IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 30, 2020, accepted February 16, 2020, date of publication February 28, 2020, date of current version May 8, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2977089

QoS-Aware Task Placement With Fault-Tolerance in the Edge-Cloud

HUAIYING SUN[1,2], HUIQUN YU[1,3], GUISHENG FAN[1], AND LIQIONG CHEN[4]

[1]Department of Computer Science and Engineering, East China University of Science and Technology, Shanghai 200237, China
[2]Shanghai Key Laboratory of Computer Software Evaluating and Testing, Shanghai 201112, China
[3]Shanghai Engineering Research Center of Smart Energy, Shanghai 200120, China
[4]Department of Computer Science and Engineering, Shanghai Institute of Technology, Shanghai 201418, China

Corresponding authors: Huaiying Sun (ecustshy@foxmail.com), Huiqun Yu (yhq@ecust.edu.cn), and Guisheng Fan (gsfan@ecust.edu.cn)

This work was supported in part by the NSF of China under Grant 61772200 and Grant 61702334, in part by the Shanghai Municipal Natural Science Foundation under Grant 17ZR1406900 and Grant 17ZR1429700, in part by the Educational Research Fund of ECUST under Grant ZH1726108, in part by the Collaborative Innovation Foundation of Shanghai Institute of Technology under Grant XTCX2016-20, and in part by the Humanities and Social Science Research Planning Fund of the Education Ministry of China under Grant 15YJCZH201.

ABSTRACT The geographically dispersed resources and ever-changing context incur unique heterogeneity, potential fragility, and vulnerability of an edge-cloud system. Thus, the reliability guarantee of services in the edge-cloud is critical. This paper firstly proposes a QoS-aware scheduling model with fault-tolerance in the edge-cloud, which extends the traditional primary-backup (PB) fault-tolerant model to improve the service reliability in the edge-cloud with the time constraints of tasks being satisfied. Then, a QoS-aware fault-tolerant scheduling algorithm including primary copy placement, backup copy placement and an adjustment mechanism is proposed to improve the QoS levels of tasks in the edge-cloud. The primary copy placement is to guarantee the earlier execution of the primary copy of a task to better satisfy the time requirements of tasks. The backup copy placement is to ensure the later execution of the backup copy of a task, reducing the overlapping of the two copies of a task, realizing the improvement of the resource utilization in the edge-cloud under the condition of redundancy and deadline requirements of tasks. The adjustment mechanism is triggered to rearrange the task copies of a computing node of the edge-cloud after the deallocation of a backup copy on the node, to better assist the goal-achievement of the primary and backup copy scheduling. Finally, through extensive simulation experiments with the real world taxi traces, the performance difference between the proposed method and the other four methods are evaluated. Results show that the proposed method generally outperforms the other methods in terms of guarantee ratio, average QoS level, and reliability cost.

INDEX TERMS Edge-cloud, QoS, fault-tolerance, time constraint, primary-backup.

I. INTRODUCTION

Edge-cloud has been widely deployed to host various applications and services because of its priority in providing resources with close distance and low latency to customers. It is promising in many important application areas, e.g., the ladder networking, robotics, energy efficiency management, predictive maintenance of rail transit equipment, energy network, intelligent transportation, smart city, military field and the industrial field such as the industrial manufacturing, etc [1]. Compared with the remote cloud data center, servers of the edge-cloud with more uncertain varying availability, credibility are more pone to failures [2]. The servers on the edge side are geographical and dispersed deployed, which incurs more complicated management and maintenance. Lacking of advanced supporting systems, for example, the complete backup electrical lines with transfer switches, diesel duplicated generators, clean agent fire suppression gaseous systems, and direct liquid cooling devices, will also increase the safety risks [3]. There are also some other hidden hardware and software failures in the edge-cloud [4]. Thus, the reliability guarantee of services in the edge-cloud is of great significance.

In addition, precision and reliability are more demanded in fields such as industrial manufacturing, predictive maintenance and the management of rail transit equipment, etc [5], [6]. For instance, Ossmann and Joos [7] studied an automated flight control system [8] which is utilized to fly a

The associate editor coordinating the review of this manuscript and approving it for publication was Honghao Gao.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

simulated model of a flight aircraft. All the flight controlling tasks of this system need to be completed within deadlines. For the purpose of improving the stability of the system, each task chooses different the quality of requirement (QoS) levels by varying its period or execution time, different QoS levels assure different flight qualities. Also, in [9], a real-time signal processing application needs different algorithms to deal with the massive generated signal data. For instance, extensive algorithms can be employed to decode block turbo codes [10], [11]. High-complexity algorithms can ensure a higher QoS level (higher data accuracy) of signal processing at the cost of processing time, low-complexity algorithms just operate the opposite [9].

Therefore, QoS-aware real-time system applications deployed on the edge-cloud should incorporate inherent high-reliability features [7], [9]. For example, the automated flight control system is adopted by the military battlefield, it must ensure that each task can be completed on time no matter whether there is a hardware or software fault or not [7]. For the case of the signal processing system, though the time requirement of it is not as rigid as the automated flight control system, the outdated or half-baked processed data may be useless for users, especially in the field of modern information battle [9]. Thus, the system must guarantee its functional and time correctness even in the presence of faults. Consequently, providing a fault-tolerant mechanism for such systems is of vital importance because of the inherent nature of tasks in these types of systems [2].

One of the effective ways to improve the service reliability is to design the fault tolerance based scheduling method [12]. The core of the fault-tolerant scheduling algorithm is to introduce redundancy to ensure that tasks can be completed smoothly even when permanent or transient system failures occur [13]. To the best of our knowledge, fault-tolerant scheduling in the edge-cloud environment is relatively rare, few works have been done on the fault-tolerant scheduling for real-time tasks with QoS requirements in the edge-cloud. To provide high system flexibility, Luo *et al.* [14] proposed a dynamic and reliability-driven real-time fault-tolerant scheduling method, DYFARS, in heterogeneous systems, considering both active and passive backup copies of tasks. However, the DYFARS does not consider the QoS requirements of real-time tasks when providing fault tolerance. Zhu *et al.* [15] proposed a QoS-aware fault-tolerant scheduling algorithm called QAFT to improve the QoS levels of real-time tasks, which is similar to the purpose of this paper. Wang *et al.* [16] presented a fault-tolerant elastic scheduling algorithm for real-time tasks in clouds named FESTAL. FESTAL takes virtualization into account and uses backup overlapping to realize high system utilization. Both QAFT and FESTAL don't have the adjustment mechanism to take full advantage of the primary and backup copy scheduling. Gao *et al.* [17] focused on improving performance in accessing and processing resources and providing resource security protection by using the cost difference of both type conversions of resources and traversing on resources in the IoT environment. Yin *et al.* [18] proposed a new matrix factorization model with deep features learning which combined a convolutional neural network. This model also contained a novel similarity computation method in order to improve the accuracy of neighbors selection. Yin *et al.* [19] focused on recommending the most suitable candidate from a huge number of available services for the recommendation task based on quality-of-service in a mobile edge computing environment by combining the model-based collaborative filtering and neighborhood-based collaborative filtering. Gao *et al.* [20] proposed a cost-driven services composition approach for enterprise workflows that adopts formal verification to recommend appropriate services for abstract workflows, ensuring that the configuration for the workflow solution has the best performance, high reliability, and low cost. Methods in [18]–[20] are proposed to recommend suitable services without fault tolerance mechanisms.

It is a challenge to design and implement novel QoS-aware fault-tolerant scheduling algorithms for real-time tasks whose application services are running on the edge-cloud with unique heterogeneity specifically. The effective utilization of resources in the edge-cloud under the condition of redundancy is also of great significance in enabling more tasks to be served in the edge-cloud smoothly. These challenges are the motivation to integrate fault tolerance with QoS-aware scheduling by developing a dynamic fault-tolerant scheduling algorithm based on the primary-backup strategy for real-time tasks in the edge-cloud. Thus, the contributions of this work are shown as follows.

(1) In view of the heterogeneous, distributed resources and the ever-changing context in the edge-cloud, we propose a novel QoS-aware scheduling model with fault-tolerance in the edge-cloud, which extends the traditional primary-backup (PB) fault-tolerant model [12] to improve the reliability of services of the edge-cloud.

(2)With the fault-tolerant scheduling model, a fault-tolerance based QoS-aware scheduling algorithm (FTBQA) including the primary copy placement, backup copy placement, and the adjustment mechanism is proposed to improve the QoS levels of tasks. The primary copy placement is to guarantee the earlier execution of the primary copy of a task to better satisfy the time requirements of tasks. The backup copy placement is to ensure the later execution of the backup copy of a task, reducing the overlapping of the two copies of a task, realizing the improvement of the resource utilization in the edge-cloud under the condition of redundancy and deadline requirements of tasks. The adjustment mechanism is triggered to rearrange the task copies of a computing node of the edge-cloud after the deallocation of a backup copy on the node, to better assist the goal-achievement of the primary and backup copy scheduling.

(3) Through extensive simulation experiments with the real world taxi traces in San Francisco, the performance difference between FTBQA and the other four benchmarks in terms

of guarantee ratio, average QoS level, reliability cost are evaluated.

The rest of the paper is organized as follows: Section II is about the related work. Then is the QoS aware scheduling model with fault-tolerance. Section IV shows the problem formulation. Then is the scheduling principles. Section VI shows the QoS-aware fault-tolerant scheduling algorithm, the following is the performance evaluation, and the last is the conclusion.

II. RELATED WORK

On one hand, fault-tolerant scheduling algorithms can be classified into two categories: preemptive scheduling [21]–[23] and non-preemptive scheduling [24]–[26]. In the preemptive scheduling method, the executing tasks can be preempted by other tasks. As for the non-preemptive scheduling approaches, the executing tasks cannot be interrupted during their executions and a task can start its execution only if the executing task before it has finished its execution [22]. Although preemptive scheduling is capable of achieving high system utilization, it is impossible or prohibitively expensive for hardware devices or software configuration to make preemptions in many practical scenarios [25]. In stark contrast, non-preemptive scheduling has the features of accurate response time analysis, ease of implementation, no synchronization overhead, and reduced stack memory requirements [26]. Non-preemptive scheduling has proven to be more beneficial than preemptive scheduling in many applications, such as multimedia applications [24].

From the other point of view, fault-tolerant scheduling algorithms can also be divided into two classes, one is the static method (i.e., offline) [25]–[27] and another is the dynamic method (i.e., online) [28]–[30]. The static fault-tolerant scheduling methods are suitable for periodic tasks [25], which will assign tasks to the computing nodes in advance and the starting time of a task is also required to be determined a priori [26], [27]. As for the aperiodic tasks which always arrive randomly should be scheduled by the dynamic fault-tolerant scheduling methods [28], [29]. With the increase of applications requiring high real-time performance, more and more studies pay attention to the dynamic fault-tolerant scheduling algorithms based on the primary backup model (or PB in short). In the PB model, two copies of a task, namely, the primary copy and the backup copy, are allocated to the two different nodes. There is also an acceptance test in the model, which is adopted to check the correctness of allocations [12].

The backup copy of a task can have two alternative schemes, one is the passive backup-copy scheme [12], the other is the active backup-copy scheme [28]. For the passive backup-copy scheme, the real-time tasks should have enough laxity to restart their backup copies [29], and the backup copy of a task is allowed to execute only when a fault occurs in the primary task [30], [31]. Ghosh *et al.* [30] proposed two approaches called deallocation and overloading to improve schedulability and provide fault tolerance with low overhead. The problem is that multiple backup copies in the overloading scheme may overlap in the same time slot on the same processor. The deallocation scheme was used to reclaim the resources reserved for backup copies once the corresponding primary copies have been completed successfully [30]. Manimaran and Murthy [31] replenished the method of [30] by considering resource constraints among tasks and partitioning processors into groups to tolerate more than one failure at a time. Al-Omari *et al.* [32] focused on a PB overloading technique that allowed the primary copy of a task to overlap with the backup copy of another task to ensure the high schedulability. There is an important assumption in these studies that the laxity of a task must be at least twice as large as its computation time so that the passive backup-copy scheme can be adopted [12]. However, this assumption is not realistic in practice, i.e., for the case of the heavily loaded real-time systems. Different from the passive backup-copy scheme, the active backup-copy scheme is sufficient for the tasks with small laxities. For instance, Tsuchiya *et al.* [33] proposed a method in which two copies of each task were concurrently executed with different start times. Yang *et al.* [34] proposed a fault-tolerant scheduling method in which the two copies of a task were executed simultaneously for the purpose of improving the schedulability. Al-Omari *et al.* [35] focused on the adaptive scheme which managed the overlap interval between the primary copy and backup copy of a task according to the primary-fault probability and task's laxity.

