electronics

Article

Deadline-Aware Dynamic Task Scheduling in Edge–Cloud Collaborative Computing

Yu Zhang [1,2], Bing Tang [1,2,*], Jincheng Luo [1,2] and Jiaming Zhang [3,*]

[1] School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China
[2] Hunan Key Laboratory for Service Computing and Novel Software Technology, Hunan University of Science and Technology, Xiangtan 411201, China
[3] School of Information Engineering, Wuhan University of Technology, Wuhan 430070, China
* Correspondence: btang@hnust.edu.cn (B.T.); jiamingzhang@whut.edu.cn (J.Z.)

Abstract: In recent years, modern industry has been exploring the transition to cyber physical system (CPS)-based smart factories. As intelligent industrial detection and control technology grows in popularity, massive amounts of time-sensitive applications are generated. A cutting-edge computing paradigm called edge-cloud collaborative computing was developed to satisfy the need of time-sensitive tasks such as smart vehicles and automatic mechanical remote control, which require substantially low latency. In edge-cloud collaborative computing, it is extremely challenging to improve task scheduling while taking into account both the dynamic changes of user requirements and the limited available resources. The current task scheduling system applies a round-robin policy to cyclically select the next server from the list of available servers, but it may not choose the best-suited server for the task. To satisfy the real-time task flow of industrial production in terms of task scheduling based on deadline and time sensitivity, we propose a hierarchical architecture for edge-cloud collaborative environments in the Industrial Internet of Things (IoT) and then simplify and mathematically formulate the time consumption of edge-cloud collaborative computing to reduce latency. Based on the above hierarchical model, we present a dynamic time-sensitive scheduling algorithm (DSOTS). After the optimization of DSOTS, the dynamic time-sensitive scheduling algorithm with greedy strategy (TSGS) that ranks server capability and job size in a hybrid and hierarchical scenario is proposed. What cannot be ignored is that we propose to employ comprehensive execution capability (CEC) to measure the performance of a server for the first time and perform effective server load balancing while satisfying the user's requirement for tasks. In this paper, we simulate an edge-cloud collaborative computing environment to evaluate the performance of our algorithm in terms of processing time, SLA violation rate, and cost by extending the CloudSimPlus toolkit, and the experimental results are very promising. Aiming to choose a more suitable server to handle dynamically incoming tasks, our algorithm decreases the average processing time and cost by 30% and 45%, respectively, as well as the average SLA violation by 25%, when compared to existing state-of-the-art solutions.

Keywords: edge computing; task scheduling; time-sensitive; real-time systems; IoT

check for updates

Citation: Zhang, Y.; Tang, B.; Luo, J.; Zhang, J. Deadline-Aware Dynamic Task Scheduling in Edge–Cloud Collaborative Computing. *Electronics* **2022**, *11*, 2464. https://doi.org/10.3390/electronics11152464

Academic Editor: Vijayakumar Varadarajan

Received: 29 June 2022
Accepted: 1 August 2022
Published: 8 August 2022

1. Introduction

With the intelligent upgrade of the manufacturing industry, more and more enterprises are exploring smart factories in the Industry 4.0 era based on CPS [1], a comprehensive computing, network, and physical environment of a multi-dimensional complex system [2], which achieves real-time sensing, dynamic-control, and information services of large engineering systems. CPS realizes the integrated design of computing, communication, and physical systems, which cannot be separated from the powerful arithmetic support of cloud computing. As the representative of the most advanced technology in IT [3], cloud computing stores and manages a massive amount of data while providing effective computing resources. However, since cloud servers have high bandwidth latency and communication waste, it may not be the most appropriate choice for low-latency request access and processing. To this end, for time-sensitive applications and a computer-intensive workload, adopting cloud computing for processing is not the most appropriate option [4,5]. To address the issue of time-sensitivity, edge computing, which may deliver computing services in the vicinity of a data source, makes the system efficient and dependable. With the rise of smart factories, applications with high latency requirements such as industrial image detection, character code detection, furnace temperature flame recognition, and remote mechanical manipulation generate numerous time-sensitive service requests. Consequently, real-time data computing is gradually moving to edge computing, where applications are processed close to the data source, leveraging the computing power of edge servers and heterogeneous end devices.

Figure 1 shows the framework structure of edge-cloud collaborative computing. When a user initiates a task request, the management platform is responsible for dispatching the task to the corresponding server for processing in a timely manner. At the same time, the management platform performs well in terms of resource allocation, load balancing, and security work, while monitoring the task and the server all the time. An effective task scheduling policy is inevitable for efficient task execution, and the management platform will prefer to allocate tasks to the edge server rather than the cloud because of the distance. If edge servers are insufficient for the task, cloud resources will be summoned or tasks will be assigned to the cloud server for execution. The data for task processing comes from the information produced by sensors and actuators in the IoT.

