sensors

MDPI

Article

A General Cross-Layer Cloud Scheduling Framework for Multiple IoT Computer Tasks

Guanlin Wu [1,*], Weidong Bao [1], Xiaomin Zhu [1,2] and Xiongtao Zhang [1]

1 College of Systems Engineering, National University of Defense Technology, Changsha 410073, China; wdbao@nudt.edu.cn (W.B.); xmzhu@nudt.edu.cn (X.Z.); zhangxiongtao14@nudt.edu.cn (X.Z.)
2 State Key Laboratory of High Performance Computing, National University of Defense Technology, Changsha 410073, China
* Correspondence: wuguanlin16@nudt.edu.cn

Received: 12 April 2018; Accepted: 17 May 2018; Published: 23 May 2018

check for updates

Abstract: The diversity of IoT services and applications brings enormous challenges to improving the performance of multiple computer tasks' scheduling in cross-layer cloud computing systems. Unfortunately, the commonly-employed frameworks fail to adapt to the new patterns on the cross-layer cloud. To solve this issue, we design a new computer task scheduling framework for multiple IoT services in cross-layer cloud computing systems. Specifically, we first analyze the features of the cross-layer cloud and computer tasks. Then, we design the scheduling framework based on the analysis and present detailed models to illustrate the procedures of using the framework. With the proposed framework, the IoT services deployed in cross-layer cloud computing systems can dynamically select suitable algorithms and use resources more effectively to finish computer tasks with different objectives. Finally, the algorithms are given based on the framework, and extensive experiments are also given to validate its effectiveness, as well as its superiority.

Keywords: IoT services; cross-layer cloud computing; general scheduling framework; computer task; specific scheduling models and algorithms

---

1. Introduction

Recently, cross-layer cloud computing systems, as shown in Figure 1, have become more and more important in people's daily lives, particularly with the development of various computing technologies. The cross-layer cloud computing system is a cloud computing environment that uses a mix of the local edge computing or the fog computing cloud, the remote cloud data center and other types of cloud platforms to support the device cluster. Edge computing or fog computing is usually deployed near the users and is in charge of delay-sensitive tasks and basic data pre-processing [1]. The remote cloud data center is usually responsible for computation-intensive tasks [2]. In this way, organizations are able to provide computer services with orchestration among these different platforms based on their requirements.

However, to meet the growing multiple computer demand of IoT services, the diversity of these multiple computer tasks has become an important feature of the cross-layer cloud computing system, and it makes achieving great performance more complex. On the one hand, these multiple computer tasks in the cross-layer cloud computing system differ not only in task type, but also in task scheduling objective. For instance, a dependent task is one of the most common types of computer tasks in IoT services. It can be decomposed into different tasks based on logical sequences in the calculation process, and researchers can obtain these important conclusions faster [3]. On the contrary, there is also a type of task called an independent task. Besides, long-running tasks in IoT services need fault-tolerant mechanisms and more time to reduce the expense of high rollback operations when failures occur, but real-time deadline tasks that are less sensitive to delay need less time [4]. On the other hand, different tasks in IoT services may also have common performance optimization goals, such as energy saving and system reliability. Sometimes, different tasks will select some common scheduling objectives according to their actual demands [5,6]. This makes the required algorithms comparable, and we must take this into account.

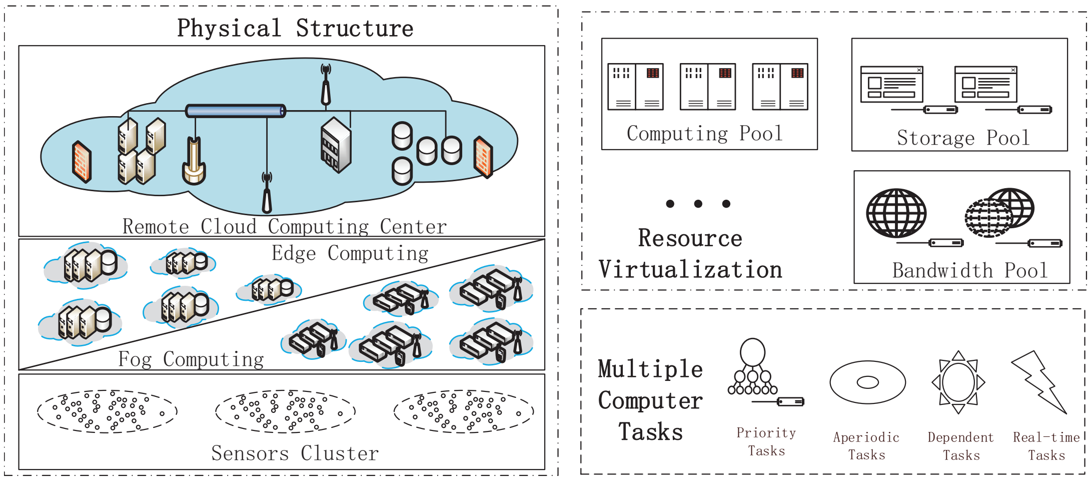


Figure 1. A schematic diagram of a cross-layer cloud computing system.

The diversity of IoT services and applications brings enormous challenges to improving the performance of multiple computer tasks' scheduling in cross-layer cloud computing systems. Unfortunately, the commonly-employed frameworks fail adapt to the new patterns on the cross-layer cloud. Firstly, the conventional computer task scheduling frameworks do not have the ability to self-adaptively adjust. The conventional frameworks are always deployed as a monolithic system component. In this way, it is hard to adjust the optimization goals and select the most appropriate algorithm after the parameters have been set. Secondly, it is cumbersome and fallible when adding new scheduling capability in conventional computer task scheduling frameworks for cross-layer cloud computing systems. That is because there are many computer task scheduling strategies designed with different optimization objectives that share some common functional components and rely on a similar software engineering framework for implementation. Mutual influence is a very complicated phenomenon in the conventional computer task scheduling framework. Finally, the cross-layer cloud computing system and a single cloud are different in many ways. The characteristics of the cross-layer cloud computing system would require more thought.

In this study, we propose a new computer task scheduling framework for multiple IoT services in the cross-layer cloud computing system with the theory of the software engineering framework to deal with the aforementioned challenges. We first analyze the features of the cross-layer cloud and multiple computer tasks. Then, we design the scheduling framework based on the analysis and present detailed models to illustrate the procedures of using the framework. With the proposed framework, the IoT services deployed in cross-layer cloud computing systems can dynamically select suitable algorithms and use resources more effectively to finish computer tasks with different objectives. Finally, the algorithms are given based on the framework and the typical examples of different computer tasks. Extensive experiments are also given to validate its effectiveness, as well as its superiority. The main contributions of the study are as follows:

- We analyze the features of the cross-layer cloud and computer tasks. The theoretical analysis illustrates that these components influence the resource management in the cross-layer cloud computing system directly, and they are the most basic contents of the computer task scheduling framework in the cross-layer cloud computing system.

- We design a general computer task scheduling framework for the cross-layer cloud computing system. With the scheduling framework, the computer tasks in cross-layer cloud computing systems can dynamically select suitable algorithms and use resources more effectively to solve these tasks with different restrictions of the objectives.
- We present specific models and examples to illustrate the detailed procedures used, and the corresponding algorithms are designed as a demonstration based on these theories and models. These can prove that the framework is actually executable in cross-layer cloud computing systems, and the scheduling objectives are able to be dynamically matched to different types of tasks.
- We evaluate the framework with extensive experiments. These two examples represent different typical scenarios, and the experimental results prove its effectiveness, as well as its superiority.

The remaining part of the paper is divided into six sections. Relevant works are briefly reviewed in Section 2. The features of the cross-layer cloud and multiple computer task and the proposed framework are presented in Section 3. Section 4 presents specific scheduling models. Section 5 gives the detail algorithms that are contained in the algorithm pool. Section 6 presents the evaluation results of the experiments. Section 7 concludes our study.

2. Related Work

Under the background of cross-layer cloud computing, researchers have proposed many computer task scheduling approaches. The works related to the cross-layer cloud that are helpful to our study will be evaluated in this section.