There are some other new approaches are proposed to improve the performance of services. Zhang *et al.* [36] firstly used a strategy based on the density of Internet of Things (IoT) devices together with the k-means algorithm to divide network of edge servers, then proposed an algorithm for making IoT devices' computation offloading decisions. This method showed great performance in reducing global cost, but it didn't take the fault-tolerance into account. Ghahramani *et al.* [37] carried out a comprehensive survey on the models proposed in the literature regarding the implementation principles to address the QoS guarantee issue. Qi *et al.* [38] proposed a novel time-aware and privacy-preserving service recommendation approach based on the Locality-Sensitive Hashing technique by extending the traditional Locality-Sensitive Hashing technique to incorporate the time factor. This approach achieves a good tradeoff between recommendation accuracy and efficiency with the guarantee of privacy-preservation. Li *et al.* [39] proposed a secure random key distribution scheme to defense against the node replication attacks. The approach has great security and effectiveness while also has good storage and communication efficiency. Gao *et al.* [40] studied a communication scheduling and remote estimation problem within a worst-case scenario that involved a strategic adversary. Zhang *et al.* [41] discussed and analyzed the architectures of fog computing, indicated the related potential security and trust issues. And Yuan *et al.* [42] proposed a time-aware task scheduling algorithm which investigated the temporal variation and scheduled all admitted tasks to execute in Green

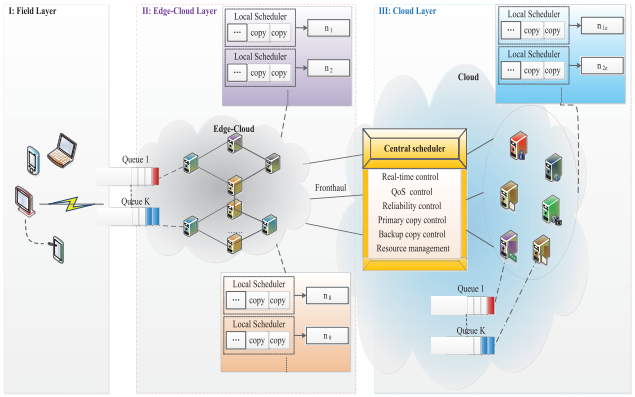


FIGURE 1. QoS-aware scheduling model with Fault-tolerance.

Data Center meeting their delay bounds. This work then was extended to the scenario of Green Hybrid Cloud [43]. Reference Kang *et al.* [44] proposed a real-time distributed load scheduling algorithm to solve an objective function that was based on constraints of power supply. Where a baseload forecasting model was established when aggregating renewable generation and non-deferrable load into a power system. To minimize the deployment cost of the cloudlet placement, Fan and Ansari [45] put forward a cost aware cloudlet placement in mobile edge computing strategy considering the cloudlet cost and average end-to-end delay and developed a Lagrangian heuristic algorithm to achieve the suboptimal solution. Meanwhile, a workload allocation scheme was designed to minimize the end-to-end delay between users and their cloudlets regarding the user mobility. To predict QoS values for service recommendation and service selection, Gao *et al.* [46] developed a holistic framework to attack the QoS prediction in IoT environment based on neural collaborative filtering and fuzzy clustering and designed a fuzzy clustering algorithm to cluster contextual information. Then a new combined similarity computation method and a new neural collaborative filtering model to leverage local and global features were proposed.

In this paper, we focus on the non-preemptive scheduling for the aperiodic and independent real-time tasks in the edge-cloud with respect to the QoS requirements of tasks. Our approach can also be applied to dependent tasks because tasks with precedence constraints could be considered as the independent tasks as long as the ready times and deadlines of dependent tasks are modified accordingly [47].

III. SYSTEM ARCHITECTURE AND COMPUTING MODELS

In this section, we will introduce the QoS aware scheduling model with fault-tolerance in the edge-cloud. It has three parts, as shown in Fig.1.

The first part I is the field layer which is close to the network having field nodes like the sensors, actuators, devices, control systems, and assets, etc [1]. These field nodes are connected with edge gateways and other devices in the edge-cloud through various types of field networks and industrial buses, so as to realize the connection of data flow and control flow between the field layer and the edge-cloud. Assume that there are K types of applications in the system, their tasks will be inserted into corresponding task queues to be processed, for example, the queues in Fig.1. As for the types of tasks for a specific application, there are two cases: if a task needs real-time services, it will be processed in the edge-cloud without the intervention of the remote cloud. In contrast, if a task demands the intervention of the remote cloud for analysis based on historical data-sets and for semi-permanent or permanent storage, it will be sent to the remote cloud after being processed in the edge-cloud [48]. Thus, both edge-cloud and cloud will maintain these K types of task queues.

The second part II is the edge-cloud layer which is the core of the whole architecture. It receives, processes and forwards data streams from the field layer, and provides time-sensitive services such as intelligent perception, security and privacy protection, data analysis, intelligent computing, process optimization, and real-time control. The edge-cloud includes distributed devices with computing and storage capabilities such as edge gateways, edge controllers, edge servers, edge sensors and network devices such as time-sensitive network switches or routers encapsulating computing, storage and network resources on the edge side. The edge-cloud also includes the edge manager software, which mainly provides the ability of business choreography or direct invocation to control various edge nodes to complete tasks.

The third part III is the remote cloud layer, which provides a decision support system, application service programs in specific fields such as intelligent production, network collaboration, service extension, and personalized customization, and provides interfaces for end-users. The remote cloud layer receives the data stream from the edge-cloud and sends control information to the edge layer and field layer through the edge layer. It optimizes resource scheduling and industrial production process in a global scope. The centralized scheduler in it is in charge of real-time controlling, QoS controlling, reliability controlling, primary copy controlling, backup copy controlling and the resource management deciding how nodes should be added or migrated if the current processing capacity is unable to meet the time requirements.

When a new task arrives, with the requirement of the task and the resource information gathered from all computing nodes of the edge-cloud and the remote cloud data center, the centralized scheduler makes decisions according to the corresponding scheduling algorithm (we will discuss it in Section VI) and the primary and backup copies of a task will be sent to the different computing nodes based on the decisions. Then the primary copy is executed if the node is idle, or waits in the local queue if the node is busy. When the primary copy is finished successfully, the backup copy is deleted and the resource occupied by the backup copy is reclaimed, and the adjustment mechanism will be triggered to rearrange the task copies of a computing node of the edge-cloud after the deallocation of a backup copy on the

node, to better assist the goal-achievement of the primary and backup copy scheduling. The local scheduler is in charge of rearranging the order of the local queue if any backup copy is removed from the node.

IV. PROBLEM FORMULATION

The application instances running on the terminal devices will have their task requests sent to the edge-cloud to be served. As for the type of tasks for corresponding applications, there are two cases: if a task needs real-time services, it will be processed in the edge-cloud without the intervention of the remote cloud. In contrast, if a task demands the intervention of the remote cloud for analysis based on historical datasets and for semi-permanent or permanent storage, it will be sent to the remote cloud after being processed in the edge-cloud [48]. To improve the service reliability of the edge-cloud through the way of scheduling based on the primary-backup strategy, each task sent to the edge-cloud has two copies, namely the primary copy and the backup copy. The two copies of the task are firstly considered to be allocated to the edge-cloud, if the edge-cloud refuses the primary/backup copy of a task, then the copy will be further considered to be assigned to the remote cloud data center. The detail of the task assignment process will be introduced in section VI.

The service delay of a task request sent by the terminal device is the corresponding response latency, which is the sum of the data transmission delay, the queuing and processing delay of the request on the computing node. The bottleneck of the transmission is the bandwidth between the terminal device and edge-cloud, the bandwidth between the edge-cloud to the remote cloud data center. As the number of active terminals increases, the amount of data generated will increase and the corresponding service delay maybe also increase. In the following, we are going to introduce the models of transmission latency, queuing delay, processing delay and the reliability cost which is used to measure the reliability of the whole computing system in a time slot [23], [49].

A. TRANSMISSION DELAY ON THE EDGE-CLOUD AND THE REMOTE CLOUD

Let ω_{ee}^{v} be the transmission delay of an unit byte data from the terminal device which is in the source point v to the edge-cloud, ω_{ec}^{v} be the unit byte data transmission delay from the edge-cloud to the remote cloud. $P_r^v(x_i,t)$, $P_s^v(x_i,t)$ are the total amount of data in bytes which are generated from the source point v in the time slot t which demands to be served and stored for the i^{th} task x_i [48]. $Q_r^v(x_i,t)$, $Q_s^v(x_i,t)$ are the total number of bytes generated from v, which will be transmitted to the remote cloud for computation and storage purposes. For the part $P_r^v(x_i,t)+P_s^v(x_i,t)-Q_r^v(x_i,t)-Q_s^v(x_i,t)$ of the task request x_i which is to be served and stored on the edge-cloud, the corresponding transmission delay is expressed as Equation (1). Where V represents the number of source points which generate the data of x_i. If task x_i just needs to be served and stored on the edge-cloud, then both $Q_r^v(x_i,t)$, $Q_s^v(x_i,t)$ are 0. Similarly, if the leftover part $Q_r^v(x_i,t)+Q_s^v(x_i,t)$ of x_i is to be served and stored on the remote cloud, then the transmission time is given by Equation (2).

$$\delta_{i,tr}^{e}(x_i,t)=\sum_{v=1}^{V}\omega_{ee}^{v}\{P_r^v(x_i,t)+P_s^v(x_i,t) - Q_r^v(x_i,t)-Q_s^v(x_i,t)\} \tag{1}$$

$$\delta_{i,tr}^{c}(x_i,t)=\sum_{v=1}^{V}(\omega_{ee}^{v}+\omega_{ec}^{v})\{Q_r^v(x_i,t)+Q_s^v(x_i,t)\} \tag{2}$$

Thus, no matter whether the task x_i would just be processed in the edge-cloud or be processed in the edge-cloud with the interference of the remote cloud, the average transmission time at the time slot t can be concluded as Equation (3) and Equation (4).

$$\Lambda_{tr}^{e}(t) = \frac{\delta_{i,tr}^{e}(x_i,t)+\delta_{i,tr}^{c}(x_i,t)}{\sum_{v=1}^{V}\{P_r^v(x_i,t)+P_s^v(x_i,t)\}} \tag{3}$$

$$\delta_{i,tr}^{e}(x_i,t)+\delta_{i,tr}^{c}(x_i,t) = \sum_{v=1}^{V}\omega_{ee}^{v}\{P_r^v(x_i,t)+P_s^v(x_i,t)\}+\omega_{ec}^{v}\{Q_r^v(x_i,t)+Q_s^v(x_i,t)\} \tag{4}$$

For the case that all part of the task x_i is processed on the remote cloud data center, the total transmission time and the average transmission time at the time slot t is given by Equation (5) and Equation (6).

$$\delta_{tr}^{c}(t)=\sum_{v=1}^{V}(\omega_{ee}^{v}+\omega_{ec}^{v})\{P_r^v(x_i,t)+P_s^v(x_i,t)\} \tag{5}$$

$$\Lambda_{tr}^{c}(t)=\frac{\sum_{v=1}^{V}(\omega_{ee}^{v}+\omega_{ec}^{v})\{P_r^v(x_i,t)+P_s^v(x_i,t)\}}{\sum_{v=1}^{V}\{P_r^v(x_i,t)+P_s^v(x_i,t)\}} \tag{6}$$

