Latency-Driven Edge Offloading in Mixed-Critical System

Tian Deng\*

Chengdu University of Information and Technology, dengtian1997@outlook.com

Yan Shen

Chengdu University of Information and Technology

Abstract: With the increasing number of services and industries including automatic chemical plants, UAVs and autopilot in mixed-critical systems, and the system does not have enough performance. Mixed-critical system comprises mixed-critical tasks **which are of either mission-critical (high critical) and normal (low critical) task(,either mission-critical(high critical) tasks or normal(low critical) ones)**. In traditional systems **which all task execution in user device**(,where all tasks are all executed in user devices), but(×) user device performance can only be **enhanced**(improved) to a certain level. In this **work**(paper), we propose a framework to schedule the **mixed criticality** tasks by calculating their virtual deadline and **providing**(provide) an **offload**(offloading) strategy based (on) simulated annealing. **As a result**(as results show), the high critical tasks meet all their deadlines, and the system has lower latency and **local**(higher) resource utilization.

CCS CONCEPTS • Computer systems organization • Real-time systems • Real-time system architecture

**Additional Keywords and Phrases:** mixed-critical system, mobile edge computing (MEC), computing offloading

ACM Reference Format:

Tian Deng, Yan Sheng, and Third Author’s Name, Initials, and Last Name. 2021

1. Introduction

For **many**(most) mixed-critical systems (such as automatic chemical plants, UAVs and autopilot), there are not only safety-critical and mission critical tasks, but also some normal tasks. If all tasks are **performed**(executed) locally, the system may not have sufficient resources to perform the necessary operations and deliver the results in time. Due to the size and cost of the hardware, energy efficiency, **size**(?) and other factor(s), user device performance(s) can only be enhanced to a certain level. (As) New computing **paradigm**(s) represented by edge computing and the wireless communication **technology**(technologies) represented by 5G **is**(are) developing **vigorously**(rapidly)**. So that**(,) low critical tasks can be **offloading**(offloaded) to edge servers (by) using wireless **connections**(networks) to reduce the **pressure**(s) on **it**(user devices).