With the rapid development and application of the cross-layer cloud in actual production, some scholars have carried out some research on the cloud computer task scheduling framework. Ergu et al. [7] divided tasks into different task pools and then constructed comparison matrices according to the task's network bandwidth requirements, completion time, computational cost and task credibility. By the method, multiple computer tasks' scheduling can meet the requirements of multi-objective tasks. However, when the number of tasks is large, the comparison matrix is larger and the solution speed is slower. Tang et al. [8] proposed and analyzed a two-sided bidding mechanism allowing each demander to bid to select a demand resource-price function and each supplier to bid to select a supply resource-price function. Zhu et al. [9] proposed a universal framework to schedule tasks and provide resources in the cloud computing systems to improve system performance. It sets a good example for our work to deal with the challenge in the cross-layer cloud. Wu et al. [10] proposed a collaborative multiple computer task scheduling architecture, in which heterogeneous information of nodes was considered and integrated to make decisions. Sun et al. [11] proposed a real-time and energy-efficient multiple computer task scheduling and optimization framework. Rao et al. [12] regarded computer task scheduling as a distributed learning task and proposed an algorithm to efficiently express learning experience as the core of the VM-side learning engine. In this mechanism, each VM or instance can make any resource request based on its own needs, and the request can be evaluated and automatically responded to. However, these methods use the unchangeable task pool to process batch tasks that are not suitable for the task that dynamically joins the situation. In addition, these frameworks also have the main drawback that the computational complexity is high and is mainly optimized for isomorphic resources. When the size of resources increases or heterogeneous resources are used, the performance drops rapidly. This is exactly what the cross-layer cloud often faces.

There are also some works that study the specific algorithms of computer task scheduling in the cloud environment. Farahabady et al. [13] explored how resources in the cross-layer-cloud environment should be used to run bag-of-tasks applications. Considering the deadline and the cost limitation, Zuo et al. [14] put forward a multi-objective scheduling method oriented by tasks based on ant colony optimization in a cross-layer cloud environment, with the purpose to optimize the restricted pool of edge and public cloud computing resources. Yuan et al. [15] proposed a temporal task scheduling algorithm that combines the energy price of edge cloud resources with the execution price of public cloud resources. Chang [16] studied the task scheduling algorithm and resource allocation algorithm to minimize data transmission energy cost in the cross-layer cloud. Zheng et al. [17] focused on how to improve the performance of social welfare, enlarge cloud bandwidth utilization, as well as increase the satisfaction ratio of the tenant in their study. The method aims to minimize the computational cost and time. An optimization model is established, which can obtain the approximate global optimal solution according to the situation of tasks. The above specific multiple computer task scheduling methods focus on how to adjust the allocation of resources according to the specific requirements of a certain service. They do not consider the universal heterogeneity of tasks in an actual cross-layer cloud system and, therefore, do not meet the diverse needs of cloud users.

Furthermore, traditional methods to solve multi-objectives include the weighting method [18,19], the restraint method [20] and the linear programming method [21]. In addition, many research works have studied intelligent optimization algorithms and proposed innovative algorithms for multi-objective optimization, including multi-objective optimization based on evolutionary algorithms [22], multi-objective optimization based on particle swarm optimization [23] and multi-objective optimization based on the ant colony algorithm [24]. Besides, considering multiple goals and satisfying the impact on task scheduling at the same time, Van et al. [25] proposed a binary scheduling algorithm based on the key factors in the computing system environment. However, when the amount of tasks or the type of resources is large, the time required by these algorithms to schedule resources is quite large. In particular, cloud-edge computing systems often have the characteristics of a large number of heterogeneous nodes, which makes these algorithms inadequate to meet the needs of the actual business. Different from previous research, the framework proposed in this paper pre-processes cloud-edge computing system resources before multi-objective optimization and combines resources' heterogeneity characteristics to reasonably divide resources and task types. Therefore, it can efficiently find out the resource scheduling method to meet the multi-QoS requirements combined with the multi-objective requirements of users in the heterogeneous resources environment. This is of great significance for the improvement of cloud-edge computing system performance.

3. General Framework Design

In this section, we analyze the features of the cross-layer cloud and multiple computer tasks firstly, which includes task scheduling objectives, task types and resource characteristics. Then, based on these results, a detailed description of the framework is given.

3.1. Task Scheduling Objective

The task scheduling objective is set by a cross-layer computing system to meet the requirements of certain computer task types and their own goals. The cross-layer computing system, which is given more flexibility in task scheduling and resource collaboration, is a complex system with many different objectives in various scenes. Specifically, it can be divided into service level agreement, energy conservation, reliability and uncertainty.

3.1.1. Service Level Agreement

IoT services issue the Service Level Agreement (SLA) which, as a human-readable document, is used to give a description of cloud-based IT resources in terms of the guarantees, restrictions and Quality-of-Service (QoS) features. SLA uses service quality metrics to express measurable QoS characteristics. The cross-layer computing systems' mangers put it into these objectives to ensure that they are fulfilling the contractual QoS requirements that are published in SLAs.

3.1.2. Energy Conservation

Generally, wholesalers will buy products in bulk if the price is low. Similarly, the cross-layer computing system provides more price options for users. For instance, as we know, both the unnecessary usage of computational resources and the computing migration between clouds and sensor clusters consume a large amount of energy [26].

3.1.3. Reliability

Reliability is an important performance measure of cross-layer computing systems and an important indicator that IoT services must consider. The mission for combining cloud computing and edge computing is to provide a more efficient method for data processing. At this point, once errors have appeared, this will bring a devastating disaster to cross-layer IoT services. Therefore, providing corresponding fault tolerance mechanisms in both cloud computing and edge computing to improve system reliability is a key issue that cross-layer computing systems must address [27].

3.1.4. Uncertainty

Controlling uncertainty within an acceptable range is another issue that cross-layer computing systems need to address. Studying how to measure and control the uncertainty is an indispensable method to effectively improve the performance of the cross-layer system. For example, the performance of remote cloud and edge clouds is constantly changing at runtime. If the uncertainty caused of system performance cannot be accurately measured at this time, the system's scheduling scheme will be very difficult to implement and even have a negative effect on tasks' allocation and operation [28].

3.2. Task Type

The requirements of different computer tasks are transported to the cross-layer system for processing and scheduling. In general, we divide computer tasks into four aspects to facilitate the processing and scheduling of computer tasks.

3.2.1. Independent and Dependent Tasks

Dependent task requirements can be divided into different types of tasks or their combinations. Sometimes, the users' service needs can be transformed to a single task directly, such as the calculation service of checking the energy account balance. In addition, sometimes it needs to be divided into multiple interrelated tasks to operate, e.g., waking up sensors and other devices in smart home can be decomposed into wake up and initialize different sensors or other devices successively. We define that a single task as an independent task and the others as dependent tasks (be denoted as a directed acyclic graph).

3.2.2. Real-Time Tasks and Non-Real-Time Tasks

Based on the time constraint for completion, tasks can be divided into real-time tasks and non-real-time task. When their deadline is constrained to a certain time, we call them real-time tasks [29]. Conversely, when their constraint to be completed is infinity, they are called non-real-time tasks.

3.2.3. Periodic and Aperiodic Tasks

Prediction of the finish time of periodic tasks is available since timeslots between any adjacent tasks are usually fixed. Therefore, once you know the initial task's detailed information, you can calculate the finish time of others. On the contrary, the finish time of non-periodic tasks cannot be predicted. For instance, some system calibration procedures are typical periodic tasks, and the processing services for mobile payments are usually aperiodic tasks.

3.2.4. Priority and Non-Priority Tasks

The priority of tasks can be given by the cross-layer computing system or the users directly. It comes from the agreement between services and the cross-layer computing system or can be calculated based on task characteristics in a specific system [30]. For example, the deadline of tasks can be the reason for giving priority. As for the non-priority tasks, they only need to wait to be executed in normal process queues.

3.3. Resource Characteristics

In the cross-layer computing system, there are two main features that should be considered in the resource scheduling process, and their differences between the local edge cloud and remoter cloud data center should be compared. We use virtualization to address those resources that process tasks in different clouds and dynamic resource provisioning to demonstrate the resource's characteristics while provisioning by different clouds.

3.3.1. Virtualization

Virtualization is the foundation of the cross-layer system to provide resilient computing services and capabilities. Although the virtualization degree of the remote cloud data center is usually higher than the local edge cloud, we can regard all of them as one big virtualized resource pool. When providing computing services, multiple Virtual Machines (VMs) or containers are established directly at hosts in the local edge cloud or remote cloud data center. Then, they are used as the basic instances of resources to handle tasks rather than physical hosts.