B. QUEUING DELAY AND PROCESSING DELAY ON THE EDGE-CLOUD AND THE REMOTE CLOUD

Compared with the remote cloud, resources on the edge side are limited. Inspired by [50], the queuing system is adopted for the edge-cloud. Let $Qu_j(t)$ be the number of task requests queuing in the edge node j at the beginning of slot t. Then the changing of queue length Qu_j is denoted by Equation (7).

$$Qu_j(t+1)=\max[Qu_j(t)+Yu_j(t)-ru_j(t),0] \tag{7}$$

where $Yu_j(t)=\sum_{m\in M}Yu_j(t)(m)$ is the number of task requests arrived, M represents the set of tasks are determined to compute in the edge node j. $ru_j(t)$ is the number of task requests serviced at the edge node j during slot t. Let $Qw_j(t)$ be the corresponding workload with respect to the number of

task requests and task sizes queuing in the edge node j at the beginning of slot t. So, it is denoted by Equation (8).

$$Qw_j(t+1) = \max[Qw_j(t) + Yw_j(t) - P_j^F/X, 0] \quad (8)$$

where P_j^F/X is the amount of data that the edge server j can process in a time slot, P_j^F is proportion to the server's CPU-cycle frequency, X is the number of CPU cycles needed to process a bit of a task [50]. $Yw_j(t) = \sum_{x=0}^{Yu_j(t)-1} S_j^x$ means the aggregated workload in the time slot t, and S_j^x shows the size of the x^{th} task request on the edge node j that arrives in time slot t. Then the queuing delay of the x^{th} task request on the edge side can be shown as Equation (9). Where $\sum_{m=0}^{x-1} S_j^m$ represents the workload that arrives ahead of the x^{th} job in time slot t [50].

$$Qd_{(j,x)}(t) = \frac{X(Qw_j(t) + \sum_{m=0}^{x-1} S_j^m)}{P_j^F} \quad (9)$$

The total processing time $\delta^e_{i,j,pr}$ of the part of task x_i which is assigned to the node j in edge-cloud is equal to the sum of queuing time and the time needed to analyze the accumulated data in a specific time slot t. As shown in Equation (10), ϕ_y^e is the weight-factor related with the set of data which is required for analysis, and the magnitude of it (within(0,1]) decreases with the increase of the staying time of the data [48], [51]. If $y = t$, then $\phi_y^e = 1$. The expression $P_r^v(x_i,t) - Q_r^v(x_i,t)$ denotes the total amount of data which is stored in the edge-cloud at time t for analysis and computation purposes.

$$\delta^e_{i,j,pr}(x_i,t) = (P_r^v(x_i,t) - Q_r^v(x_i,t))\zeta_j^e + \sum_{y=t-\tau}^{t} \phi_y^e \sum_{v=1}^{V} \{P_s^v(x_i,y) - Q_s^v(x_i,y)\} + Qd_{(i,x)}(t) \quad (10)$$

Accordingly, the resources on the remote cloud are relatively more sufficient than the edge side, so the queuing delay is ignored. The processing time of the part of the task x_i which is served on the cloud can be given by Equation (11). Where ζ_j^e, ζ_j^c are the unit byte data processing time of the computing node j on the edge-cloud and the remote cloud data center respectively. Different nodes have different processing capabilities, which shows the heterogeneity of nodes in the edge-cloud and the remote cloud data center. The average processing latency of task x_i on the edge-cloud with or without the interference of the remote cloud can be seen as Equation (12).

$$\delta^c_{i,j,pr}(x_i,t) = Q_r^v(x_i,t)\zeta_j^c \sum_{y=t-\tau}^{t} \phi_y^c \sum_{v=1}^{V} Q_s^v(x_i,y) \quad (11)$$

$$\Lambda^e_{pr}(t) = \frac{\delta^e_{i,j,pr}(x_i,t) + \delta^c_{i,j,pr}(x_i,t)}{\sum_{v=1}^{V} \{P_r^v(x_i,t) + P_s^v(x_i,t)\}} \quad (12)$$

If all part of the task x_i is served on the remote cloud, then the processing latency of task x_i and the average processing latency at the time slot t on the remote cloud can be seen as Equations (13)-(14).

$$\delta^c_{pr}(t) = P_r^v(x_i,t)\zeta_j^c \sum_{y=t-\tau}^{t} \phi_y^c \sum_{v}^{V} P_s^v(x_i,y) \quad (13)$$

$$\Lambda^c_{pr}(t) = \frac{P_r^v(x_i,t)\zeta^c \sum_{y=t-\tau}^{t} \phi_y^c \sum_{v}^{V} P_s^v(x_i,y)}{\sum_{v=1}^{V} \{P_r^v(x_i,y) + P_s^v(x_i,y)\}} \quad (14)$$

Thus, the average service time at the time slot t for task x_i in the edge-cloud with or without the inference of the remote cloud can be given by Equation (15). The average service time at the time slot t for task x_i which is totally processed by the remote cloud is shown as Equation (16).

$$\Lambda^e_{sr}(t) = \Lambda^e_{tr}(t) + \Lambda^e_{pr}(t) \quad (15)$$

$$\Lambda^c_{sr}(t) = \Lambda^c_{tr}(t) + \Lambda^c_{pr}(t) \quad (16)$$

C. RELIABILITY COST MODEL

The scheduling model in this paper is based on the primary and backup fault-tolerant model [12]. For a new arriving task x_i, two copies are corresponding to it: the primary copy x_i^P and the backup copy x_i^B, which need to be allocated. According to the time-related models described above, we will introduce the reliability cost model of tasks.

The reliability model is used to evaluate the fault tolerance level of the system [12], [15]. Reliability is defined as the probability that no task will fail even if there is a hardware or software failure. Reliability cost is a very important index for system reliability. In order to describe the task reliability cost based on PB fault-tolerant mechanism in this paper, we have improved the reliability model in [12], [14] which does not reflect the task's QoS requirements, nor does it consider the reliability cost of the backup copy of a task in different implementation scenarios under PB technology. The status setup of the backup copy of a task can be referred to the Property 1 in Section V. Equation (17) represents the reliability cost model of the primary copies. λ_i is the failure rate of node n_j. $z_{ij} = 1$ denotes that task x_i^P is assigned to the node n_j of the edge-cloud, otherwise $z_{ij} = 0$. $q(x_i^P)$ represents the QoS level that a task can obtain when it assigned to the node n_j, and $\delta_{ij}(q(x_i^P))$ is the service time of task x_i^P. $o_{ij}^P = 1$ indicates that the task is successfully executed on the node n_j, otherwise $o_{ij}^P = 0$. bT is a positive real number.

$$rc(X^P) = \sum_{j=1}^{m}\sum_{i=1}^{n} \lambda_j z_{ij}^P o_{ij}^P \Delta_{ij}^P \quad (17)$$

$$\Delta_{ij}^P = \Delta_{ij}^P(q(x_i^P)) = q(x_i^P) \times bT \times \delta_{ij}(q(x_i^P)) \quad (18)$$

The reliability cost model of backup copies is shown as Equation (19) [15]. Where $r\Delta_{ij}^B$ denotes the actual service time of the backup task on the node n_j, which is not only related to the execution scheme of the task, but also the execution result of its corresponding primary copy. As shown

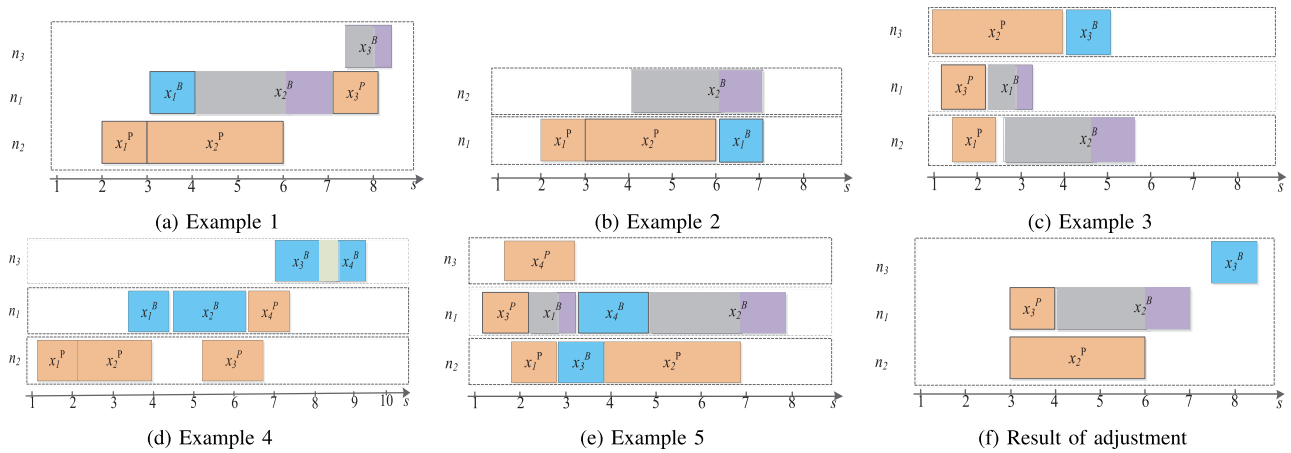


FIGURE 2. Illustration examples.

in Equations (19)-(21), if the primary copy x_i^P corresponding to x_i is successfully executed and x_i^B adopts the passive execution mechanism, that is, $o_{ij}^P = 1 \wedge s(t_{ij}^B) = 0$, then $r\Delta_{ij}^B = 0$. If $r\Delta_{ij}^B$ uses the active strategy, the real service time of it is equal to the time period from its starting time to the finish time of its corresponding primary copy. If x_i^P fails, e.g. the corresponding node fails, then only the successful execution of the corresponding backup copy can assure the completion of x_i, then $r\Delta_{ij}^B = \Delta(q(x_i^B))$.

$$rc(X^B) = \sum_{j=1}^{m} \sum_{i=1, o_{ij}^P=1, s(X_{ij}^B)=1}^{n} \lambda_j z_{ij}^B o_{ij}^B r\Delta_{ij}^B + \sum_{j=1}^{m} \sum_{i=1, o_{ij}^P=0}^{n} \lambda_j z_{ij}^B o_{ij}^B \Delta_{ij}^B \tag{19}$$

$$r\Delta_{ij}^B = \begin{cases} 0 & if\ o_{ij}^P = 1 \wedge s(x_{ij}^B) = 0 \\ (0, \Delta_{ij}(q(x_i^B))] & if\ o_{ij}^P = 1 \wedge s(x_{ij}^B) = 1 \\ \Delta_{ij}(q(x_i^B)) & if\ o_{ij}^P = 0 \end{cases} \tag{20}$$

$$\Delta_{ij}^B = \Delta_{ij}(q(x_i^B)) = q(x_i^B) \times bT \times \delta_{ij}(q(x_i^B)) \tag{21}$$

Based on the above models, we define the reliability cost of a set of tasks in a certain period of time as shown in Equation (22) [15], and the reliability r of a cluster of nodes corresponding to the set of tasks X is shown as Equation (23).

$$rc = rc(X^P) + rc(X^B) \tag{22}$$

$$r = e^{-rc} \tag{23}$$

The purpose of this paper is to maximize the QoS levels of all accepted tasks under time constraints, as shown in Equation (24). Where $\sum_{j=1}^{m}\sum_{i=1}^{n}(z_{ij}^P q(x_i^P)o_{ij}^P) + \sum_{j=1}^{m}\sum_{i=1}^{n}(z_{ij}^B\, q(x_i^B)o_{ij}^B)$ represents the sum of the QoS levels of all successfully executed tasks and $\sum_{j=1}^{m}\sum_{i=1}^{n}(z_{ij}^P o_{ij}^P + z_{ij}^B o_{ij}^B)$ is the number of successfully executed tasks [49].

$$MQ(X^P, X^B) = \max_{x_i^P \in X^P, x_i^B \in X^B} \frac{\sum_{j=1}^{m}\sum_{i=1}^{n} z_{ij}^P q(x_i^P) o_{ij}^P + \sum_{j=1}^{m}\sum_{i=1}^{n} z_{ij}^B q(x_i^B) o_{ij}^B}{\sum_{j=1}^{m}\sum_{i=1}^{n}(z_{ij}^P o_{ij}^P + z_{ij}^B o_{ij}^B)} \tag{24}$$

V. SCHEDULING PRINCIPLES FOR THE PRIMARY AND BACKUP COPIES PLACEMENT IN THE EDGE-CLOUD

Considering the realistic requirements in the edge-cloud environment, based on [12], [15], [49], [51], this section is going to introduce some significant properties and theorems before explaining the proposed primary-backup scheme based QoS-aware task scheduling method.