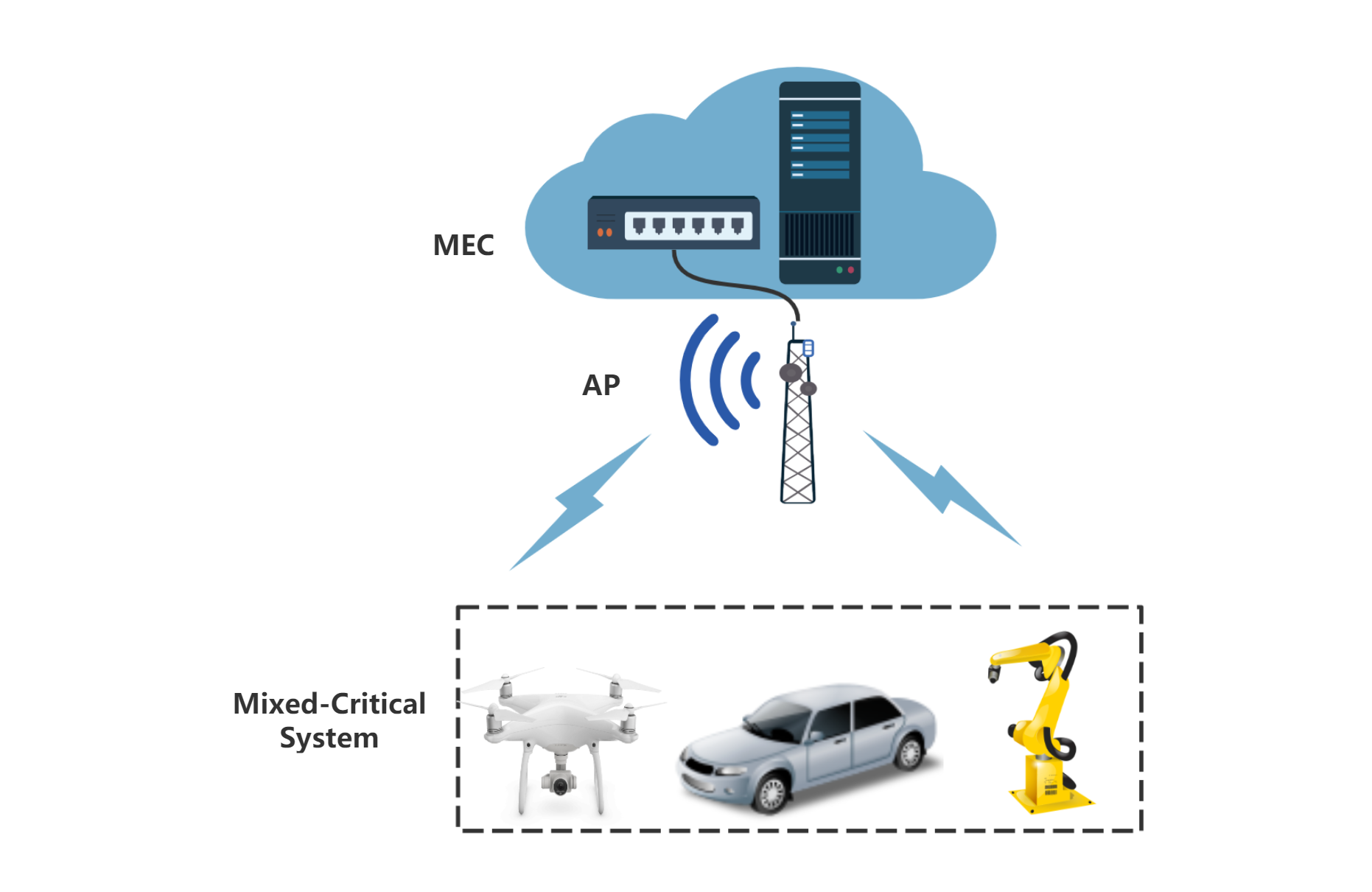


Figure 1: Mobile Edge Offloading

In this paper, we **firstly**(first) **introduced**(introduce) mixed-critical systems and mobile edge computing. Then we define **the**(a) system model, including the scheduling algorithm of the CPU, and formulated the offloading decision problem as a minimization of the delay problem. Finally, the simulation experiments are conducted and the experimental results show that our proposal can reduce delay in the mobile edge computing environment and have a better performance of user device core usage. we focus on two problems in this paper-- How to allocate tasks between user devices and edge servers, and what scheduling algorithms are used in user devices.

**Related work(s)**

In recent years, many studies have begun to focus on task offloading of hybrid critical systems. Aiming at the offloading of hybrid key tasks in the CPS system(s), a framework is proposed to schedule them by analyzing **the**(their) deadline(s) and execution time of mixed-critical tasks[1]. These tasks are processed in parallel by using OpenMP. **At the same time**(meanwhile), the proposed framework (also) introduces machine learning-based predictions for task offloading in the cloud. Adiththan et al. [2] **propose**(proposed) an adaptive offloading **technology**(algorithm) suitable for control applications ~~is proposed~~, which makes all offloading decisions online based on a network performance monitor. However, ~~due to the enlightenment of this method~~ considering the method is based on heuristics, it is impossible to verify the timing behavior of the system. In addition, the author(s) mentioned that the loss(losses) of connectivity and large communication delays must be dealt with, and pointed out that in **this case**(such cases), the task execution must be redirected to the local system. Moreover, the potential consequences have not been discussed further. L.Schönberger et al. [3] designed two protocols for mixed mission-critical (tasks?systems) in unreliable environments, allowing the offloading of critical and non-critical tasks respectively. At the same time, the system level is changed when the operation of uninstalling critical tasks fails. **At this time**(besides), different service levels are provided for non-critical tasks to ensure that critical tasks can be completed before the worst execution time. Xu et al. [4] designed a software-defined task-critical wireless sensor network. The architecture is based on the idea of SDN architecture, combined with hierarchical cloud and edge computing technology, and is used for task offloading of mixed critical-level systems.