3.3.2. Dynamic Resource Provisioning

The edge cloud can use resources from remote clouds through task migration. In this way, through migrating tasks from edge clouds to the remote data center or vice versa, resources supported by the cross-layer computing system can be dynamic and scalable based on the sensor's demand and the cloud's status. Dynamic resource provisioning is one of the main features of the cross-layer system, which is a significant advantage compared with conventional patterns [31].

3.4. Scheduling Framework

Based on the results of the feature analysis, we design a new multiple computer task scheduling framework for cross-layer cloud computing systems, and it has the ability to be dynamically customizable. Figure 2 shows the overview of the scheduling framework, and it works with the following steps:

Step 1: Demands of different IoT services and applications are decomposed into different computer task types, and a snapshot is created by the service divider, in which the relationship between tasks is described by a directed graph.

Step 2: The task analyzer analyzes the features of computer tasks and attributes various labels to them, including task scheduling objectives, task type, and so forth.

Step 3: The task analyzer distributes the snapshot of tasks to the Scheduling Management Objectives (SMO) analyzer and task scheduler. Besides, tasks are also distributed to the task scheduler.

Step 4: The SMO analyzer can tell the differences between the objectives of each task. The SMO analyzer is an important component that we design for our proposed scheduling framework. The SMO analyzer is responsible for deciding which objectives should be selected. The SMO decision can be derived from three aspects, i.e., task nature, resource state and system hierarchy. The SMO analyzer creates a corresponding objective controller form. The objective controller selects objectives from the objective pool based on the objective controller form.

Step 5: Thisobjective controller monitors the task scheduler to make sure the output of each task is acceptable.

Step 6: The task scheduler performs arithmetic functions with the algorithms from the algorithm pool. Then, it sends information from the resource monitor and the related algorithms' results to the resource scheduler.

Step 7: The resource scheduler allocates resources to these tasks based on the algorithms' results from the task scheduler.

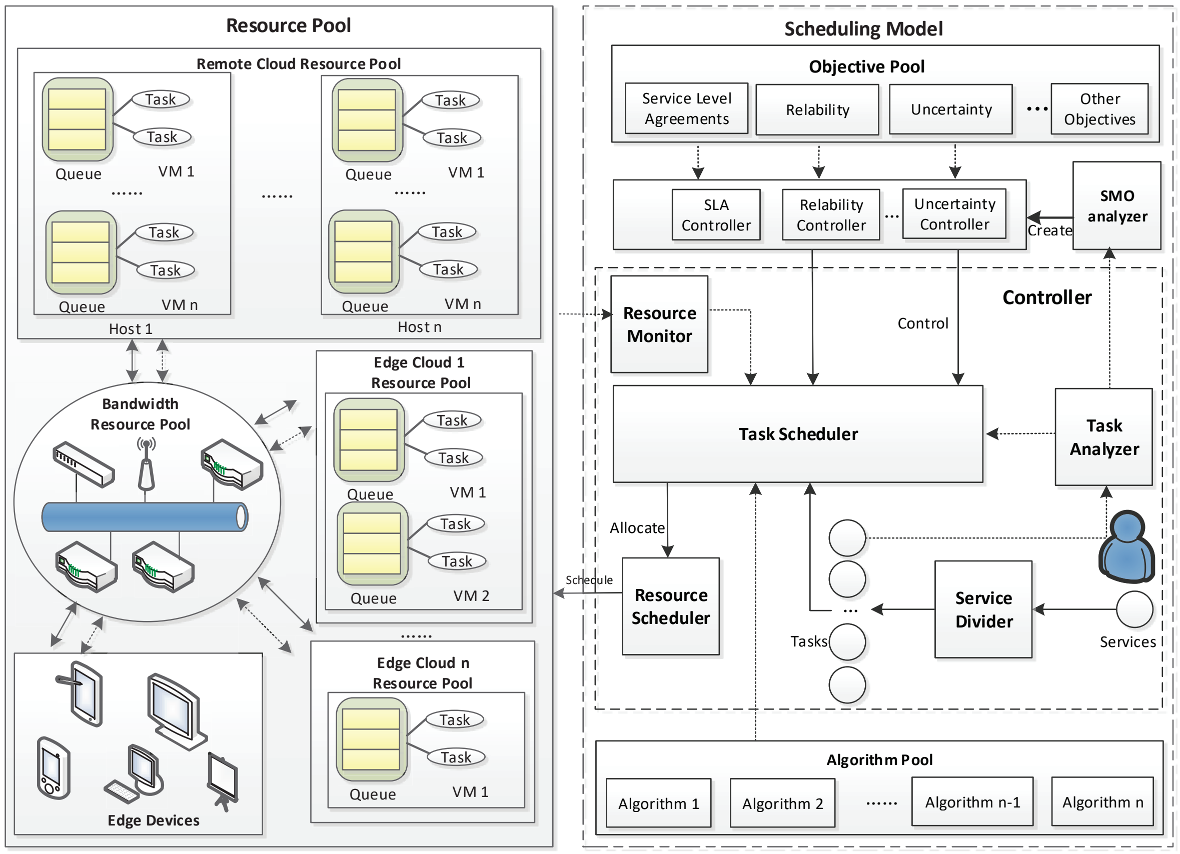


Figure 2. The overview of the scheduling framework.

It is also worth noting that the purpose of the task scheduler is to decide how much of the resources should be allocated to the task, and the resource scheduler schedules resources to perform tasks according to the requests from the task scheduler.

4. Specific Examples and Models

To demonstrate our framework in detail, we use our framework to allocate the computing resource dynamically based on specific tasks. We use Example 1 as the instance of processing an independent task, and it focuses on controlling the uncertainty. Example 2 is for dependent tasks with the aim of good system reliability.

We use ζ as the component in the cross-layer cloud computing system. It can be the remote cloud C_0, a certain edge cloud from the edge cloud set $\{C_1, \ldots, C_n\}$ or an end sensor or other device from the end sensors and other devices set $\{Ed_1, \ldots, Ed_n\}$. Formally, we have $\zeta \in N$, $N = \{C_0, C_1, \ldots, C_n, Ed_1, \ldots, Ed_n\}$, and consider that the resources in the cross-layer cloud computing system are $CS()$. Assume that both the remote and edge cloud in the cross-layer cloud computing system have many hosts to provide computing resources, which can be divided into many VMs or containers. Therefore, a given cloud can be regarded as a set of hosts $\{H_1, H_2, \ldots, H_{n-1}, H_n\}$, and hosts' computing resource can be regarded as the set of VMs $\{VM_1, VM_2, \ldots, VM_{n-1}, VM_n\}$. The relationship of the computing resources between the cloud, host and VM can be denoted as follows:

$$\begin{aligned} CS(\zeta, H_1) &= \sum_{i=1}^{m} CS(\zeta, H_1, VM_i), \\ CS(\zeta) &= \sum_{i=1}^{n} CS(\zeta, H_i). \end{aligned} \tag{1}$$

4.1. Example 1

Example 1 takes the uncertainty as an important objective. It considers that the time-varying of CPU performance cp_{ij} in VM VM_{ij} is the reason for the uncertainty in the resource pool. Besides, the size of a specific task ta_k cannot be measured accurately. Furthermore, the start time, execution time and finish time for tasks in the waiting queue are various, as well. Thus, the uncertainty from tasks should be calculated. We denote $\widetilde{cp}_{ij}$ as the uncertain performance. Similarly, $\widetilde{size}(ta_k)$ represents the uncertain task size; $\widetilde{st}(VM_{ij}, ta_k)$ represents the uncertain start time of task ta_k in VM_{ij}; $\widetilde{et}(VM_{ij}, ta_k)$ is marked as the uncertain execution time of task ta_k in VM_{ij}; $\widetilde{ft}(VM_{ij}, ta_k)$ is marked as the uncertain finish time of task ta_k in VM_{ij}. Therefore, we have:

$$\begin{aligned} cp_{ij} &= [cp_{ij}{}^-, cp_{ij}{}^+], \\ size(ta_k) &= [size(ta_k)^-, size(ta_k)^+], \end{aligned} \tag{2}$$

where $size(ta_k)^-, size(ta_k)^+, cp_{ij}^-, cp_{ij}^+ > 0$. The uncertain execution time $\widetilde{et}(VM_{ij}, ta_k)$ is:

$$\widetilde{et}(VM_{ij}, ta_k) = [size(ta_k)^- / cp_{ij}{}^-, size(ta_k)^+ / cp_{ij}{}^+] \tag{3}$$