Property 1: Each backup copy has two alternative states: active and passive, denoted by 0 and 1 respectively. The state of each backup copy is set based on the finish time of its primary copy. Let $s(x_{ij}^B)$ denote the state of backup copy x_i^B on the node n_j, then the state of x_i^B when it is assigned to the node n_j can be seen as Equation (25).

$$s(x_{ij}^B) = \begin{cases} 1, & if\ (f_i^P > s_{ij}^B) \\ 0, & otherwise \end{cases} \tag{25}$$

where f_i^P is the finish time of the primary copy x_i^P, if it is greater than the start time s_{ij}^B of backup copy x_i^B on the node n_j, then the state of x_i^B will be set as active, otherwise passive. For example, in Fig.2(a), the finish time f_1^P of the primary copy x_1^P is 3. It is smaller than the starting time of x_1^B, which is larger but smaller than 4. so the state of x_1^B can be set as passive. However, the finish time of x_2^P is greater than the start time of x_2^B, the state of x_2^B on the node n_1 should be set as active. (The areas in orange represent the primary copies, the blue areas are the backup copies in the passive state, the areas with gray and purple colors are the backup copies in the active state. The gray areas mean that in the period, the primary copy

and backup copy of a task are processed simultaneously on two different nodes but at the different phases of processing.)

Property 2: The QoS level of a task copy is decided by the start time, service time and finish time of it, which is shown as Equations (26) and (27). Where d_i is the deadline of task x_i. est_{ij}^P is the earliest start time of the primary copy x_i^P for task x_i on the node n_j. lst_{ik}^B is the latest start time of the backup copy x_i^B for task x_i on the node n_k.

$$\forall x_i \in X, \delta_{ij}(q(x_i^P)) \leq d_i - est_{ij}^P \tag{26}$$

$$\forall x_i \in X, \delta_{ik}(q(x_k^B)) \leq d_i - lst_{ik}^B \tag{27}$$

Property 2 shows that the primary and backup copies of a task can be accepted only if these two conditions are satisfied. The expected QoS levels for the primary and backup copies of a task can be different. This will to some degree improve the system flexibility. This is also the feature of it compared with the traditional fault-tolerance scheduling which will refuse a task if the primary and backup copies of it can not get the same QoS level guarantee. Thus the flexible QoS level selection for the primary and backup copies of tasks can improve the schedulable capability of tasks.

Property 3: The earliest start time est_{ij}^P of the primary copy x_i^P of task x_i on the node n_j should be subject to the following constraints:

(1)The corresponding node n_j has the idle slot to accommodate the primary copy x_i^P.

(2)The finish time of a primary copy should be smaller than or equal to the deadline of the task, i.e, $f_i^P \leq d_i$.

Assume that there are totally N primary and backup copies which have been assigned to node n_j, the occupied time slot is $[a_i, b_i]$, namely, $[s_{i1}, f_{i1}], \ldots, [s_{i(N-1)}, f_{i(N-1)}], [s_{iN}, f_{iN}]$. s_{in} and f_{in} are the start and finish time of the number n task, $(1 \leq n \leq N)$, and $a_i \leq s_{i1} \leq s_{i2} \ldots \leq s_{iN} \leq f_{iN} \leq b_i$. The primary copy of a task can not overlap with any other task copies, so these time slots are not available to x_i^P. Therefore, it is necessary to traverse these time slots from left to right to find the minimum index value k which subjects to $s_{i(k+1)} - max\{a_i, f_{ik}\} \geq \delta_{ij}(q(x_{ij}^P))$, that is, the earliest start time of primary copy x_i^P is $est_{ij}^P = max\{a_i, f_{ik}\}$.

As for the backup copy x_i^B, the unavailable time slots for it on node n_j is the occupied time slots by primary copies and the active backup copies. Thus, suppose that the unavailable time slots for x_i^B is $[s_i^P, d_i]$, namely, $[s_{i1}, f_{i1}], \ldots, [s_{i(N-1)}, f_{i(N-1)}], [s_{iN}, f_{iN}]$, and $s_i^P \leq s_{i1} \leq s_{i2} \ldots \leq s_{iN} \leq f_{iN} \leq d_i$. To find the latest start time of a backup copy, we can traverse the time slots from right to left, the largest index k which subjects to $s_{i(k+1)} - max\{s_i^P, f_{ik}\} \geq \delta_{ij}(q(x_{ij}^B))$ can be selected. Then, the latest start time of x_i^B is given by $lst_{ij}^B = s_{i(k+1)} - \delta_{ij}(q(x_{ij}^B))$.

Theorem 1: For any given $x_i \in X$ and n_k, if $n(x_i^P) = n_k$, then, $n(x_i^B) \neq n_k$.

Proof: Assume that $n(x_i^P) = n(x_i^B) = n_j$, if node n_j encounters a fault, both the primary and backup copies of task x_i are on the node n_j, then neither of the copies can assure the successful execution of task x_i, which will disobey the purpose of fault tolerance. As the example 2 in Fig.2(b), x_1^P and x_2^P are primary copies of task x_1 and x_2 respectively, both of them are assigned to node n_1, then the corresponding backup copies of them only can be assigned to the nodes except node n_1. As we can see, the assignment of x_2^B shown in the Fig.2(b) is reasonable but that of x_1^B is illegal. ■

Theorem 1 shows that the backup copy and primary copy of a task can not be assigned to the same node for the sake of fault tolerance.

Theorem 2: For any given n_k, x_i^P and x_j^B with $i \neq j$ are assigned to n_k, there are two cases.

Case 1: if $s_{ik}^B < s_{jk}^P$, then, the two copies of two different tasks subject to the constraint that $f_{ik}^B < s_{jk}^P$.

Case 2: if $s_{ik}^B > s_{jk}^P$, then, the two copies of two different tasks subject to the constraint that $f_{ik}^B > s_{jk}^P$.

Proof: Assume that $n(x_i^B) = n(x_j^P) = n_k$, for case 1, if $s_{ik}^B < s_{jk}^P$, if we adopt the overlapping mechanism to deal with these two copies, then we can change the constraint $s_{ik}^B < s_{jk}^P$ to $s_{ik}^B <= s_{jk}^P$, and the finish time of x_i^B can subject to $f_{ik}^B > s_{jk}^P$. If in the time period of $f_{ik}^B - s_{jk}^P$, the primary copy x_i^P of x_i encounters a fault when executing on the assigned node, the backup copy of task x_i must be at the common processing state either keeping on the execution process (if it is at the active state initially) or turning into active state from the passive state (if it is at the passive state initially). However, the overlapping of $x_i^B = x_j^P$ will incur a conflict between them leading to neither of them can successfully execute on the corresponding node. For the illustration of case 2, the inference is similar to case 1. As the example 3 in Fig.2(c), $s(x_3^B) = 0$, it can not overlap with x_2^P which is also on the node n_3. It is because that if $s_{33}^B < f_{23}^P$, the x_3^B will compete with x_2^P if there is a fault that occurs when x_3^P is still on its execution. Because $s(x_1^P) = s(x_2^P) = 1$, they can not be processed simultaneously on a node, not even overlap with the previous unfinished primary copy. ■

Theorem 2 shows that the backup copy on a node can not overlap with any other primary copies on the node.

Theorem 3: For any given x_i^P and x_j^P with $i \neq j$, if $n_i^P = n(x_j^P) = n_k$, $n_i^B = n(x_j^B) = n_t$, and $s_{it}^B < s_{jt}^B$, then, $f_{it}^B < f_{jt}^B$.

Proof: $n(x_i^P) = n(x_j^P) = n_k$, $n(x_i^B) = n(x_j^B) = n_t$, x_i^B overlaps x_j^B on the node n_t, when x_i^P is still on its execution and suddenly the node n_k fails, in order to ensure the completion of tasks x_i and x_j, whether x_i^B and x_j^B are in active states or not, they should be finally adjusted to be in the active states. Because the two copies overlap on the node n_t, there will be a conflict between them. As the example 4 of Fig.2(e), x_1^P and x_2^P are allocated to node n_2, x_1^B and x_2^B are allocated to node n_1. Although both x_1^B and x_2^B are in passive states, they can not overlap on the node n_1. This is because their primary copies are on the same node. If the node fails during the execution of x_1^P, the execution of x_i and x_j will totally depend on x_1^B and x_2^B. If x_1^B and x_2^B have a conflict, neither of them can be completed on time. A contrary example is the x_3^P

and x_4^P, they are primary copies of tasks x_3 and x_4 respectively. They are assigned to nodes n_2 and n_1 respectively. If their backup copies are assigned to nodes n_3, their backup copies can overlap on the node n_3. Because no matter what happens x_3 and x_4 can always have a successful execution. That is, $f_{33}^B > s_{43}^B$ shown in the common gray part of x_3^B and x_4^B in the Fig.2(e) is reasonable. ■

Theorem 3 illustrates that two primary tasks are assigned to the same node in sequence, if their corresponding backup tasks are also assigned to the same node sequentially, their backup tasks can not overlap with each other on the allocated node.

Theorem 4: For any given $x_i, x_j \in X$ with $i \neq j$, if $n_i^P = n(x_j^P) = n_k$, $s(x_i^B) = 1$, there are two cases.

Case 1: if $s_{ik}^B < s_{jk}^B$, then the two copies of two different tasks subject to the constraint that $f_{ik}^B < f_{jk}^B$.

Case 2: if $s_{ik}^B > s_{jk}^B$, then the two copies of two different tasks subject to the constraint that $f_{ik}^B > f_{jk}^B$.

Proof: $n(x_i^B) = n(x_j^B) = n_k$, $s(x_i^B) = 1$. Assume that $s(x_j^B) = 0$, $s(x_{jk}^B) < f(x_{ik}^B)$, if there is a fault during the execution of x_j^P, then the state of x_i^B should be turn into active, the conflict between x_j^B and x_i^B will occur. As the example 5 in Fig.2(f), x_1^B can not overlap with x_4^B on the node n_1, x_2^B can't overlap with x_4^B on the node n_1 as well. This is because if node n_2 has a fault when x_1^p is on its execution, the completion of x_1 will turn to x_1^B. x_1^B overlaps with x_4^B on the node n_1, that is, $s_{41}^B < f_{11}^B$. Then, if during the period of $f_{11}^B - s_{41}^B$, node n_3 encounters a failure, the completion of x_4 will depend on x_4^B, but for now, x_1^B and x_4^B will have a conflict in occupying the node. For the case of x_2^B and x_4^B is the same. ■

Theorem 4 states that backup copies in active states cannot overlap with any backup copies on a node.

VI. FAULT-TOLERANCE BASED QOS-AWARE SCHEDULING ALGORITHM IN THE EDGE-CLOUD

The purpose of this paper is to maximize the QoS level of all the accepted tasks in the edge-cloud under the conditions of fault-tolerance and the time constraints. The fault-tolerance based QoS-aware scheduling algorithm (FTBQA) inspired by [15], [49], [52] including primary copy placement, backup copy placement, and the adjustment mechanism is proposed to improve the QoS levels of tasks. Moreover, to reduce the reliability cost of tasks and improve the reliability of the whole edge-cloud system, it indicates that nodes offering smaller reliability costs should be chosen in task allocations. Generally, our scheduling method also obeys the following allocation principles besides the scheduling principles in Section V. (1) Given the same service time, the algorithm should assign tasks to the nodes with lower failure rate. (2) For a group of nodes with the same failure rate, the algorithm will assign tasks in sequence to the nodes providing shorter service time. That is, in this case, tasks should be allocated to the computing nodes with powerful processing capacity and lower failure rate to improve the system's reliability. (3) For primary copies, they should be executed as early as possible. (4) For backup copies, they should be executed as late as possible with the deadline constraint being satisfied.

