherwise, the results consolidator sends a notification message to the resource handler. Lastly, the finished tasks are combined into one final job in the results consolidator that is then delivered to the cloud service consumer.

5. Performance Study

5.1 Experimental setup

For the experiments, we created randomly generated data including the throughputs of the required resources for each task and those of the available mobile resources. We then implemented the proposed method by using the Python programming language to compute the sum of the $\max_j F(s_j, r_i)$ values that were extracted from each iteration with $|R| < |S(j_i)|$, where $|R| = [10, 20, 30, \ldots, 980, 990]$ and $|S(j_i)| = [100, 300, 500, 1000]$. Next, we compared the sum of the $\max_j F(s_j, r_i)$ values with and without the suggested method. Lastly, we demonstrated that the proposed method outperforms the existing method, and analyzed the experiment results including a comparison with the optimal solution. All of the experiments were conducted on a computer with eight Intel® Core™ i7-3770 CPUs @ 3.40 GHz and 16 GB RAM, running Windows 7. In the experiments, we assumed that each mobile device has only one virtual machine, which means that each device is capable of processing only one task at a time (per iteration). The reason is an ability to guarantee the basic mobile functionalities of mobile devices such as call, text messaging, and Web browsing, thereby allowing users to use their mobile devices for other purposes and minimizing any corresponding inconvenience.

5.2 Results

In the experiment results, as the value of the sum of $\max_j F(s_j, r_i)$ approaches 0, the associated job is processed more rapidly in the mobile cloudlet; alternatively, as the value of the sum of $\max_j F(s_j, r_i)$ increases, the processing time for the given job in the mobile cloudlet is increased. **Fig. 8** shows comparisons of the experiment results of the proposed

method with those of other methods such as First-In-First-Out (FIFO), random, and the optimal method.

According to the results of the experiment in **Fig. 8**, the proposed method outperforms two conventional methods, in that it achieves its purpose of resource allocation and the calculation of the expected optimal time $\left(\sum_j \tau(s_j)/\sum_i \tau(r_i)\right)$ for the given jobs. As shown in **Fig. 8**, the performance of our newly proposed method is very close to the optimal one with comparisons to the conventional scheduling algorithms. Furthermore, the proposed method has a low time complexity (less than 1 second per iteration) because the method is based on heuristic approach, and thus uses small size of matrix computations to allocate mobile resources in a mobile cloudlet. A further examination of comparison of results confirms this favorable performance, whereby a great difference was not found between the expected time consumed and the expected optimal time, even though the number of mobile resources or tasks for a given job increases.