Therefore, the uncertain finish time $\widetilde{ft}(VM_{ij}, ta_k)$ is:

$$\widetilde{ft}(VM_{ij}, ta_k) = \widetilde{st}(VM_{ij}, ta_k) \oplus \widetilde{et}(VM_{ij}, ta_k). \tag{4}$$

In terms of a real-time task scheduling, if the task ta_k is completed at or earlier than the deadline, task ta_k will be finished successfully. The label of task $z(VM_{ij}, ta_k)$ can be determined by the following principle:

$$\begin{cases} 1, & \text{if } ((ft(VM_{ij}, ta_k) \leq d_k) \\ & \text{and } (ta_{status}(t)_k = 1)), \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Thus, the quantity of successfully-finished tasks is to be maximized:

$$\begin{aligned} &\max \sum_{k=1}^{|TA|} \sum_{i=1}^{|H_a|} \sum_{j=1}^{|VM_i|} \frac{z(VM_{ij}, ta_k)}{|TA|} | \sum_{i=1}^{|H_a|} \sum_{j=1}^{|VM_i|}, \\ &s.t. : z(VM_{ij}, ta_k) \leq 1, \forall ta_i \in TA. \end{aligned} \tag{6}$$

Then, we consider the energy conservation as another task scheduling objective; the sum of energy consumption in the system should be minimized while scheduling:

$$\min \sum_{i=1}^{|H_a|} \int_{st}^{ft} (\alpha \cdot pow_i \cdot x_i^t + (1 - \alpha) \cdot pow_i \cdot u(t)) dt, \tag{7}$$

where $x_i^t \in \{1, 0\}$ is the "power on" status of host h_i in time slot t. α is the rate of energy consumption in an idle host. pow_j means the overall power consumption of a host. $u(t)$ represents host h_i's CPU utilization at time t.

4.2. Example 2

In this example, we create a task scheduling objective based on reliability. As there is a small chance that two hosts can fail at the same time, it is assumed that when the primary copies are in the host that fails, backup copies will achieve successful completion. We consider the reliability when a host fails at once, which can be ensured by a Primary/Backup (PB) model. By using the PB model, any task ta_k will have a corresponding backup ta_k^B. If ta_k has encountered an accident before it is finished, ta_k^B will be wakened. To distinguish parameters between primary and backup, we us $()^B$ to represent backup as ta_k^B.

The backup task ta_k^B can be executed either actively or passively. Its execution mode $mode(ta_i^B)$ is determined by the following principle:

$$mode(ta_i^B) = \begin{cases} passive & \text{if } d_k - ft_k \geq et(VM_{ij}, ta_k)^B, \\ active & \text{otherwise.} \end{cases} \tag{8}$$

where ft_k represents the predicted ta_k end time. Therefore, the actual execution time $et^a(VM_{ij}, ta_k)^B$ of ta_k^B is:

$$\begin{cases} et(VM_{ij}, ta_k)^B & \mathrm{z(VM_{ij}, ta_k) = 0.} \\ (0, et(VM_{ij}, ta_k)^B] & \mathrm{z(VM_{ij}, ta_k) = 1} \\ & \mathrm{andmode(ta_i^B) = active.} \\ 0 & \mathrm{z(VM_{ij}, ta_k) = 1} \\ & \mathrm{andmode(ta_i^B) = passive.} \end{cases} \tag{9}$$

The sum of the actual execution time of the backup tasks is to be minimized. Thus, the objective is:

$$\min \sum_{k=1}^{|TA|} \sum_{i=1}^{|H_a|} \sum_{j=1}^{|VM_i|} et(VM_{ij}, ta_k)^B \cdot ac_{ij}^B \tag{10}$$

where $ac_{ij} = 1$ when ta_i is allocated to resource ζ_j at time t.

5. Algorithm Design

We discuss three task scheduling algorithms, EASU (Energy-Aware Scheduling under Uncertainty), RAS (Reliability-Aware Scheduling) and the basic resource provisioning algorithm in this section. The EASU and RAS algorithms correspond to Example 1 and Example 2 and can be easily added into the algorithm pool in the architecture we setup.

5.1. EASU Algorithm

To process the uncertainty in the system, WQ, a waiting queue, and UQ, an urgent queue, are defined. When the tasks' deadline is urgent in WQ, they will be sent to UQ. We define that a task is urgent when its laxity L_k [32] reaches the given threshold L_d. The threshold is the time cost to set up a VM on a shutdown host. We have the EASU pseudocode as follows:

In this Algorithm 1, the uncertainty is measured by the interval number. ft_{ijk}^- and ft_{ijk}^+ represent the minima and maxima of the estimated finish time. By checking all VMs' ft_{ijk}^- and ft_{ijk}^+ (as shown in Lines 4–9), ft_{min} and ft_{max} are recorded. If ta_k's deadline d_k is larger than or equal to ft_{max} and is shorter than or equivalent to d_k, then the task will be scheduled for a VM that has minimal energy consumption (as shown in Lines 10–11). If d_k is between ft_{max} and ft_{min}, EASU will choose a VM holding the minimal ft_{ijk}^+ (as shown in Lines 12–13). Otherwise, the task ta_k will be rejected, while $L_i > L_d$ means that it is feasible to firstly launch a host and accordingly create a new VM to operate ta_k (as shown in Lines 15–18). In the resource provisioning algorithm, the function scaleUpResources() will be introduced.

Algorithm 1: Pseudocode of EASU.

```
$WQ \leftarrow \emptyset; UQ \leftarrow \emptyset$;
foreach new task $ta_k$ do
    $ft_{MAX} \leftarrow 0; ft_{MIN} \leftarrow +\infty$;
    foreach VM $VM_{ij}$ do
        calculate the minimum of estimated finish time $ft^{-}_{ijk}$ and maximum of estimated $ft^{+}_{ijk}$
         and energy $ec_{ijk}$;
        if $ft^{-}_{ijk} < ft_{min}$ then
            $ft_{MIN} \leftarrow ft^{-}_{ijk}$;
        if $ft^{+}_{ijk} > ft_{max}$ then
            $ft_{max} \leftarrow ft^{+}_{ijk}$;
    if $ft_{max} \leq d_k$ then
        allocate $ta_k$ to the $VM_{ij}$ with the minimal $ec_{ijk}$;
    else if $ft_{max} > d_k$ and $ft_{min} \leq d_k$ then
        allocate $ta_k$ to the $VM_{ij}$ with minimal $ft^{+}_{ijk}$;
    else
        if $L_k < L_d$ then
            Reject task $ta_k$;
        else
            call function scaleUpResources() and allocate $ta_k$ to a new VM;
```

5.2. RAS Algorithm

Example 2 involves using the PB model to ensure the reliability-aware scheduling, and the primary tasks and backup tasks need to be respectively scheduled. Algorithm 2 shows the pseudocode of primary scheduling in RAS.

In this algorithm, the top α% hosts holding less primaries are selected as candidate hosts (as shown in Line 1), firstly. Then, we choose the VM that can complete primary tasks at the earliest time (as shown in Lines 4–11). If candidate hosts have no VM to complete primary tasks prior to the deadline, the next top α% hosts will be selected for the convenience of the next round of the search (as shown in Lines 12–15). In this way, the primary tasks can achieve an even distribution. In the case of no existing VMs able to accommodate any primary task, the scaleUpResources() function will be called (as shown in Line 17).

We omitted the backup scheduling algorithm similar to Algorithm 2. It is worth noting that it is impossible for a backup to schedule on a host that sees the allocation of primary task. In addition, a certain number of dependent constraints should be integrated into the backup scheduling algorithm.