Algorithm 1 Primary Copy Placement for the Task in the Edge-Cloud

Input: The set of tasks: X, the range of QoS level: $\{q_1, \ldots, q_m\}$, the set of nodes: $\{n_1, n_2.., n_s\}$
Output: $\{n(x_i^P)\}$

```
1  for each new task x_i ∈ X do
2      est_i^P = INF;
3      flag = 0;
4      Temp = null ;
5      q_t = q_m;
6      while q_t ≠ q_1 do
7          {(n_ia, est_ia^P)} = Filter(x_i^P, {n_s});
8          r = 0;
9          for each node n_j ∈ {n_ia} do
10             if est_ij^P ≤ d_i − δ_ij^P then
11                 calculate system reliability r_ij
12                 if est_ij^P ≤ est_i^P and r_ij ≥ r then
13                     Temp = n_j;
14                     est_ij^P = est_i^P;
15                     r = r_ij;
16                     flag = 1;
17                 end
18             end
19         end
20         if flag == 0 then
21             q_t = q_{m−1};
22             else
23                 break;
24             end
25         end
26     end
27     if flag == 0 then
28         Give up to assign the primary copy x_i^P to the
           edge-cloud;
29         Consider to assign the primary copy x_i^P to
           the remote cloud center;
30         else
31             n(x_i^P) = Temp;
32         end
33     end
34 end
```

A. PRIMARY COPY PLACEMENT FOR THE TASK IN THE EDGE-CLOUD

The pseudo-code of primary copy placement for tasks in the edge-cloud is shown as Algorithm 1. The purpose of primary copy scheduling is to allocate primary copies to the nodes which make the system reliability maximal and to make the QoS levels of primary copies maximal with the time constraints being satisfied. As shown in Algorithm 1, from Line 1 to Line 5 is the parameter initialization.

At the beginning, the primary copy x_i^P is set as the maximal QoS level (Line 5). Based on the scheduling principles given by Section 5, we can get a set of nodes $\{(n_{ia}, est_{ia}^P)\}$ which has the appropriate idle time slot to process the corresponding task copy (Line 7). For the nodes in the candidate set, the node has the earliest start time for x_i^P and incurs the maximum system reliability is finally selected (Line 9-Line 18). If a node is successfully selected in the candidate set under the QoS level of q_m, the label flag is set to be 1, otherwise 0. For the latter case, the QoS level q_m will be decreased by 1 iteratively to find another candidate set under this level of QoS value until it turns to be the smallest value q_1. If no nodes in the edge-cloud can be satisfied with the requirement of x_i^P, the scheduler will further consider whether this primary copy can be scheduled to the remote cloud data center (Line 26-Line 32).

The time complexity of this algorithm is related with the number of tasks $|X|$, the number of QoS levels is m, and the number of nodes s. m is much smaller than $|X|$ and s. So, the time complexity of Algorithm 1 is $O(s|X|)$.

B. BACKUP COPY PLACEMENT FOR THE TASK IN THE EDGE-CLOUD

The pseudo-code of backup copy placement for tasks in the edge-cloud is shown as Algorithm 2. The backup copy allocation is also to assign the backup copies of tasks to the nodes which make the system reliability maximal and to make the QoS levels of backup copies maximal within the time constraints. The main difference between primary copy scheduling and backup copy scheduling is that the latter has the step of determining the status of the backup copy of a task. The value of the QoS level is also set from the biggest one to the smallest in sequence to find the candidate set of nodes (Line 3-Line 30). According to the scheduling principles given by Section 5, we can get a set of nodes (n_{ib}, lst_{ib}^B) which has the appropriate idle time slot to process the corresponding task copy (Line 5).

For each node in the candidate set, the status of the backup copy is firstly set as active (Line 10), If the finish time f_i^P of corresponding primary copy of the backup copy is lower than or equal to the latest start time of the backup copy (Line 19), then the status of the backup copy will be changed as passive. The node which has the latest start time for x_i^B and can lead to the maximum system reliability is finally selected for x_i^B (Lines 13-17). If no nodes in the edge-cloud can be satisfied with the requirement of x_i^B, the scheduler will consider whether this primary copy can be scheduled to the remote cloud data center (Lines 31-37).

The number of tasks whose primary copies have been allocated is $|T_p|$, which is lower than or equal to $|X|$. The number of QoS levels is m, the number of nodes is s. Thus, the time complexity of Algorithm 2 is $O(s|T_p|)$.

Algorithm 2 Backup Copy Placement for the Task in the Edge-Cloud

Input: The set of tasks whose primary copy have been allocated: T_p, $\{n(x_i^P)\}$, the range of Qos level: $\{q_1, \ldots, q_m\}$, the set of nodes: $\{n_1, n_2, \ldots, n_s\}$
Output: $\{n(x_i^B)\}$

```
1  for each backup task x_i^B ∈ T_p do
2      flag = 0;
3      q_t = q_m;
4      while q_t ≠ q_1 do
5          {(n_ib, lst_ib^B)} = Filter(x_i^B, {n_s});
6          r = 0;
7          lst^B = 0;
8          temp = null;
9          for each node n_j ∈ {n_ib} do
10             S(t_ij^B) = 1;
11             if lst_ij^B ≤ d_i − δ_ij(q_m) then
12                 calculate system reliability r_ij
                   according to Equation (13);
13                 if lst^B < lst_ij^B and r_ij ≥ r then
14                     temp = n_j;
15                     lst^B = lst_ij^B;
16                     r = r_jk;
17                     flag = 1;
18                     if f_i^P ≤ lst_ij^B then
19                         S(t_ij^B) = 0;
20                     end
21                 end
22             end
23         end
24         if flag==0 then
25             q_t = q_{m−1};
26             else
27                 break;
28             end
29         end
30     end
31     if flag==0 then
32         Give up to assign the backup copy x_i^B to the
           edge-cloud;
33         Consider to assign the backup copy x_i^B to the
           remote cloud center;
34         else
35             n(x_i^B) = temp;
36         end
37     end
38 end
```

C. REARRANGEMENT MECHANISM FOR THE PERFORMANCE IMPROVEMENT IN THE EDGE-CLOUD

The pseudo-code of adjustment mechanism is shown in Algorithm 3. When the primary copy of a task on a computing node is completed, the corresponding backup copy of the task will be removed from the initially allocated node. If the idle time slot left by the backup copy can be fully utilized by the primary copies located on the same node, the resource

Algorithm 3 Adjustment Mechanism for the Performance Improvement in the Edge-Cloud

Input: The finished primary copy x_t^P of node n_p, the node n_j which has the backup copy x_t^B corresponding to x_t^P, the unexecuted copies sequence CS_j of n_j;

Output: The adjusted execution orders of copies on the node n_j;

1 If x_t^P is completed, then the backup copy x_t^B corresponding to it will be removed from node n_j and the adjustment of the leftover task copies on the node n_j is invoked;
2 **for** *each* $x_i^P \in CS_j$ **do**
3 $I\_st = s_i^P$;
4 $I\_ft = f_i^P$;
5 Remove x_i^P from n_j;
6 Get the new finish time N_ft of x_i^P on the n_j;
7 **if** $N\_ft < I\_ft$ **then**
8 Reassign x_i^P to node n_j with the new available time slice;
9 $s_i^P = N\_ft - \delta_{ij}(x_i^P)$;
10 $f_i^P = N\_ft$;
11 Inform the centralized scheduler to update the status of x_i^B according to Property 1;
12 **else**
13 Reassign x_i^P to node n_j with the initial available time slice;
14 **end**
15 **end**
16 **end**
17 **for** *each* $x_i^B \in CS_j$ **do**
18 $I\_st = s_i^B$;
19 $I\_ft = f_i^B$;
20 Remove x_i^B from n_k;
21 Get the new start time N_st of x_i^B on the n_k;
22 **if** $N\_st > I\_st$ **then**
23 Reallocate x_i^B to n_k;
24 $s_i^B = N\_st$;
25 $f_i^B = N\_st + \delta_{ij}(x_i^B)$;
26 Update the status of x_i^B according to the Property 1;
27 **else**
28 Reassign x_i^B to node n_k with the initial available time slice;
29 **end**
30 **end**
31 **end**

utilization can be greatly promoted and the time performance of the following primary copies can also be further improved. Inspired by [49], [52], we also adjust the execution sequence of copies on a node if the backup copy corresponding to a task is removed from the node. The adjustment mechanism can advance the start time of primary copies, reduce the redundant parts of active backup copies. When a primary copy completes its execution and the corresponding backup copy is deleted, the adjustment process will be invoked on the node where the backup copy is removed. Firstly, all primary copies waiting on the node are checked if they can be brought forward (Lines 2-16). Then all backup copies on the node are checked if they can be brought backward (Lines 17-31). It should be noted that such adjustment still follows the scheduling principles in Section 5. The number of unexecuted tasks on node n_j is $|CSj|$, which is lower than or equal to $|X|$. Thus, the time complexity of Algorithm 3 is $O(|CS_j|)$.

As Example 1 shown in Fig. 2(a), when x_1^P is executed on node n_2, x_1^B is assigned to node n_1 with the passive scheme. When the execution of x_1^P is finished, it is necessary to judge whether the time length of $s_{21}^B - f_{12}^P$ is enough to support the operation of x_3^P. If possible, as the result of adjustment shown in Fig. 2(f), x_3^P can be moved forward, so that task x_3 can be executed earlier, and the state of x_3^B can be set to passive at the same time to save the resource of the corresponding node.

VII. PERFORMANCE EVALUATION

The experimental results obtained from extensive simulations to evaluate the performance of the proposed method FTBQA in this paper are presented in this section. All the simulations are conducted using Python 3.6 on a machine with 3.60GHz Intel(R) Core(TM) i7-7700 CPU and 8GB RAM. In the simulation experiments, the following metrics are concerned [15], [16].

(1) Guarantee Ratio (GR). It is equal to the number of tasks that can be completed within their deadlines divides the total number of accepted tasks $\times 100\%$.

(2) Average QoS Level is used to record the average QoS levels of all accepted tasks.

(3) Reliability Cost (RC) is a metric measuring the reliability of the edge-cloud in a time unit.

To evaluate the performance of FTBQA, the following questions are involved.

RQ1: Does the PB strategy based fault-tolerance mechanism can improve the guarantee ratio of tasks in the edge-cloud environment? For this, we have the compared method SIMPLE which is a variant of FTBQA. It has no fault-tolerance mechanism and the adjustment mechanism. For ease of comparison, SIMPLE uses the same scheduling strategy as Algorithm 1.

RQ2: Does the adjustment mechanism included in the scheduling model can have a great impact on the performance of task processing? To answer this question, we have the comparing between FTBQA and NOADJUST. NOADJUST is also a variant of FTBQA. The main difference between these two methods is that the former has the adjustment mechanism which can be adopted to rearrange the task copies of a node.

RQ3: How about the performance difference of FTBQA between the state-of-the-art methods which have a similar goal with our work? For this, we have two methods,

namely, QAFT [15] and FESTAL [16] as the benchmarks. QAFT can adaptively adjust the QoS levels of tasks and the execution schemes of backup copies to attain high system flexibility, which is similar to our method. The difference is that our method FTBQA firstly consider assigning copies of tasks to the edge-cloud, if a task is refused by the edge-cloud, it can also be allocated to the remote cloud data center within the time constraint. The FTBQA has the adjustment mechanism which can be invoked when the backup copy of a task on a computing node has been removed due to the completion of corresponding primary copy, in order to improve the resource utilization of corresponding node and service time of the leftover primary task copies. FESTAL also considers backup overlapping. Different from FTBQA, FESTAL each time selects the nodes with the minimum processing capacities which can satisfy the requirements of tasks. It neither has the adjustment mechanism nor the consideration of the QoS requirement of tasks.

A. SIMULATION SETUP

The experimental parameters in the simulations are similar to those used in the literature [15], [48], [51], [52].