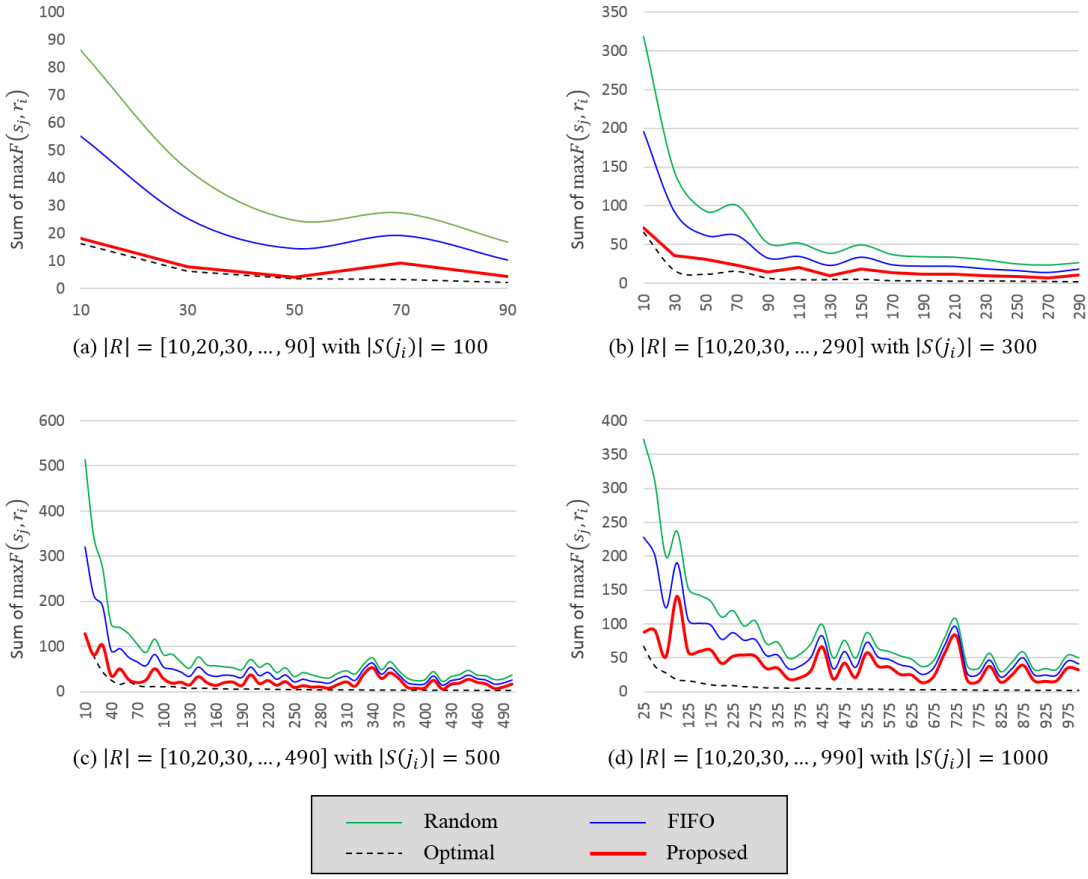


Fig. 8. Results of proposed method and other methods (random, FIFO, optimal)

6. Conclusion

In this paper, we proposed an efficient method of cooperative mobile resource sharing for the monitoring of mobile devices on demand and the scheduling of requested tasks across the mobile resources in mobile cloud environments. The proposed method is composed of the following four components: mobile resource monitor, job handler, resource handler, and results consolidator. To validate the proposed method, we conducted experiments and

demonstrated that the method performs mobile resource sharing for cooperative computation in mobile cloudlets, while taking into account both the intrinsic properties and the number of mobile devices. The experiment results show that the mobile resources in a mobile cloudlet can play a significant role in the processing of a job that consists of many independent tasks. We have therefore shown that the method is feasible in terms of the attainment of a cooperative computational power. In future works, we will examine how an idle mobile resource in a mobile cloudlet can be reallocated to another job in a parallel fashion. Moreover, we will conduct research to address a case wherein mobile devices enter and leave the mobile cloudlet. We will also improve the proposed job scheduling method to deal with much more complicated cases in dynamic cloud computing environments.

References

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," *National Institute of Standards and Technology*, 2011. Article (CrossRef Link)

[2] S. Ried, H. Kisker, P. Matzke, A. Bartels, and M. Lisserman, "Sizing The Cloud–Understanding And Quantifying The Future Of Cloud Computing," *Forrester Research, Cambridge*, MA, 2011. Article (CrossRef Link)

[3] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. Ullah Khan, "The rise of "big data" on cloud computing: Review and open research issues," *Information Systems,* vol. 47, pp. 98-115, 2015. Article (CrossRef Link)

[4] G. Aceto, A. Botta, W. D. Donato, and A. Pescapè, "Cloud monitoring: A survey," *Computer Networks*, vol. 57, pp. 2093-2115, 2013. Article (CrossRef Link)

[5] H. Lee, Y.-S. Jeong, and H. Jang, "Performance analysis based resource allocation for green cloud computing," *The Journal of Supercomputing,* vol. 69, pp. 1013-1026, 2014. Article (CrossRef Link)

[6] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, pp. 1587-1611, 2013. Article (CrossRef Link)

[7] L. Fangming, S. Peng, J. Hai, D. Linjie, Y. Jie, N. Di, *et al.*, "Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications," *Wireless Communications, IEEE,* vol. 20, pp. 14-22, 2013. Article (CrossRef Link)