Algorithm 2: Pseudocode of primaries scheduling in RAS.

```
$H_{candidate} \leftarrow$ top $\alpha$% hosts in $H_a$;
$eft \leftarrow +\infty$; $VM \leftarrow null$;
while all hosts in $H_a$ have been scanned do
    foreach $h_i$ in $H_{candidate}$ do
        if $h_i$ satisfies $ta_i$'s scheduling dependent constraints then
            foreach $VM_{ij}$ in $h_i.VmList$ do
                calculate the earliest start time $est_i$;
                $eft_i^B \leftarrow est_i^B + e_{ikl}^B$;
                if $eft_i^B < eft$ then
                    $eft \leftarrow eft_i^B$;
                    $VM \leftarrow VM_{ij}$;
    if $eft > d_k$ then
        $H_{candidate} \leftarrow$next top $\alpha$% hosts in $H_a$;
    else
        break;
if $eft > d_k$ then
    if scaleUpResources($ta_k$) then
        return $true$;
    else
        allocate $ta_k$ to $VM_{ij}$;
else
    allocate $ta_k^B$ to $VM_{ij}$;
```

5.3. Resource Provisioning Algorithm

There are two functions in the resource provisioning algorithm, scaleUpResources() and scaleDownResources(), which are proposed based on the prior algorithms.

As shown in Algorithm 3, function scaleUpResources() can select a host with the highest probability of holding the lowest accommodating ability to VM_j (as shown in Lines 3–5). With no such host being observed, it will migrate the VM (see Lines 7–11). Then, it will be checked if this migrated VM VM_j is able to be added to the host. Thus, the VM_j will be created to check if the task is able to achieve completion on VM_j prior to the deadline or not (as shown in Lines 12–15). When the migration is infeasible or tasks are unable to achieve successful completion, h host h_i will be launched, and a VM_{ij} will be created. Subsequently, it is necessary to check if tasks can achieve successful completion on VM_{ij} (as shown in Lines 16–20).

As for the scaleDownResources() function shown in Algorithm 4, the VM will be deleted if its ideal time exceeds the threshold (as shown in Lines 2–4). Host with no VMs will be shut down (as shown in Lines 5–7). In the case that all VMs on a host in SH are able to be migrated to one or more hosts in DH, these VMs will be migrated to a destination host, and the outset host will be closed after the migration. In contrast, all VMs will fail in migrating when a certain number of VMs is unable to migrate from the outset host (as shown in Lines 10–21).

Algorithm 3: Pseudocode of scaleUpResources().

```
select a type of $VM_j$ with minimal MIPS if $ta_i$ can be completed prior to the deadline;
sort hosts in $H_a$ following the descending order of the CPU utilization;
foreach host $h_i$ in $H_a$ do
    if $VM_j$ can be added in host $h_i$ then
        create $VM_{jk}$; $findTag \leftarrow$ TRUE; break;
if $findTag == FALSE$ then
    search host $h_s$ with the lowest CPU utilization;
    seek out $VM_{ps}$ with minimal MIPS in $h_s$;
    foreach host $h_i$ except $h_s$ in $H_a$ do
        if $VM_{ps}$ will be added in host $h_i$ then
            migrate $VM_{ps}$ to host $hi$; break;
    if VM $VM_j$ will be added in host $h_s$ then
        create $VM_{js}$;
        if $t_i$ will be completed in $VM_{js}$ prior to the deadline then
            $findTag \leftarrow$ TRUE;
if $findTag == TRUE$ then
    start up a host $h_n$, and place it in $H_a$;
    create $VM_{jn}$ on $h_n$;
    if $ta_i$ can be finished in $VM_{jn}$ before its deadline then
        $findTag \leftarrow$; TRUE;
```

Algorithm 4: Pseudocode of scaleDownResources().

```
$SH \leftarrow \emptyset$; $DH \leftarrow \emptyset$;
foreach $VM_{ij}$ in the system do
    if $VM_{ij}$'s idle time $it_{ij} > THRESH$ then
        remove $VM_{ij}$ from host $h_i$, and delete it;
foreach host $h_i$ in $H_a$ do
    if no on $h_i$ then
        shut down host $h_i$, and remove it from $H_a$;
sort the hosts in $H_a$ in the increment order of CPU utilization;
$SH \leftarrow H_a$; $DH \leftarrow H_a$, and sort $DH$ inversely;
foreach host $h_i$ in $SH$ do
    $shutDownTag \leftarrow$ TRUE; $AH \leftarrow \emptyset$;
    foreach VM $VM_{ij}$ in $h_i$ do
        $migTag \leftarrow$ FALSE;
        foreach host $h_p$ in $DH$ except $h_i$ do
            if $VM_{ij}$ can be added in $h_p$ then
                $migTag \leftarrow$ TRUE; $AH \leftarrow h_p$; break;
        if $migTag == FALSE$ then
            $shutDownTag \leftarrow$ FALSE; break;
    if $shutDownTag \leftarrow TRUE$ then
        migrate VMs in $h_i$ to destination hosts; $SH \leftarrow SH - AH - h_i$; $DH \leftarrow DH - h_i$;
        shut down host $h_i$, and remove it from $H_a$;
```

6. Framework Implementation

In this section, we conduct our simulation experiments with the simulation platform CloudSim toolkit and set the parameters of the cloud environment on the basis of Apache CloudStack 4.2.0. CloudStack is a kind of open source software designed to deploy and manage large networks of virtual machines, as a highly available, highly scalable Infrastructure as a Service (IaaS) cloud computing platform. CloudStack has been widely deployed in IoT environment to provide multiple cloud services. By setting the cloud environment parameters based on CloudStack, it can make our method be more easily deployed in a real system.

The physical host in CloudSim was equipped with 500 G of disk storage, 3.7 G of memory and a i3, 3.9-GHz quad-core CPU. Its peak power is 200 W. Each VM has two 23.9-GHz CPU cores and 1.5 G of memory. According to [32], the machines operated with a 1-Gbps Ethernet network. CloudSim is an extensible simulation toolkit that supports both system and behavior modeling of different cloud system components [33]. It implements generic application provisioning techniques and can be extended with ease and limited effort. We can easily simulate different clouds in the IoT environment based on CloudSim. With the simulation platform CloudSim, we set the "cloudletList()" component to simulate the workload collected from the sensor clusters. Combining the "CloudSim.clock" and "CloudSim.processEvent()" components, the operating properties of our method under different conditions can be sufficiently tested with the simulation environment.

In addition, we set different types of workloads for different test targets. For Example 1, the arrival rate is determined with the Poisson distribution, and the average amount of the task counts is set as 5000. Considering the features of the realistic workload, $intervalTime$ determines the time interval between two sequential tasks, and $intervalTime$ = 0.5 s in the study. For Example 2, the workload is represented by the DAG count, and we assume each task set T(or a DAG job) has random precedence constraints that are generated by the steps in [34]. The DAG size is determined by the message count M, and we have $M = \theta \times N$, where θ represents the degree of task dependence and N is set as 200. In the experiments, we test the performance impacts of the workload by setting different DAG counts and values of θ.

6.1. Parameter Setting and Experimental Results of Example 1

Since Example 1 takes the uncertainty as an important objective, we denote $vmUncertainty$ as the upper bounds of the uncertainty of VMs in the system. The lower bound and upper bound of the performance of a VM are shown as follows:

$$cp_{ij}^{-} = cp_{ij}^{+} \times (1 - U[0, vmUncertainty]). \tag{11}$$

where $U[0, vmUncertainty]$ represents an evenly-distributed random variable between zero and $vmUncertainty$ and cp_{ij}^{+} represents the required capacity for VM_{ij} of the CPU.

Based on the real-time nature of the tasks, we define the $deadlineBase$ for controlling the deadline of a task. It is used to check that the task has a loose or tight deadline:

$$d_i = a_i + U[deadlineBase, a \times deadlineBase]. \tag{12}$$

where $deadlineBase$ is set at 400 s and a is set at four. The arrival rate is determined with the Poisson distribution. The $intervalTime$ determines the time interval between two sequential tasks, and $intervalTime$ = 0.5 s in the study.