To study the performance under the more realistic scenario, our simulations are based on the real-world trace of San Francisco taxis which contains the GPS coordinates of approximately 500 taxis collected over 24 days in the San Francisco Bay Area [53]. This trace of taxis naturally shows the characteristics of requests like the number and requirement of requests from end devices during a time slot at different regions which are covered by edge-clouds. Similar to [51], we adopt a part of the data which is equivalent to a period of 5 consecutive days in the simulations. The distance between base stations (center of the cell) is set to be 1000 m, and the hexagon structure is adopted to depict the range of the geographical location. User locations then can be mapped to the cell location in the way of considering which cell a user is included in. In this dataset, there are totally 536 individual users, and only part of them are at the active state within a specific period of time. The number of active users at any time is a random value in [0, 409], the average number of active users is 278 [2]. Assume that only active users can generate task requests, so the average task arrival rate λ_r in the whole period of time is 0.95 for each application. Specifically, the arrival rate λ_r is given by $\lambda_r = \lambda_{r-1} + iT$. iT is the interval time which is a random positive real number in $\{1, 2, 3, 4, 5, 6, 7\}$, and $\lambda_0 = 0$ [49], [52]. The instantaneous task arrival rate is related to the number of active users, the larger (smaller) the number of active users, the higher (lower) the task arrival rate will be.

The data traffic generated from each source point is proportional to the number of active users [48], [52]. Data from all the source points are transmitted to the edge-cloud in the form of packets. The packet size is a random value in [34, 6550]B [48]. The instruction size is taken as 64 bits. The packet arrival rate follows a Poisson distribution with the mean packet arrival rate being 1 packet per node per second. The bandwidth capacity between source points and the edge-cloud is 1Gbps, the bandwidth capacity between the edge-cloud and the remote cloud data center is as 10Gbps [48].

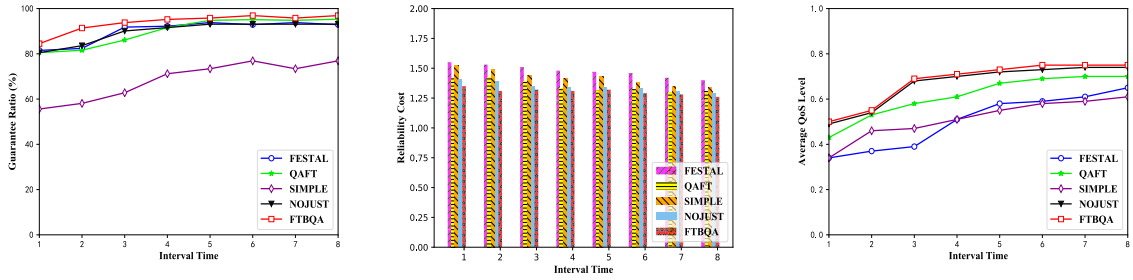
To describe the node heterogeneity, the parameter p_j, a positive real number, is used to denote the node power of node n_j [15]. The parameter Ap represents the average processing power of all nodes, Ps is the power span which takes the average power Ap as the center. p_j is uniformly distributed between $Ap - Ps$ and $Ap + Ps$. Ap is set to be 700. Ps is in $\{160, 200, 240, 280, 320, 360, 400\}$. The fault-tolerance based scheduling model proposed in this paper is a general one, without losing generality, we can give a general definition of QoS level. The QoS level of x_i^P and x_i^B, namely, $q(x_i^P)$ and $q(x_i^B)$, are subject to $0 \leq q(x_i^P) \leq 1$, $0 \leq q(x_i^B) \leq 1$ [21], [46]. Specificly, they are in $[0, 0.1, 0.2, \ldots, 0.9, 1]$. The deadline d_i of task x_i is set as $d_i = a_i + max\{\delta_{ij}\} + bD$ [7]. bD is the base deadline which is a random positive real number in $\{170, 200, 230, 260, 290, 320, 350, 380\}$. It determines whether the tasks have loose deadlines or not. The failure rate of node n_j is uniformly distributed with the average value λ_u in (1.2, 2.0) and the time unit is $10^{-7}/h$ [12], [15].

B. IMPACT OF TASK ARRIVAL RATE

This section assesses the impact of task arrival rates on service performance. iT is the task arrival interval, the smaller the value of iT is, the greater the task arrival rate will be, the larger the workload per unit of time will be. The value of iT is set to be 1,2,3,4,5,6,7 respectively, other parameters keep the same.

Fig. 3(a) shows the change in the guarantee ratio (GR) of tasks with the five methods as the value of the iT increases, that is, the arrival rate of the task decreases. With the decrease of the task arrival rate, the GR values corresponding to the five methods show different degrees of the upward trend. Moreover, the corresponding GRs of the other four methods are higher than those of SIMPLE method. This is because SIMPLE simply uses a scheduling strategy similar to the method FTBQA. There is no other special optimization measure in it, so the overall GR value is lowest. As the task arrival rate decreases, i.e. iT is from 5 to 8, the GR values of FESTAL, QAFT, NOJUST, FTBQA all increase slowly. This is because the task arrival rate is reduced, the load is decreasing, the resources are more sufficient, and the performance optimization of these methods may reach their upper limits. The GR values corresponding to FTBQA are generally in a more advantageous state, having a 2% to 9% advantage over those of QAFT, FESTAL, NOJUST.

Fig. 3(b) shows the change in reliability cost (RC) of tasks corresponding to the five methods as the task arrival rate decreases. It can be seen from the figure that as the arrival rate of the task decreases, that is, the load decreases, the reliability costs corresponding to FESTAL and SIMPLE show a downward trend, while the RC values of QAFT, NOJUST and FTBQA are relatively stable. It is because they have taken the impact of reliability cost into account when they

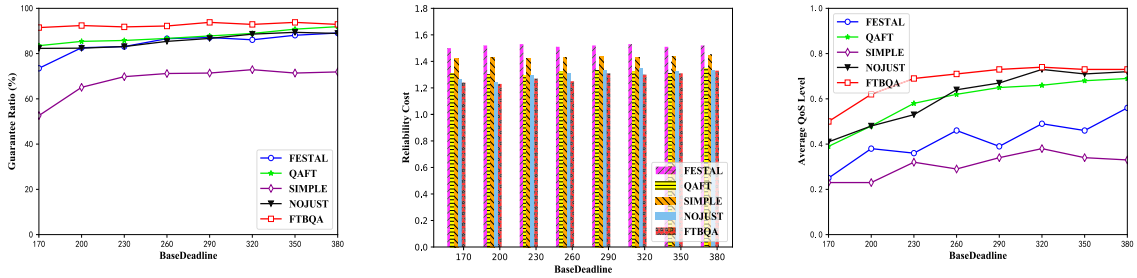


(a) Effect of task arrival rate on the guarantee ratio

(b) Effect of task arrival rate on the reliability cost

(c) Effect of task arrival rate on the average QoS level

FIGURE 3. Impact of Task Arrival Rate.



(a) Effect of task deadline on the guarantee ratio

(b) Effect of task deadline on the reliability cost

(c) Effect of task deadline on the average QoS level

FIGURE 4. Impact of Task Deadline.

are doing the scheduling. Although SIMPLE uses a strategy similar to FTBQA, it has no other optimization mechanism, so its ability of improving the task time is limited, that is, the reliability cost corresponding to it will be high.

Fig. 3(c) shows the results of the average QoS levels of tasks under the five methods with the decrease of the task arrival rate. With the decrease of the task arrival rate, the average QoS level that the task can obtain is guaranteed to rise. This is because the load is reduced and the competition for the resource is small. At the same time, because FTBQA, QAFT, and NOJUST have the QoS adaptation mechanism when scheduling tasks, their corresponding average QoS will be higher. The average QoS levels corresponding to the FTBQA is slightly better those of the NOJUST because the former has an adjustment mechanism, which can further optimize the completion time of tasks. Interestingly, when iT is set to be from 1 to 4, the average QoS levels corresponding to FESTAL are lower than those of the SIMPLE, but later when iT is greater than 4, FESTAL shows better results than SIMPLE in the QoS levels.

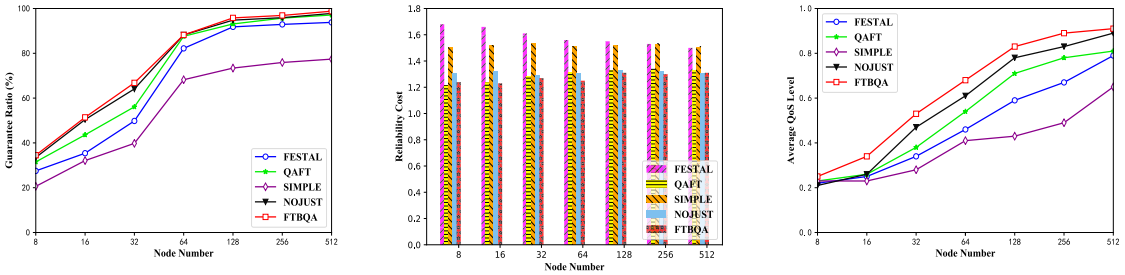
C. IMPACT OF TASK DEADLINE

To show the impact of task deadline, the base deadline is set to be 170,200,230,260,290,320,350,380 respectively. The higher the base deadline, the more loose deadline constraint of a task, other parameters are not changed.

Fig. 4(a) shows the change in GR values of the five methods as the deadline constraint is relaxed. The changes in the GR values corresponding to FTBQA, QAFT, NOJUST, and FESTAL are generally stable, this is because they all have the fault tolerance and overlapping strategies. SIMPLE simply uses a scheduling strategy similar to FTBQA. In task assignment, FTBQA considers the two factors of QoS and reliability cost with the fault-tolerant mechanism. Therefore, it is normal for FTBQA to show the advantage in the GR values compared with the other methods.

Fig. 4(b) depicts the results of the reliability costs corresponding to the five methods when the deadline constraints of tasks become more and more relaxed. It can be seen from the figure that the RC values of FTBQA, QAFT, and NOJUST are not very large as a whole because they have considered the reliability cost when performing task scheduling. The changing in reliability costs of FESTAL is not very large, but its corresponding RC value is larger than the previous three. Because FESTAL uses the fault tolerance and overlay mechanism but does not consider the reliability cost, so the RC value is relatively high. SIMPLE adopts a scheduling strategy similar to FTBQA, the reliability cost factor is also considered when performing task scheduling, so its corresponding RC value is smaller than FESTAL.

Fig. 4(c) shows the results of the five methods in the average QoS levels when the tasks' deadline constraints are getting looser. The more relaxed the task deadline, the QoS level values of these five methods are expected to rise. This is because, when the deadline of the task is loose, the schedulability of the resource node is enhanced, and the tasks can obtain a higher QoS level guarantee. As seen from the figure, when the bT value is between 170 and 290, the average QoS levels corresponding to QAFT and NOJUST exhibit a cross-change. When bT exceeds 290, NOJUST exhibits a state closer to FTBQA. FESTAL and SIMPLE have large fluctuations, but they are generally weaker than the former



(a) Effect of node number on the guarantee ratio

(b) Effect of node number on the reliability cost

(c) Effect of node number on the average QoS level

FIGURE 5. Impact of Node Number.

three methods. This is because they do not pay attention to the optimization of QoS levels, but with the loose change of deadlines, their corresponding QoS levels are overall in a growing trend.

D. IMPACT OF NODE NUMBER

This section shows the performance impact of node numbers. The node number in edge-cloud is set to be 8,16,32, 64,128,256,512 respectively and other parameters are the same as the former experiments.

Fig. 5(a) represents the change in the GR values of the five methods as the number of compute nodes increases in the edge-cloud. As the number of computing nodes increases, the GR values corresponding to the five methods are increasing. This is because the number of compute nodes is increased and the resources are relatively more abundant, so more task requests can be served. The GR values of FTBQA, QAFT, and NOJUST are relatively higher because they can adaptively adjust the QoS level so that more tasks can be accepted. Because FTBQA and NOJUST not only use the computing resources of the edge cloud but also assign tasks to the remote cloud data center, overall, their guarantee rates will be higher. Meanwhile, the guaranteed rate difference between FTBQA and QAFT and FESTAL is between 3%-15% and 1%-8%, respectively. Compared to FTBQA, the task guarantee rate of NOJUST is slightly lower, which is reasonable because there is no adjustment mechanism in NOJUST. Compared to SIMPLE, FESTAL has the fault-tolerant strategy in it, so its guarantee rate could be higher.