[8] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, and R. Buyya, "Cloud-Based Augmentation for Mobile Devices: Motivation, Taxonomies, and Open Challenges," *Communications Surveys & Tutorials, IEEE,* vol. 16, pp. 337-368, 2014. Article (CrossRef Link)

[9] O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, "The Rise of RaaS: The Resource-as-a-Service Cloud," *Communications of the ACM*, vol. 57, pp. 76-84, 2014. Article (CrossRef Link)

[10] M. Jo, T. Maksymyuk, B. Strykhalyuk, and C.-H. Cho, "Device-to-device-based heterogeneous radio access network architecture for mobile cloud computing," *Wireless Communications, IEEE,* vol. 22, pp. 50-58, 2015. Article (CrossRef Link)

[11] D. Huang, T. Xing, and H. Wu, "Mobile cloud computing service models: a user-centric approach," *Network, IEEE*, vol. 27, pp. 6-11, 2013. Article (CrossRef Link)

[12] Y. Li and W. Wang, "Can Mobile Cloudlets Support Mobile Applications?," in *Proc. of IEEE INFOCOM*, 2014. Article (CrossRef Link)

[13] A. Khalifa and M. Eltoweissy, "Collaborative autonomic resource management system for mobile cloud computing," in *Proc. of CLOUD COMPUTING 2013, The Fourth International Conference on Cloud Computing, GRIDs, and Virtualization*, pp. 115-121, 2013. Article (CrossRef Link)

[14] D. Rohr, "ALICE TPC online tracker on GPUs for heavy-ion events," in *Proc. of Cellular Nanoscale Networks and Their Applications (CNNA), 2012 13th International Workshop on*, pp. 1-6, 2012. Article (CrossRef Link)

[15] D. Rohr, S. Gorbunov, A. Szostak, M. Kretz, T. Kollegger, T. Breitner, *et al.*, "ALICE HLT TPC Tracking of Pb-Pb Events on GPUs," *Journal of Physics: Conference Series*, vol. 396, p. 012044, 2012. Article (CrossRef Link)
[16] I. Psaras and L. Mamatas, "On demand connectivity sharing: Queuing management and load balancing for user-provided networks," *Computer Networks*, vol. 55, pp. 399-414, 2011. Article (CrossRef Link)
[17] L. Massoulie and J. W. Roberts, "Bandwidth sharing and admission control for elastic traffic," *Telecommunication systems*, vol. 15, pp. 185-201, 2000. Article (CrossRef Link)

Kyunglag Kwon received the B.S. and M.S. degrees in Computer and Information Science from Korea University, Korea, in 2008 and 2010, respectively. He is a Ph.D. student at Korea University, Korea. Recently, he is listed as a research scientist in the 2016 33rd edition of "Marquis Who's Who in the World" for his work in the field of computer science. His research interests include soft computing, sentic computing, artificial intelligence, intelligent information systems, social web, and data mining.

Hansaem Park received a B.S. degree in Computer and Information Science from Korea University, Korea, in 2014. He is a Master student at Korea University, Korea. His research interests include social web, data mining, and content recommendation system.

Sungwoo Jung is a B.S. student in Computer and Information Science from Korea University, Korea. His research interests include collective intelligence, data mining, and content recommendation.

Jeungmin Lee received a B.S. degree in Computer and Information Science from Korea University, Korea, in 2014. He is a Master student at Korea University, Korea. His research interests include social network analysis, and social semantic web.

In-Jeong Chung earned a B.S. degree from Seoul National University, Korea in 1978, a M.S. degree in Computer Science from Korean Advanced Institute of Science and Technology (KAIST), Korea in 1980, and a Ph.D. degree from University of Iowa, USA in 1989. He is a professor in the Department of Computer and Information Science, and Founding Director of the Intelligent Information Systems Laboratory at Korea University, South Korea from 1990. Recently he registered on Marquis, Who's Who in Science and Engineering (2008-2009). His research interests include semantic web, intelligent web service, information retrieval, data mining, decision support system, agent, and expert system.