1. System Model
   1. Task Model

Mixed-critical system usually consists of multiple independent tasks. We assume (that) tasks set **as**(is) , and use a tuple to represent the tasks in :,where

* . represents critical level of the task
* represents the worst-case execution time (WCET) (related to the system critical level) of the task on (the) user device. ~~Tasks have different under different system critical levels~~
* represents the time required for the task execution on the edge server
* represents the period of (the) task
* represents relative deadline of

In order to reduce the complexity of the problem, ~~we only consider that the system has only high and low tow critical level case~~ we only consider the case that the system has just high and low critical levels, = {LO, HI}. For , and represent the worst-case execution time of two critical levels of low and high, where . Although the actual execution time of the task cannot be estimated, it does not exceed the WCET of the corresponding critical level. This paper adopts the implicit deadline, which is =.

According to the definition of critical level, the task(s) set can be divided into low critical task(s) set and high critical task(s) set, **namely**(denoted as) and ,where . At the same time, we also **made**(make) the following definitions:

**Definition 1:** The resource utilization rate of a task is:.Under different system critical (levels), tasks have different utilization rates. In the low critical execution mode, .**and**(as) in high critical execution mode

**Definition 2:** The sum of task utilization:

(1)

(2)

**Definition 3:** The task instance that each periodic task will be released after the time interval is called a job. The actual execution time of **the**(a) job is related to the actual running situation, so it changes with the actual execution situation.

* 1. User Device

~~We assumed that (the) user equipment is a mixed-critical system, and there are also multiple processors which each processor can~~ **~~execution~~**~~(execute) task(s) at the same time~~ we assume the user equipment is a mixed-critical system with multiple processors, which can execute tasks at the same time.

There are high critical tasks and low critical tasks in the set of periodic tasks that each processor in the system needs to schedule. When the execution time of a **high** critical (level) task exceeds its , then the critical level of the system needs to be **level**(levelled) up to ensure the (required) execution time ~~requirement~~ of the ~~high critical~~ task. **Corresponding to the mission-critical level is the system critical level(?)**. To ensure the robustness of the **mission**(system), when the system is in the low critical level state, ~~the high critical level and the low~~ both high and low critical level tasks can be executed; however, when the system is in a high critical (level) state, only high critical tasks can be executed.