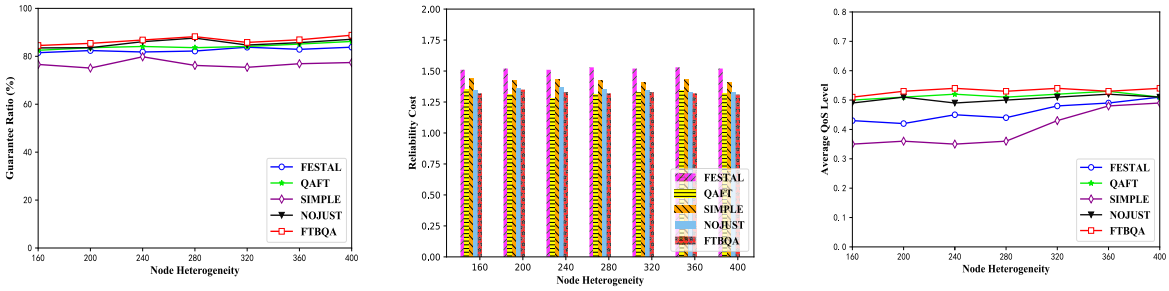
Fig. 5(b) depicts the change in reliability costs of the five methods as the number of compute nodes increases in the edge-cloud. As the number of nodes increases, since the failure rate of nodes is uniformly distributed, the number of nodes with high reliability also increases. Because the number of tasks is relatively fixed at this period, more tasks can be assigned to the nodes with higher reliability. Therefore, for example, it is reasonable that the reliability costs of FESTAL will decrease. The reliability values of the other four methods are relatively small because they all take into account the impact of reliability costs when performing task scheduling. As can be seen from the figure, the reliability values of FTBQA, NOJUST, and QAFT are not much different, while FTBQA has a performance improvement of 2.5%-5% compared to FESTAL, and about 2.5% performance improvement compared to SIMPLE.

Fig. 5(c) shows the change in the average QoS levels of tasks with respect to the five methods as the number of computing nodes increases. With the increase of the number of computing nodes in the edge-cloud, the average QoS levels of tasks corresponding to the five methods are in the tendency of increase. This is because, as the number of computing nodes increases, that is, the computing power increases, there is a relatively more sufficient resource to handle relatively fixed tasks, so the corresponding average QoS levels can be improved. When the number of nodes is from 8 to 16, the overall improvement of the QoS levels of these five methods are relatively small, and when the number of nodes is from 16 to 128, the corresponding increment is relatively large. Moreover, when the number of nodes is from 128 to 512, the QoS levels of FTBQA, NOJUST, and QAFT increase relatively flatly, because they all consider QoS requirements when performing task scheduling, so when the resource nodes grow to a certain value, they maintain a relatively stable average QoS levels. FESTAL and SIMPLE are growing at a smaller rate.

E. IMPACT OF NODE HETEROGENEITY

To investigate the impact of node heterogeneity, the parameter p_j, a positive real number, is used to denote the node power of node n_j [32]. The parameter Ap represents the average processing power of all nodes, Ps is the power span which takes the average power Ap as the center. p_j is uniformly distributed between $Ap - Ps$ and $Ap + Ps$. Ap is set to be 700. Ps is set to be 160,200,240,280,320,360,400 respectively.

Fig. 6(a) represents the changing in GR values of the five methods when the task amount is fixed and the performance difference of nodes in the edge-cloud is varying. When the power span value increases from 160 to 240, the larger the value, the greater the difference in processing power of the computing nodes in the edge-cloud. In Fig. 6(a), the GR values of FTBQA, NOJUST, QAFT, and FESTAL are higher than those of SIMPLE as a whole because they all adopt a fault-tolerant strategy and show a relatively stable change when the difference in processing power of nodes varies. Moreover, FTBQA not only adopts the adjustment mechanism but also considers assigning tasks to the remote cloud

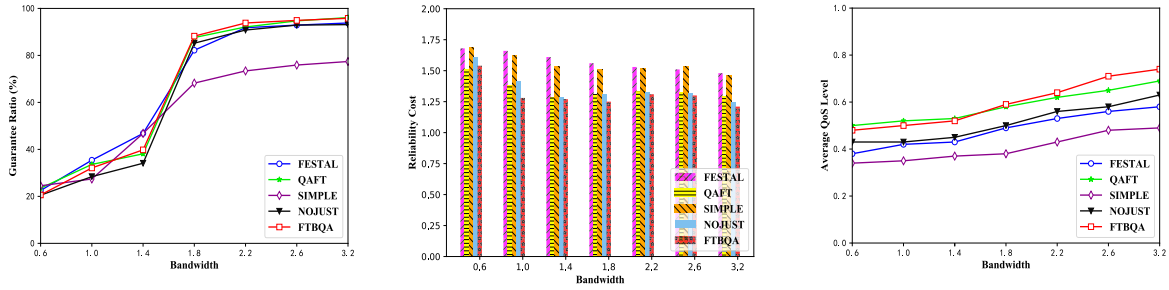


(a) Effect of node heterogeneity on the guarantee ratio

(b) Effect of node heterogeneity on the reliability cost

(c) Effect of node heterogeneity on the average QoS level

FIGURE 6. Impact of Node Heterogeneity.



(a) Effect of bandwidth on the guarantee ratio

(b) Effect of bandwidth on the reliability cost

(c) Effect of bandwidth on the average QoS level

FIGURE 7. Impact of Bandwidth.

for execution, which can improve the guarantee ratio of tasks, so its GR values will be higher than other methods. As seen from the figure, when the power span value is 160-240, the GR values corresponding to SIMPLE fluctuate greatly. When the power span value is greater than 240, the GR values corresponding to SIMPLE change slightly. Because with the relatively fixed task load, although the difference in the processing power of the computing node is increased, SIMPLE can allocate more tasks to the nodes with high processing power and high reliability, which can also guarantee a certain task guarantee rate.

Fig. 6(b) shows the impact of the variation of the node computing power on the reliability cost of tasks. It can be seen from the figure that the RC values corresponding to the five methods are less affected by the change of the node power. Because FTBQA, QAFT, NOJUST, and FESTAL all adopt fault-tolerant strategies, they can make full use of the currently available resources. FTBQA, NOJUST, and SIMPLE all consider reliability cost factors when performing task scheduling. Overall, the RC values of FTBQA, QAFT and NOJUST are similar and they all show better performance than FESTAL and SIMPLE.

Fig. 6(c) represents the average QoS levels of tasks corresponding to the five methods affected by the varying node heterogeneity in the edge-cloud. The average task QoS levels corresponding to FTBQA, QAFT, and NOJUST are hardly affected by the change in the processing power of the nodes. This is because they consider the QoS level factor when performing task scheduling, so their corresponding QoS level values of tasks, in general, are higher than the other two methods. The average QoS level values corresponding to the FESTAL and SIMPLE methods are relatively stable when the power span is 160-280. When the power span is greater than 280, their corresponding average QoS level values show an upward trend. This is because, when the difference in computing power of the edge nodes is larger, in the case of the same load, more tasks may be allocated to the computing nodes with higher processing power, so the overall QoS level value will be relatively in a slightly increasing trend.

F. IMPACT OF BANDWIDTH

To show the impact of bandwidth, the bandwidth capacity between source points and the edge-cloud is set to be 0.6Gbps, 1Gbps, 1.4Gbps,...,3.2Gbps respectively. The bandwidth capacity between the edge-cloud and remote cloud data center is set as 10Gbps [48]. Fig 7 shows the GR values, RC values and average QoS levels related to the five methods with the change of bandwidth.

Fig. 7(a) represents the trends in GR values of these five methods with the change of bandwidth between source points and the edge-cloud. From the figure, we can see that the guarantee ratios of these five methods are low when the bandwidth capacity between source points and the edge-cloud is lower than 1.8Gbps. FTBQA, NOJUST, QAFT, and FESTAL all have a fault-tolerant strategy, so their performance is limited by the relatively lower bandwidth. As for the method SIMPLE, it has no fault-tolerance mechanism and the adjustment mechanism, the limitation of bandwidth has no significant negative influence on the guarantee ratio of tasks compared with other methods. When the bandwidth resource is relatively enough, all these methods have higher GR values than before, because more tasks can be transmitted to

edge-cloud or the remote cloud to be processed. FTBQA, NOJUST, QAFT, and FESTAL present more advantages than SIMPLE. Especially, FTBQA and QAFT explicitly outperform the other three methods. While we can see the RC values of these methods in Fig.7(b), With the increase of bandwidth, the RC values of FESTAL and SIMPLE are still higher than those of FTBQA, NOJUST, QAFT. Because they don't take the reliability cost into consideration when assigning tasks. Also the same as the case of average QoS level, as shown in Fig.7(c), the average task QoS levels corresponding to FTBQA, QAFT are greater than those of the other three methods. Although the bandwidth between source points and the edge-cloud can affect the performance of FTBQA, we can generally draw the conclusion that FTBQA indeed outperforms other methods according to all the experiments we conducted.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we propose a novel QoS-aware scheduling model based on fault-tolerance in the edge-cloud by extending the traditional primary-backup fault-tolerant model, which enables the improvement of the reliability of application services in the edge-cloud under the time constraints of tasks. This fault model can be easily extended to the case with multiple failing nodes at a time in the edge-cloud environment which has a large number of computing nodes. Because the large set of nodes can be divided into several small groups, then the fault model outlined in Section I can also be applied to each group. With the fault-tolerant scheduling model, a fault-tolerance based QoS-aware scheduling algorithm (FTBQA) having three parts is proposed to improve the performance of services. It schedules independent real-time tasks tolerating hardware failures in the edge-cloud with heterogeneous resources. We present the scheduling principles and algorithms for primary and backup copies in Section V and Section VI. With the adjustment mechanism in FTBQA, when the backup copy of a task is removed from its assigned computing node, the start time of the leftover primary copies on the node could be advanced, the time performance of tasks can be better satisfied. Finally, we conduct extensive simulation experiments to verify the performance difference between FTBQA and the four benchmarks, namely QAFT, NOJUST, FESTAL and SIMPLE in terms of guarantee ratio, average QoS level, and the reliability cost. Although the bandwidth between source points and the edge-cloud, compared with other factors such as task arrival rate, task deadline, fog node number and node heterogeneity, has greater impact on the performance of FTBQA and QAFT, we can make the conclusion that FTBQA generally outperforms other methods according to all the experiments we have done.

In the future, we will refine our fault-tolerance based scheduling model to multidimensional computing resources i.e memory, network bandwidth, etc. in the edge-cloud environment. We also plan to implement FTBQA in the real edge-cloud scenario.