The EASU is compared to the algorithm NMEASU and the Earliest Deadline First algorithm (EDF). NMEASU is a scheduling algorithm that schedules tasks to virtual machines with the earliest starting time, which we developed based on the OUD-OLBmethod in [35]. The algorithm does not use virtual machine online migration technology to merge virtual machine resources. In addition, all tasks to be executed by the algorithm are on the virtual machine. We discuss their following three metrics:

a Guarantee Ratio (GR): This refers to the ratio of tasks completed prior to the deadlines;

b Resource Utilization (RU): This refers to the average host utilization. We denote it as:

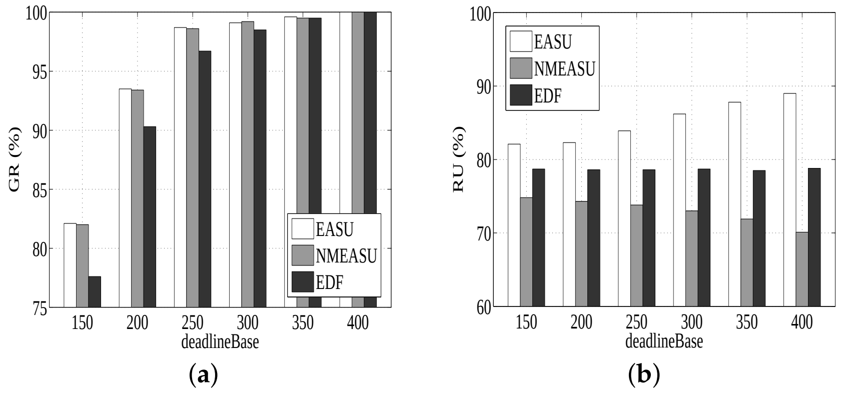
$$RU = (\sum_{k=1}^{|TA|} \sum_{i=1}^{|H_a|} \sum_{j=1}^{|V_i|} et_{ijk} \cdot z_{ijk}) / (\sum_{i=1}^{|H_a|} cp_i \cdot wt_i), \tag{13}$$

where wt_i represents the active time of host h_i in the entire experimental process.

c Total Energy Consumption (TEC): This refers to the total energy consumption by hosts for the execution of the task set T.

6.1.1. Performance Impact of the Task Deadline

As shown in Figure 3a, all three algorithms' GR climb as *deadlineBase* rises, as more tasks will be completed prior to deadlines, and the mechanism of resource scale-up is not needed. It also can be found that the GR of EASU and NMEASU is higher than EDF. The reasons can be easily found. Firstly, more tasks satisfy their deadline because EASU and NMEASU have higher priority. Secondly, EASU and NMEASU are more flexible and are able to achieve the dynamic allocation of resources to accommodate more tasks. Finally, EASU takes the uncertainties into consideration, and NMEASU has a better quality of scheduling.



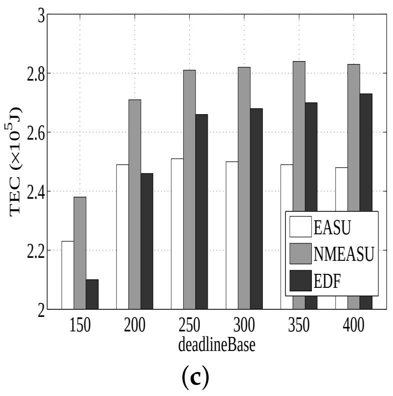


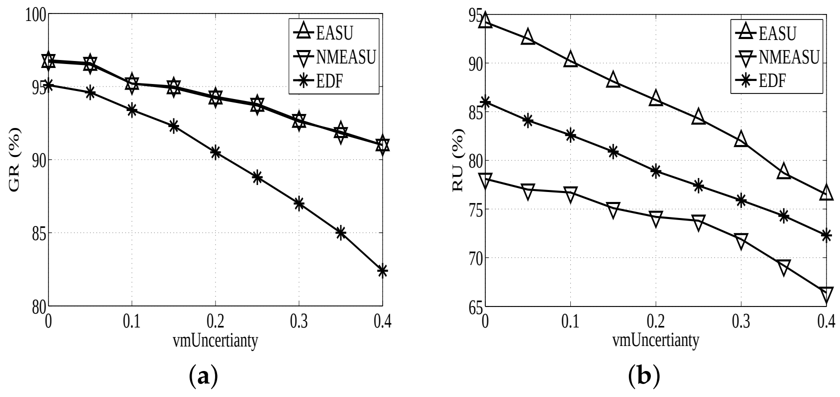
Figure 3. Performance impacts of task deadlines. (**a**) Impact of *deadlineBase* on GR, (**b**) Impact of *deadlineBase* on RU, (**c**) Impact of *deadlineBase* on TEC.

According to Figure 3b, RU of three algorithms grows as *deadlineBase* rises. It is helpful due to the fact that the active hosts' utilization will be higher and more tasks will be completed inside currently active hosts with no need to start more hosts, leading to looser deadlines. Similarly, since VM migration is beneficial for strengthening resources in an effective way, resources can achieve effective utilization, and the RU of EASU is higher than NMEASU. In addition, no scale-down function is deployed for EDF, causing the minimal RU.

According to Figure 3c, the TEC numbers of EASU, NMEASU and EDF increase as $deadlineBase$ increases. The reason is that with the rise of $deadlineBase$, more tasks require consuming more energy. EASU consumes less energy compared to NMEASU since NMEASU is inefficient in resource utilization. EDF requires more energy than EASU once $deadlineBase$ exceeds 200 s because it ignores energy conservation.

6.1.2. Performance Impact of Uncertainty

Figure 4a indicates that the GR of the three algorithms drops as $VMUncertainty$ increases, especially the EDF's trend. This is because EDF fails to deploy strategies to manage uncertainties, which will cause really bad scheduling quality.



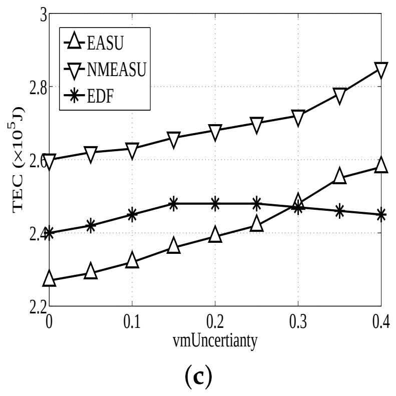


Figure 4. Performance impacts of task deadlines. (**a**) Impact of $VMUncertainty$ on GR, (**b**) Impact of $VMUncertainty$ on RU, (**c**) Impact of $VMUncertainty$ on TEC.

Based on Figure 4b, RU of the three algorithms presents an obvious decrease as $VMUncertainty$ increases because the performance of VMs will obviously decline and require more resources of physical hosts with the rise of $VMUncertainty$. EASU is better than the others for deploying the uncertainty-aware and VM migration strategies.

Based on Figure 4c, the TECs of EASU and NMEASU rise because too many tasks need to be finished urgently prior to the deadlines in VMs when $VMUncertainty$ increases, and this consumes more resources. EASU requires more energy compared with EDF when $VMUncertainty$ exceeds 0.3. This is because EASU tends to manage the uncertainty and process more tasks without considering the energy consumption.

6.2. Parameter Setting and Experimental Results of Example 2

Since Example 2 pays more attention to the dependent tasks, we denote TA or a DAG job as each task set. According to [34], we set the random precedence constraints of TA. We denote the message count M and have $M = \theta \times N$ when the number of dependent tasks N is given. Other parameters of Example 2 are the same as Example 1.

The RAS with the algorithm NCRAS (NCRAS simulates a non-fault tolerant real-time scheduling algorithm that is unable to tolerate any failure. It is proposed for the purpose of comparison.) is compared with the algorithm eFRD, which is an easy approach to conduct primary, as well as backup scheduling [34]. We discuss their following three metrics.

(1) Guarantee Ratio (GR): This refers to the ratio of tasks completed prior to their deadlines;
(2) Host Active Time (HAT): This refers to the total running time of all hosts in the cross-layer cloud computing system;
(3) Ratio of Task time and Host time (RTH): This refers to the rate of the execution time of all of the tasks with respect to the total active time of hosts.

6.2.1. Performance Impact of the DAG Count

As shown in Figure 5a, RAS and NCRAS are basically unchangeable GRs for various DAG counts. This is due to RAS and NCRAS taking into account the limitless resources in the cross-layer cloud computing system. The increase in DAG count will make new hosts dynamically attached to more DAGs. Because eFRD cannot dynamically manage resources, the GR of eFRD drops with the rise of the DAG count.