* + 1. Critical Level Up

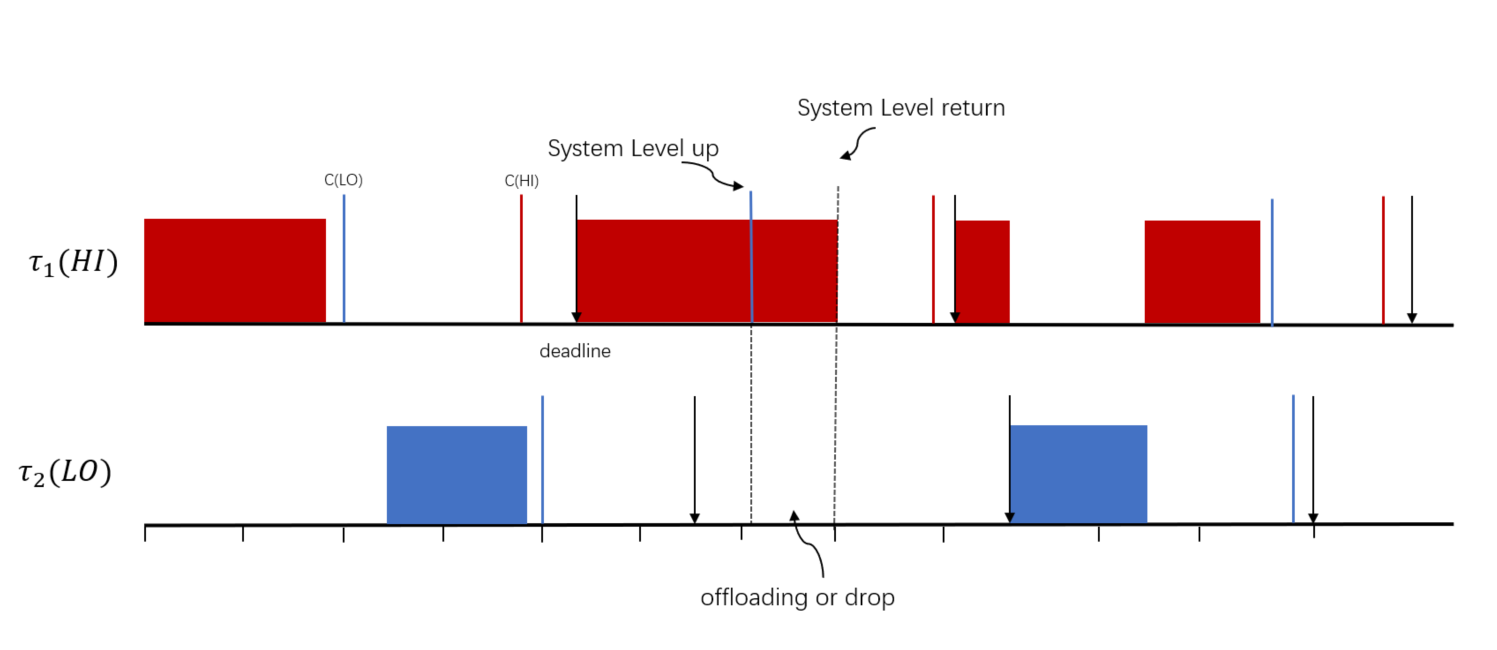
****

Figure 2: Critical Level up

The mixed-critical system cannot **actually**(precisely) predict ~~the time of~~ ~~system~~ when to level up**,(**.) **Which**(It) may ~~be due to the influence of~~  depend on external factors ~~of the system: for example~~, such as a car **~~enters a~~**(getting on the) highway from a city street, **and**(or) a military drone **flies(flying)** into a civil aviation area; it may also be due to high when the high critical task has completed its low-critical-level time budget, that is ,if the compute result is still not obtained, the critical level of the system will be level up and continue to be executed. The main focus of this paper is execution time as the key parameter. When the critical level of the system is increased, a typical approach is that the system discards all low critical operations. By offloading edge computing, low critical jobs can be offloaded to edge servers for execution when the critical level of the system level up, thereby reducing local **load**(offloading time) without affecting the execution of high critical tasks.

* + 1. Uniprocessor Scheduling Algorithm

**The task scheduling research on the uniprocessor platform is the earliest design of the mixed-critical scheduling research, and it is also the most studied content at present(?)**. EDF-VD [8], [9] **are(is)** currently **known**(regarded as) the best **performance**(performed) scheduling **algorithms**(algorithm) for mixed-critical tasks with implied deadlines on a single processor. In this paper, EDF-VD scheduling algorithm is used.

ALGORITHM 1: Base on PARTITION EDF-VD

A task(s) set with two critical degrees is scheduled on the preemptive processor per unit speed:

1. Calculate the deadline reduction factor for high critical **tasks**(task) 𝑥:

(?)