REFERENCES

[1] M. Mukherjee, L. Shu, and D. Wang, "Survey of fog computing: Fundamental, network applications, and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 1826–1857, 3rd Quart., 2018.
[2] Y. Harchol, A. Mushtaq, J. McCauley, A. Panda, and S. Shenker, "Cessna: Resilient edge-computing," in *Proc. Workshop Mobile Edge Commun.*, 2018, pp. 1–6.
[3] A. Aral and I. Brandic, "Dependency mining for service resilience at the edge," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2018, pp. 228–242.
[4] N. Ivaki, S. Boychenko, and F. Araujo, "A fault-tolerant session layer with reliable one-way messaging and server migration facility," in *Proc. IEEE 3rd Symp. Netw. Cloud Comput. Appl. (NCCA)*, Feb. 2014, pp. 75–82.
[5] M. A. Haque, H. Aydin, and D. Zhu, "On reliability management of energy-aware real-time systems through task replication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 813–825, Mar. 2017.
[6] X. Liu, Y. Ma, Y. Liu, T. Xie, and G. Huang, "Demystifying the imperfect client-side cache performance of mobile Web browsing," *IEEE Trans. Mobile Comput.*, vol. 15, no. 9, pp. 2206–2220, Sep. 2016.
[7] D. Ossmann and H.-D. Joos, "Multiobjective optimization-based fault-tolerant flight control system design," *Int. J. Robust Nonlinear Control*, vol. 29, no. 16, pp. 5341–5355, Sep. 2017.
[8] Q. Wei, J. Jiao, and T. Zhao, "Flight control system failure modeling and verification based on SPIN," *Eng. Failure Anal.*, vol. 82, pp. 501–513, Dec. 2017.
[9] D. He, H. Cao, S. Wang, and X. Chen, "Time-reassigned synchrosqueezing transform: The algorithm and its applications in mechanical signal processing," *Mech. Syst. Signal Process.*, vol. 117, pp. 255–279, Feb. 2019.
[10] Q. Zhai and Y. Wang, "Stochastic resonance in parallel concatenated turbo code decoding," *Digit. Signal Process.*, vol. 56, pp. 93–99, Sep. 2016.
[11] Z. Yan, G. He, W. He, S. Wang, and Z. Mao, "High performance parallel turbo decoder with configurable interleaving network for LTE application," *Integration*, vol. 52, pp. 77–90, Jan. 2016.
[12] J. Liu, M. Wei, W. Hu, X. Xu, and A. Ouyang, "Task scheduling with fault-tolerance in real-time heterogeneous systems," *J. Syst. Archit.*, vol. 90, pp. 23–33, Oct. 2018.
[13] K. Cao, G. Xu, J. Zhou, M. Chen, T. Wei, and K. Li, "Lifetime-aware real-time task scheduling on fault-tolerant mixed-criticality embedded systems," *Future Gener. Comput. Syst.*, vol. 100, pp. 165–175, Nov. 2019.
[14] W. Luo, J. Li, F. Yang, G. Tu, L. Pang, and L. Shu, "DYFARS: Boosting reliability in fault-tolerant heterogeneous distributed systems through dynamic scheduling," in *Proc. 8th ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw., Parallel/ Distrib. Comput. (SNPD)*, Jul./Aug. 2007, pp. 640–645.
[15] X. Zhu, X. Qin, and M. Qiu, "QoS-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters," *IEEE Trans. Comput.*, vol. 60, no. 6, pp. 800–812, Jun. 2011.
[16] J. Wang, W. Bao, X. Zhu, L. T. Yang, and Y. Xiang, "FESTAL: Fault-tolerant elastic scheduling algorithm for real-time tasks in virtualized clouds," *IEEE Trans. Comput.*, vol. 64, no. 9, pp. 2545–2558, Sep. 2015.
[17] H. Gao, Y. Duan, L. Shao, and X. Sun, "Transformation-based processing of typed resources for multimedia sources in the IoT environment," *Wireless Netw.*, pp. 1–17, 2019.
[18] Y. Yin, L. Chen, Y. Xu, J. Wan, H. Zhang, and Z. Mai, "QoS prediction for service recommendation with deep feature learning in edge computing environment," *Mobile Netw. Appl.*, pp. 1–11, 2019.
[19] Y. Yin, W. Zhang, Y. Xu, H. Zhang, Z. Mai, and L. Yu, "QoS prediction for mobile edge service recommendation with auto-encoder," *IEEE Access*, vol. 7, pp. 62312–62324, 2019.
[20] H. Gao, W. Huang, Y. Duan, X. Yang, and Q. Zou, "Research on cost-driven services composition in an uncertain environment," *J. Internet Technol.*, vol. 20, no. 3, pp. 755–769, 2019.
[21] S. Creemers, "The preemptive stochastic resource-constrained project scheduling problem," *Eur. J. Oper. Res.*, vol. 277, no. 1, pp. 238–247, Aug. 2019.
[22] H. Yao, M. Xiong, H. Li, L. Gu, and D. Zeng, "Joint optimization of function mapping and preemptive scheduling for service chains in network function virtualization," *Future Gener. Comput. Syst.*, to be published.
[23] F. Jaramillo and M. Erkoc, "Minimizing total weighted tardiness and overtime costs for single machine preemptive scheduling," *J. Comput. Ind. Eng.*, vol. 107, pp. 109–119, May 2017.
[24] O. Kermia, "An efficient approach for the multiprocessor non-preemptive strictly periodic task scheduling problem," *J. Syst. Archit.*, vol. 79, pp. 31–44, Sep. 2017.

[25] J. Chen, C. Du, F. Xie, and B. Lin, "Scheduling non-preemptive tasks with strict periods in multi-core real-time systems," *J. Syst. Archit.*, vol. 90, pp. 72–84, Oct. 2018.
[26] P. M. Castro, I. Harjunkoski, and I. E. Grossmann, "Discrete and continuous-time formulations for dealing with break periods: Preemptive and non-preemptive scheduling," *Eur. J. Oper. Res.*, vol. 278, no. 2, pp. 563–577, Oct. 2019.
[27] T. Wei, P. Mishra, K. Wu, and J. Zhou, "Quasi-static fault-tolerant scheduling schemes for energy-efficient hard real-time systems," *J. Syst. Softw.*, vol. 85, no. 6, pp. 1386–1399, Jun. 2012.
[28] Y. Guo, D. Zhu, H. Aydin, J.-J. Han, and L. T. Yang, "Exploiting primary/backup mechanism for energy efficiency in dependable real-time systems," *J. Syst. Archit.*, vol. 78, pp. 68–80, Aug. 2017.
[29] Q. Zheng, B. Veeravalli, and C.-K. Tham, "On the design of fault-tolerant scheduling strategies using primary-backup approach for computational grids with low replication costs," *IEEE Trans. Comput.*, vol. 58, no. 3, pp. 380–393, Mar. 2009.
[30] S. Ghosh, R. Melhem, and D. Mosse, "Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, no. 3, pp. 272–284, Mar. 1997.
[31] G. Manimaran and C. S. R. Murthy, "A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 11, pp. 1137–1152, 1998.
[32] R. Al-Omari, A. K. Somani, and G. Manimaran, "Efficient overloading techniques for primary-backup scheduling in real-time systems," *J. Parallel Distrib. Comput.*, vol. 64, no. 5, pp. 629–648, May 2004.
[33] T. Tsuchiya, Y. Kakuda, and T. Kikuno, "A new fault-tolerant scheduling technique for real-time multiprocessor systems," in *Proc. 2nd Int. Workshop Real-Time Comput. Syst. Appl. (RTCSA)*, Oct. 1995, pp. 197–202.
[34] C.-H. Yang, G. Deconinck, and W.-H. Gui, "Fault-tolerant scheduling for real-time embedded control systems," *J. Comput. Sci. Technol.*, vol. 19, no. 2, pp. 191–202, Mar. 2004.
[35] R. Al-Omari, A. K. Somani, and G. Manimaran, "An adaptive scheme for fault-tolerant scheduling of soft real-time tasks in multiprocessor systems," *J. Parallel Distrib. Comput.*, vol. 65, no. 5, pp. 595–608, May 2005.
[36] C. Zhang, H. Zhao, and S. Deng, "A density-based offloading strategy for IoT devices in edge computing systems," *IEEE Access*, vol. 6, pp. 73520–73530, 2018.
[37] M. H. Ghahramani, M. Zhou, and C. T. Hon, "Toward cloud computing QoS architecture: Analysis of cloud systems and cloud services," *IEEE/CAA J. Autom. Sinica*, vol. 4, no. 1, pp. 6–18, Jan. 2017.
[38] L. Qi, R. Wang, C. Hu, S. Li, Q. He, and X. Xu, "Time-aware distributed service recommendation with privacy-preservation," *Inf. Sci.*, vol. 480, pp. 354–364, Apr. 2019.
[39] L. Li, G. Xu, L. Jiao, X. Li, H. Wang, J. Hu, H. Xian, W. Lian, and H. Gao, "A secure random key distribution scheme against node replication attacks in industrial wireless sensor systems," *IEEE Trans Ind. Informat.*, vol. 16, no. 3, pp. 2091–2101, Mar. 2020.
[40] X. Gao, E. Akyol, and T. Basar, "Communication scheduling and remote estimation with adversarial intervention," *IEEE/CAA J. Autom. Sinica*, vol. 6, no. 1, pp. 32–44, Jan. 2019.
[41] P. Zhang, M. Zhou, and G. Fortino, "Security and trust issues in fog computing: A survey," *Future Gener. Comput. Syst.*, vol. 88, pp. 16–27, Nov. 2018.
[42] H. Yuan, J. Bi, M. Zhou, and A. C. Ammari, "Time-aware multi-application task scheduling with guaranteed delay constraints in green data center," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 3, pp. 1138–1151, Jul. 2018.
[43] H. Yuan, J. Bi, and M. Zhou, "Temporal task scheduling of multiple delay-constrained applications in green hybrid cloud," *IEEE Trans. Services Comput.*, to be published.
[44] M. Kang, C. Wen, and C. Wu, "A model predictive scheduling algorithm in real-time control systems," *IEEE/CAA J. Autom. Sinica*, vol. 5, no. 2, pp. 471–478, Mar. 2018.
[45] Q. Fan and N. Ansari, "On cost aware cloudlet placement for mobile edge computing," *IEEE/CAA J. Autom. Sinica*, vol. 6, no. 4, pp. 926–937, Jul. 2019.
[46] H. Gao, Y. Xu, Y. Yin, W. Zhang, R. Li, and X. Wang, "Context-aware QoS prediction with neural collaborative filtering for Internet-of-Things services," *IEEE Internet Things J.*, to be published, doi: 10.1109/JIOT.2019.2956827.
[47] L. Niu and D. Zhu, "Reliability-aware scheduling for reducing system-wide energy consumption for weakly hard real-time systems," *J. Syst. Archit.*, vol. 78, pp. 30–54, Aug. 2017.
[48] S. Sarkar, S. Chatterjee, and S. Misra, "Assessment of the suitability of fog computing in the context of Internet of Things," *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 46–59, Jan. 2018.
[49] P. Guo, M. Liu, J. Wu, Z. Xue, and X. He, "Energy-efficient fault-tolerant scheduling algorithm for real-time tasks in cloud-based 5G networks," *IEEE Access*, vol. 6, pp. 53671–53683, 2018.
[50] M. Guo, L. Li, and Q. Guan, "Energy-efficient and delay-guaranteed workload allocation in IoT-edge-cloud computing systems," *IEEE Access*, vol. 7, pp. 78685–78697, 2019.
[51] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Perform. Eval.*, vol. 91, pp. 205–228, Sep. 2015.
[52] P. Guo and Z. Xue, "Real-time fault-tolerant scheduling algorithm with rearrangement in cloud systems," in *Proc. IEEE 2nd Inf. Technol. Netw., Electron. Autom. Control Conf.*, Dec. 2017, pp. 399–402.
[53] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, "A parsimonious model of mobile partitioned networks with clustering," in *Proc. 1st Int. Commun. Syst. Netw. Workshops*, Jan. 2009, pp. 1–10.

HUAIYING SUN received the B.S. degree from the Jiangxi University of Science and Technology, in 2015. She is currently pursuing the Ph.D. degree in computer science with the East China University of Science and Technology (ECUST). Her research interests include software engineering, cloud computing, service oriented computing, fog/edge computing, and formal methods.

HUIQUN YU received the B.S. degree from Nanjing University, in 1989, the M.S. degree from the East China University of Science and Technology (ECUST), in 1992, and Ph.D. degree from Shanghai Jiaotong University, in 1995, all in computer science. He is currently a Professor of computer science with the Department of Computer Science and Engineering, ECUST. His research interests include software engineering, high-confidence computing systems, cloud computing, and formal methods.

GUISHENG FAN received the B.S. degree from the Anhui University of Technology, in 2003, and the M.S. and Ph.D. degrees from the East China University of Science and Technology (ECUST), in 2006 and 2009, respectively, all in computer science. He is currently a Research Assistant with the Department of Computer Science and Engineering, ECUST. His research interests include formal methods for complex software systems, service oriented computing, and techniques for analysis of software architecture.

LIQIONG CHEN received the B.S. degree from the Anhui University of Technology, in 2004, and the Ph.D. degree from the East China University of Science and Technology (ECUST), in 2009, all in computer science. She is currently an Associate Professor with the Department of Computer Science and Information Engineering, Shanghai Institute of Technology. Her research interests include formal methods for complex software systems, service oriented computing, and techniques for analysis of software architecture.