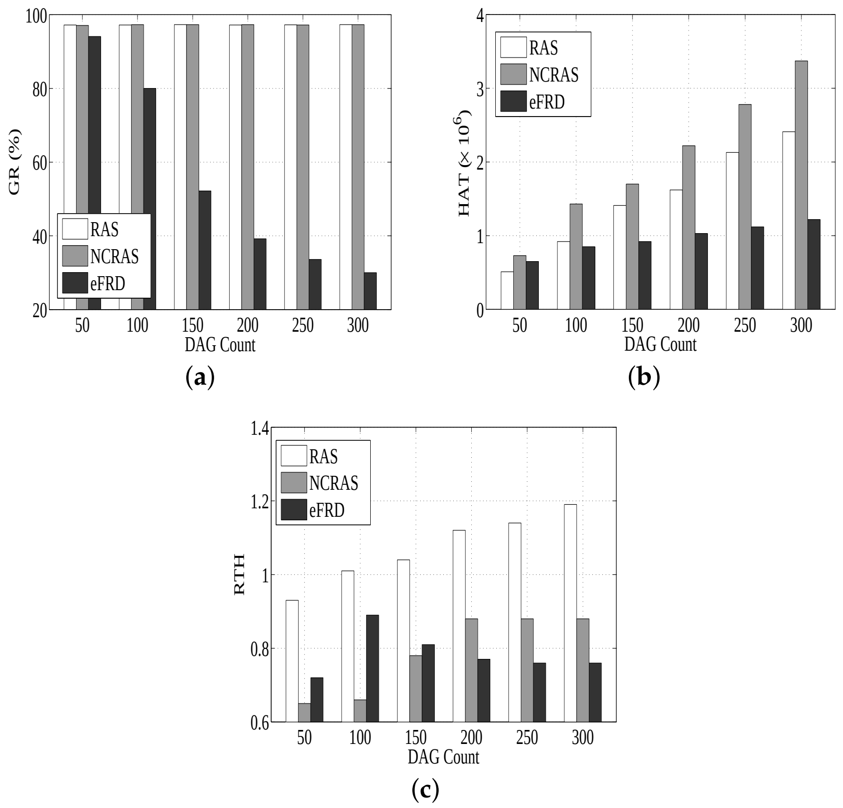


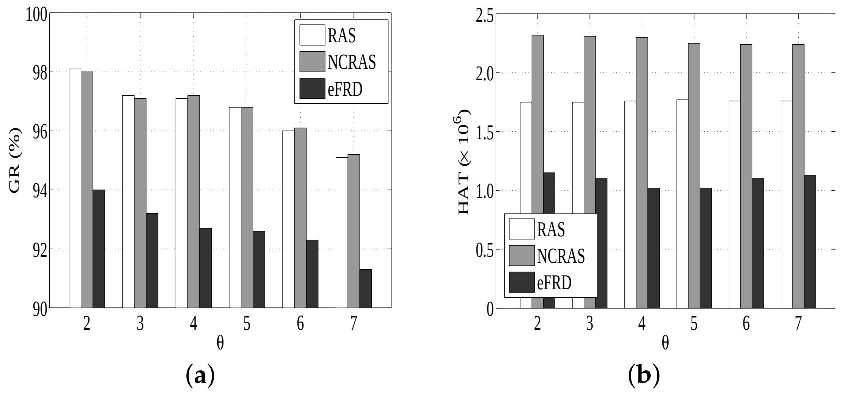
Figure 5. Performance impacts of the DAG count. (**a**) Impact of the DAG count on GR, (**b**) Impact of the DAG count on HAT, (**c**) Impact of the DAG count on RTH.

Figure 5b illustrates that the HAT of RAS is lower than NCRAS, and it proves that RAS does better compared to NCRAS in the resource management. Furthermore, without a consolidation mechanism in NCRAS, the highest resource utilization is caused by the idle state of some resources.

In Figure 5c, the RTHs of RAS and NCRAS ascend. The RTH of eFRD rises first and then drops. This is because it is assumed that its resources are fixed in eFRD. There are enough resources to accommodate most DAGs (ranging from 50–100). However, a further rise of the DAG count, DAG count $>$ 100, leads to the saturation of the system, long running times of hosts and a low level of resource utilization. It also can be observed that the highest RTH is in RAS, since it considers the consolidated resource mechanism.

6.2.2. Performance Impact of Task Dependence

As shown in Figure 6a, the GR of all the algorithms modestly drops when the θ rises. The reason is that the schedulability of the system is degraded with the rise of task dependency. This is similar to the other experiments such as RAS and NCRAS.



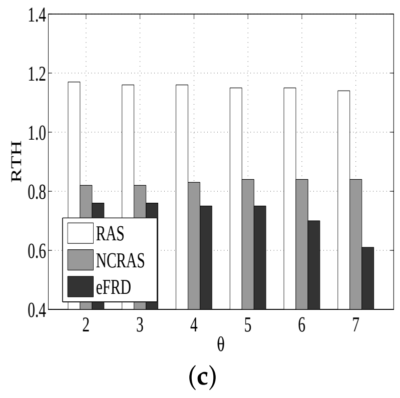


Figure 6. Performance impacts of Task Dependence. (**a**) Impact of Task Dependence on GR, (**b**) Impact of Task Dependence on HAT, (**c**) Impact of Task Dependence on RTH.

Figure 6b illustrates that the HAT of NCRAS is the highest. Although the small task dependence is able to allow more tasks to be executed in parallel and some hosts will be released, as well as closed at an earlier time, NCRAS does not have the advantage, and it requires more resources. Hence, the difference in HAT between NCRAS and other algorithms drops with the rise of θ (this means that the number of tasks supporting parallel execution is decreased).

In Figure 6c, RAS has a higher RTH than the others, and the RTH of all the algorithms drops when θ rises. The reason is due to the high task dependence, which leads to more execution limitations and fewer tasks that are available for VMs despite their idleness. It is helpful to save resources and maintain a lower RTH, since the tasks executed in parallel drop with the rise of θ.

7. Conclusions

In this paper, we propose a new cross-layer cloud scheduling framework for multiple IoT computer tasks, and the IoT services deployed in cross-layer cloud computing systems can dynamically select suitable algorithms and use resources more effectively to finish computer tasks with different objectives. The features of the cross-layer cloud and IoT computer tasks are analyzed, and the scheduling framework is given based on the analysis. Then, two typical examples of different task scheduling schemes and their detailed models are proposed based on the framework so as to illustrate the procedures of using the framework. To cope with two typical examples with our framework, we proposed EASU, the RAS algorithm and the basic resource provisioning algorithm in Section 4. We compared them to the approaches previously proposed by [34,35]. The experiments' results validate its effectiveness, as well as its superiority. Our further studies will focus on the following aspects: (1) we will combine some intelligent forecast mechanisms into our framework to make the scheduling results more precise; and (2) we will extend our framework to improve the ability of QoS estimation and management since different IoT services usually have different SLA levels.

Author Contributions: Conceptualization, Guanlin Wu and Xiaomin Zhu; Methodology, Guanlin Wu and Weidong Bao; Software, Xiongtao Zhang; Validation, Guanlin Wu, Xiaomin Zhu and Weidong Bao.; Formal Analysis, Xiongtao Zhang; Resources, Xiaomin Zhu; Data Curation, Xiongtao Zhang; Writing—Original Draft Preparation, Guanlin Wu; Writing—Review and Editing, Xiaomin Zhu and Weidong Bao; Visualization, Xiongtao Zhang; Project Administration, Weidong Bao; Funding Acquisition, Weidong Bao