2.If，then calculate the virtual deadline for each high critical task，

3. Take the changed deadline as input and use the EDF algorithm to schedule the mixed-critical task(s) set 。

Jobs released by tasks are dynamically assigned (with) priorities **at runtime based on virtual deadlines(based on virtual deadlines at runtime).** Once the execution of a high critical task exceeds its **low critical WCET**(?), the system enters the high-critical execution mode, and performs EDF scheduling according to the actual deadline of the high-critical task. For low critical tasks, if the task(s) set is offloaded, the system will offload ~~the task~~ them to the edge server(s) for execution. If ~~it is~~ a low critical task is in the locally executed task(s) set, it will determine whether to discard the job or offload it to the edge according to the **scheduling algorithm service execution**(?).

When a task is allocated to the user device for execution, due to local resource constraints, the job released by the task will **enter(join)** the queue and wait to be executed. **The time from when the task is released is called the response delay, which is calculated according to the virtual deadline**(?). Response delay(s) of tasks in each processor:

(3)

Where represents the task with high priority

* + 1. Multiprocessor Scheduling Algorithm

Multiprocessor scheduling algorithms can be divided into two categories: global scheduling and divisional scheduling. In global scheduling, the whole system uses a unified task queue. In a partitioning scheduling algorithm, each processor has a separate queue of tasks, of which each task is assigned to a specific processor and runs on that processor only and does not migrate to other processors. **Dividing**(divisional) scheduling algorithms are more widely used on multiprocessor platforms of mixed-critical systems than global scheduling. The reason for choosing partition scheduling is to make better use of processors, as task migration between processors and unavailable cache information can lead to global scheduling showing more unpredictability.

|  |  |
| --- | --- |

Figure 3: global scheduling and partition scheduling

There are **more**(many) partition scheduling algorithms used on multi-processor platforms for mixed-critical systems than global scheduling algorithms. The reason for choosing partition scheduling is to make better use of processors, **because**(as) task migration between processors and unavailable cache information will cause global scheduling ~~to show~~(and ) more unpredictability. let represents processors set,**.**

When assigning high-critical tasks to each processor in a multi-processor system, the following needs to be met:

(4)

Need to meet when assigning low critical tasks

(5)

The assignment algorithm is as follows:

1. Assign high-critical tasks to processors in descending order of utilization and meet condition 4

2. Assign low-critical tasks to processors in descending order of utilization and meet condition 5

* 1. Edge Server

Since the edge server is close to the user equipment, **part**(some) of the tasks of the user equipment can be offloaded to the edge server to run. When performing edge offloading, the response time is composed of three parts: offloading delay, server execution time~~,~~ and result return delay. In general, the returned result is much smaller than the input data uploaded by the task. We assume that the task unloading delay is , and the execution result return delay is , then

(6)

There ~~is multiprocessor in~~ are multiple processors on the edge server, which can execute multiple tasks at the same time, and the task scheduling is arranged according to the scheduling strategy. Since the focus of this article is on local **resource constraints**(resource-constrained environment), the resource assumptions for edge tasks are sufficient, and the time delay for tasks in the queue is very short. And the global EDF algorithm is used for edge server ~~schedule~~ scheduling. ~~It should be noted that the literature [6] pointed out that the global EDF schedulable needs to satisfy, where M means that the edge server has how many cores.~~ As pointed out by [6], when using the global EDF scheduling algorithm, we should manage to guarantee , where M is the number of the cores of the edge server.

After the task is assigned, the schedulable condition must ~~be met~~ meet:

(7)

1. offloading strategy

For the periodic tasks, we propose an offline ~~offloading task allocation~~ tasks offloading algorithm, which can ~~allocate~~ offload the execution of the task to the corresponding equipment in advance before the execution of the task. **When the task releases the job, all the jobs ~~which~~ released ~~by~~ and assigned to the edge server execution task(s) set will be ~~offloading~~ offloaded to the edge server; As for the jobs released by the tasks and assigned to the user device, the jobs released by the high-critical tasks and the tasks released when the system level is at the low-critical level will be calculated on the CPU of the user device. When the system level is upgraded, low critical jobs will try to be offloaded to the edge server for computing through a wireless connection, and then the results will be returned to the user device after the execution is completed**(建议将整句的中文进行翻译).

The simulated annealing algorithm is a random optimization algorithm based on Monte-Carlo iterative solution strategy. We propose an algorithm for Offloading Decision Optimization Based on Simulated Annealing (ODSA). Take the offloading decision vector as the solution space,

The decision vector is used to allocate the task(s) to be executed on the user device or on the edge server. When ,the job released by the task is executed on the user device, and when , the job released by the task will be ~~unloaded Execute~~ uploaded to the edge. The initial solution is randomly generated. We use the running delay of the mixed-critical system in the low critical mode as the objective function in the simulated annealing process:

(9)

The Evaluation function is: =

ALGORITHM 2: Offloading Decision Optimization Algorithm Based on Simulated Annealing

init low critical task set

init the randomly

**while** execution on temperature level is not reached **do**

generate form

random select in , and change decision of

**if** meet condition 4,5 and 8 **then**

=

**if**  **then**

**else**

randomly generate 𝜌 ∈ (0,1)

**if** 𝜌 < **then**

**end if**

**end if**

𝑡 = 𝛼 × 𝑡

**end while**

1. Simulation

In this section, we use Python simulation to evaluate the performance of the proposed edge computation **offload** offloading scheme.

we take the processor(s) (m?),and .The task period is generated ~~according to the~~ by lognormal distribution. The period of the smallest task and the largest task are 100 times different. Hard real-time tasks in most embedded applications have such attributes. Task deadline . we ~~using~~ use Uunifast [7] algorithm to generate the utilization value of each task at a low critical level, thereby ensuring that the unbiased **distribution**(estimation?) of the value is added to the target utilization of the system. Then let ，. Among them, **CF**(cf?) is the key factor. Finally, the best execution time (BCET) is randomly selected between 80% and 100% of C(LO).

For the actual execution time of the job, if the job is released by a low-critical (level?) task, the actual execution time follows the normal distribution of . If the job is a high-critical (level?) task release, ~~first generate~~ we use a random variable of [0,1]. If the value of this variable is less than , then the actual time of the job is set to the range [C(LO), C(HI)] uniform distribution function, otherwise set to [BCET,C(LO)]. For the task execution time on the server side is set to

we set cf=2.0 and cp=0.5, and set uniform random [0.2,1.6]

|  |  |
| --- | --- |
| figure 4 Total Delay | figure 5 user device avege cpu usage |

we set cf=2.0 and cp=[0.1,0.8],and set 1.0

1. conclusion

In a mixed-critical system, ~~various~~ task(s) ~~that having~~ with different levels of ~~critically~~ criticality~~, which~~ cannot all be executed ~~in~~ on the local device because of the limited performance of the hardware. We proposed a strategy for offloading **period**(periodic) task(s) by offline divide task set using SA algorithm. **we design the schedule algorithm in user device, for assignment to edge server tasks will offloading to edge execution; for assignment to user device tasks, the execution mode will be selected according to the critical level of the system**(建议使用完整的中文语句进行翻译).

ACKNOWLEDGMENTS

REFERENCES

1. Atul Adya, Paramvir Bahl, Jitendra Padhye, Alec Wolman, and Lidong Zhou. 2004. A multi-radio unification protocol for IEEE 802.11 wireless networks. In Proceedings of the IEEE 1st International Conference on Broadnets Networks (BroadNets’04) . IEEE, Los Alamitos, CA, 210–217. https://doi.org/10.1109/BROADNETS.2004.8
2. Sam Anzaroot and Andrew McCallum. 2013. UMass Citation Field Extraction Dataset. Retrieved May 27, 2019 from <http://www.iesl.cs.umass.edu/data/data-umasscitationfield>
3. Martin A. Fischler and Robert C. Bolles. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM 24, 6 (June 1981), 381–395. https://doi.org/10.1145/358669.358692