Funding: This research is supported by the National Natural Science Foundation of China under grant 61572511, 91648204 and 71702186, and the China Postdoctoral Science Foundation under grants 2016M602960 and 2017T100796.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Vaquero, L. M.; Rodero-Merino, L. Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing. *Acm SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 27–32. [CrossRef]
2. Erl, T.; Puttini, R.; Mahmood, Z. *Cloud Computing: Concepts, Technology and Architecture*; Prentice Hall Press: Upper Saddle River, NJ, USA, 2013; pp. 236–239.
3. Mehta, G.; Deelman, E.; Knowles, J.A.; Chen, T.; Wang, Y.; Vöckler, J.; Buyske, S.; Matise, T. Enabling Data and Compute Intensive Workflows in Bioinformatics. In *International Conference on Parallel Processing*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 23–32.
4. Dinh, T.; Kim, Y.; Lee, H. A Location-Based Interactive Model of Internet of Things and Cloud (IoT-Cloud) for Mobile Cloud Computing Applications. *Sensors* **2017**, *17*, 489. [CrossRef] [PubMed]
5. Liu, A.; Huang, M.; Zhao, M.; Wang, T. A Smart High-Speed Backbone Path Construction Approach for Energy and Delay Optimization in WSNs. *IEEE Access* **2018**, *6*, 13836–13854. [CrossRef]
6. Liu, Y.; Ota, K.; Zhang, K.; Ma, M.; Xiong, N.; Liu, A.; Long, J. QTSAC: An Energy-Efficient MAC Protocol for Delay Minimization in Wireless Sensor Networks. *IEEE Access* **2018**, *6*, 8273–8291. [CrossRef]
7. Ergu, D.; Kou, G.; Peng, Y.; Shi, Y.; Shi, Y. The analytic hierarchy process: task scheduling and resource allocation in cloud computing environment. *J. Supercomput.* **2013**, *64*, 835–848. [CrossRef]
8. Tang, L.; He, S.; Li, Q. Double-Sided Bidding Mechanism for Resource Sharing in Mobile Cloud. *IEEE Trans. Veh. Technol.* **2017**, *66*, 1798–1809. [CrossRef]
9. Zhu, X.; Zha, Y.; Liu, L.; Jiao, P. General Framework for Task Scheduling and Resource Provisioning in Cloud Computing Systems. In Proceedings of the The IEEE Computer Society International Conference on Computers, Software and Applications, Atlanta, GA, USA, 10–14 June 2016; pp. 664–673.
10. Wu, G.; Chen, J.; Bao, W.; Zhu, X.; Xiao, W.; Wang, J. Towards Collaborative Storage Scheduling using Alternating Direction Method of Multipliers for Mobile Edge Cloud. *J. Syst. Softw.* **2017**, *134*, 29–43. [CrossRef]

11. Sun, D.; Zhang, G.; Yang, S.; Zheng, W.; Khan, S.U.; Li, K. Re-Stream: Real-time and energy-efficient resource scheduling in big data stream computing environments. *Inf. Sci.* **2015**, *319*, 92–112. [CrossRef]
12. Rao, J.; Bu, X.; Wang, K.; Xu, C.Z. Self-adaptive provisioning of virtualized resources in cloud computing. *ACM SIGMETRICS Perform. Eval. Rev.* **2011**, *39*, 321–322.
13. Farahabady, M.H.; Lee, Y.C.; Zomaya, A.Y. Randomized approximation scheme for resource allocation in hybrid-cloud environment. *J. Supercomput.* **2014**, *69*, 576–592.
14. Zuo, L.; Shu, L.; Dong, S.; Chen, Y.; Yan, L. A Multi-objective Hybrid Cloud Resource Scheduling Method Based on Deadline and Cost Constraints. *IEEE Access* **2016**, *PP*, 1. [CrossRef]
15. Yuan, H.; Bi, J.; Tan, W.; Zhou, M.; Li, B.H.; Li, J. TTSA: An Effective Scheduling Approach for Delay Bounded Tasks in Hybrid Clouds. *IEEE Trans. Cybern.* **2016**, *PP*, 1–11. [CrossRef] [PubMed]
16. Chang, Z.; Gong, J.; Ristaniemi, T.; Niu, Z. Energy-Efficient Resource Allocation and User Scheduling for Collaborative Mobile Clouds With Hybrid Receivers. *IEEE Trans. Veh. Technol.* **2014**, *65*, 9834–9846. [CrossRef]
17. Zheng, Z.; Gui, Y.; Wu, F.; Chen, G. STAR: Strategy-Proof Double Auctions for Multi-Cloud, Multi-Tenant Bandwidth Reservation. *IEEE Trans. Comput.* **2015**, *64*, 2071–2083. [CrossRef]
18. Qi, L.; Ni, J.; Xia, X.; Wang, H.; Yan, C. A Multi-dimensional Weighting Method for Historical Records in Cloud Service Evaluation. In Proceedings of the IEEE 4th International Conference on Big Data and Cloud Computing, Sydney, Australia, 3–5 December 2014; pp. 265–266.
19. Shravan, N.; Vaidehi, V.; Sangeetha, D.; Antony, R.R.; Vaikundam, R.R. Abnormality Detection on Vital Parameters Using Modified Weighted Average Method in Cloud. In Proceedings of the IEEE International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials, Chennai, India, 2–4 August 2017; pp. 63–68.
20. Xiao, Y.; Wang, J.; Li, Y.; Gao, H. An Energy-Efficient Data Placement Algorithm and Node Scheduling Strategies in Cloud Computing Systems. *J. Comput. Res. Dev.* **2013**, *42*, 59–63.
21. Wang, C.; Ren, K.; Wang, J. Secure Optimization Computation Outsourcing in Cloud Computing: A Case Study of Linear Programming. *IEEE Trans. Comput.* **2015**, *65*, 216–229. [CrossRef]
22. Yusoh, Z.I.M.; Tang, M. Composite SaaS Placement and Resource Optimization in Cloud Computing Using Evolutionary Algorithms. In Proceedings of the IEEE International Conference on Cloud Computing, Honolulu, HI, USA, 24–29 June 2012; pp. 590–597.
23. Netjinda, N.; Sirinaovakul, B.; Achalakul, T. Cost optimal scheduling in IaaS for dependent workload with particle swarm optimization. *J. Supercomput.* **2014**, *68*, 1579–1603. [CrossRef]
24. Pacini, E.; Mateos, C.; Garino, C.G. Balancing Throughput and Response Time in Online Scientific Clouds via Ant Colony Optimization. *Adv. Eng. Softw.* **2014**, *84*, 31–47. [CrossRef]
25. Bossche, R.V.D.; Vanmechelen, K.; Broeckhove, J. Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads. In Proceedings of the IEEE International Conference on Cloud Computing, Miami, FL, USA, 5–10 July 2010; pp. 228–235.
26. Tang, J.; Liu, A.; Zhang, J.; Xiong, N.N.; Zeng, Z.; Wang, T. A Trust-Based Secure Routing Scheme Using the Traceback Approach for Energy-Harvesting Wireless Sensor Networks. *Sensors* **2018**, *18*, 751. [CrossRef] [PubMed]
27. Liu, Y.; Dong, M.; Ota, K.; Liu, A. ActiveTrust: Secure and Trustable Routing in Wireless Sensor Networks. *IEEE Trans. Inf. Forens. Secur.* **2017**, *11*, 2013–2027. [CrossRef]
28. Tao, X.; Ota, K.; Dong, M.; Qi, H.; Li, K. Performance Guaranteed Computation Offloading for Mobile-Edge Cloud Computing. *IEEE Wirel. Commun. Lett.* **2017**, *6*, 774–777. [CrossRef]
29. Liu, Z.; Dong, M.; Zhang, B.; Ji, Y.; Tanaka, Y. RMV: Real-Time Multi-View Video Streaming in Highway Vehicle Ad-Hoc Networks (VANETs). In Proceedings of the Global Communications Conference, Singapore, 4–8 December 2017; pp. 1–6.
30. Ota, K.; Dong, M.; Gui, J.; Liu, A. QUOIN: Incentive Mechanisms for Crowd Sensing Networks. *IEEE Netw.* **2018**, *PP*, 1–6. [CrossRef]
31. Wu, G.; Bao, W.; Zhu, X.; Xiao, W.; Wang, J. Optimal Dynamic Reserved Bandwidth Allocation for Cloud-Integrated Cyber-Physical Systems. *IEEE Access* **2017**, *PP*, 1. [CrossRef]
32. Chen, H.; Zhu, X.; Guo, H.; Zhu, J.; Qin, X.; Wu, J. Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment. *J. Syst. Softw.* **2015**, *99*, 20–35. [CrossRef]

33. Calheiros, R.N.; Ranjan, R.; Beloglazov, A.; Rose, C.A.F.D.; Buyya, R. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exp.* **2010**, *41*, 23–50. [CrossRef]
34. Qina, X.; Jiangb, H. A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems. *Parallel Comput.* **2006**, *32*, 331–356. [CrossRef]
35. Huang, D.; Yuan, Y.; Zhang, L.; Zhao, K. Research on Tasks Scheduling Algorithms for Dynamic and Uncertain Computing Grid Based on a+bi Connection Number of SPA. *J. Softw.* **2009**, *4*, 1102–1109. [CrossRef]