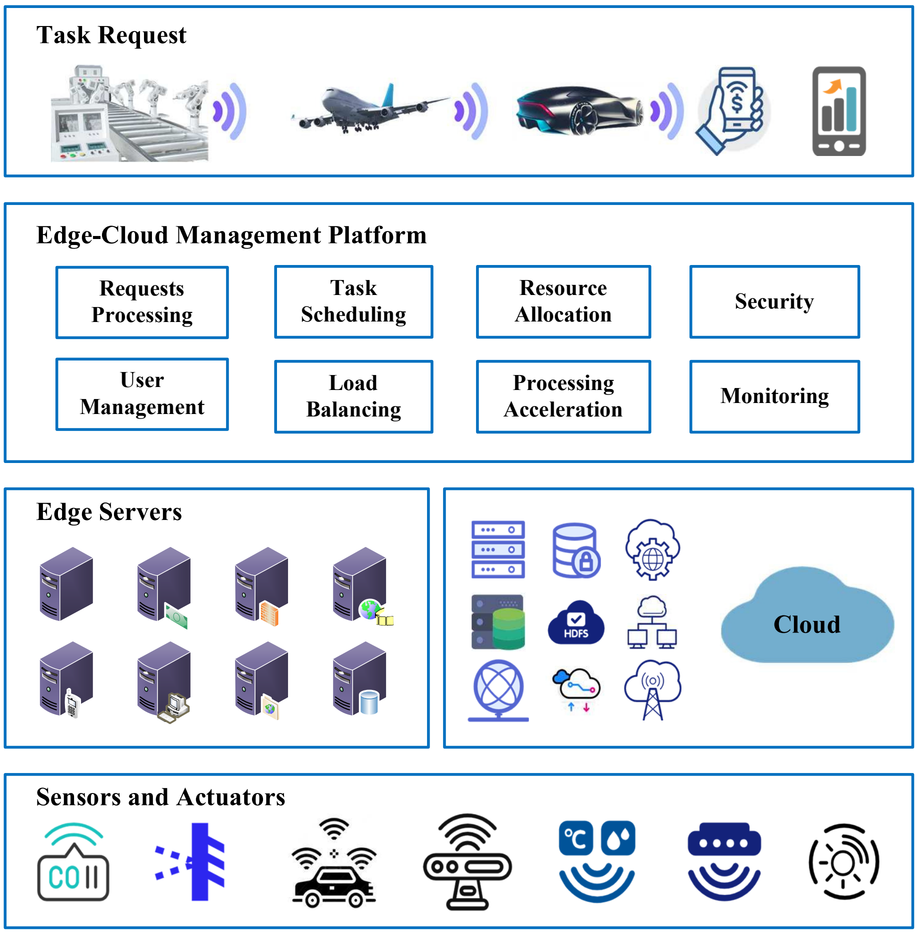


Figure 1. Layered architecture for edge-cloud collaborative computing.

1.1. Motivations

Large amounts of data created by industrial IoT devices may be analyzed at the data source using the edge computing paradigm, which decreases data transfer and network traffic between cloud servers and IoT devices and increases the efficiency and reliability of IoT edge systems. Edge computing enables the provision of scalable, cost-effective distributed computing power over the network, allowing users to access the various IT resources they need at any time and from any location simply [6–9]. The edge layer is an intermediary layer that resides between the terminal device and the cloud platform and is typically used in edge servers close to terminal devices. Further more, edge servers are responsible to handle the user requests from the terminal and shorten the communication consumption of user requests, which is extremely significant for processing time-sensitive applications.

However, the characteristics of edge computing systems of IoT, such as decentralization, heterogeneity, and dynamics, pose a huge challenge to task scheduling, resource allocation, and load balancing. Moreover, the constrained resources and distributed job assignment of edge servers make resource allocation and provisioning more challenging [10]. Virtual power plant (VPP) [11,12] aggregates and distributes power sources for resource optimization through edge intelligence and IoT technologies, requiring real-time and the timely scheduling of resources using edge computing [13]. Smart IoT applications, such as intelligent remote control in smart factories, generate real-time data all the time, and we must accept the dynamic arrival of data streams and respond to requests in a short period of time in order to ensure the safety of personnel and the proper functionalities of the equipments. A failure to complete a task within the predetermined time constrain might cause serious consequences. In this situation, it is extremely principal to decrease the latency and computational cost through reasonable task scheduling and resource allocation, to satisfy user dynamically changing requirements with the least consumption.

However, most prior research and optimization approaches emphasize on-task offloading [14–16] and resource allocation [17] without paying enough attention to improving task processing efficiency through effective task scheduling.In addition, these initiatives are more concerned with energy [18] consumption or the cost [17] of task execution and do not evaluate how to satisfy users' time-sensitive demands through improved techniques from the user's perspective.

We argue that the further development of edge-cloud collaborative computing should pay more attention to the dynamic changes and time-sensitivity of tasks, and of the server load capacity, on the basis of dispersed computing paradigm. In order to achieve higher efficiency, we break the tightly coupled structure of the control system by optimizing the edge-cloud collaborative computing system architecture.This paper designs a task-processing model that maximizes user satisfaction from the perspective of the time-sensitive demand, taking into account the deadline for request processing. Experimentally, this scheduling strategy proves to be effective in improving urgent tasks in task-intensive scenarios.

1.2. Contributions

In order to reduce edge computing processing latency and fulfill user deadlines, novel task scheduling and load balancing are proposed in this paper. The suggested time consumption status architecture allows for the flexible development, deployment, and scheduling of dynamic tasks with time constrained to achieve real-time performance. System models of this paper can be flexibly applied in IoT scenarios in an edge-cloud collaborative computing environment. The main contributions of this paper are listed as follows:

- We modeled the architecture of edge-cloud collaborative computing system and propose predicting the task execution time by obtaining the task submission queue, the waiting queue, and the execution queue of the server by considering the server's comprehensive execution capabilities.
- We proposed to use a dynamic time-sensitive priority algorithm (DSOTS) and a dynamic time-sensitive scheduling algorithm with greedy strategy (TSGS) to schedule

dynamically arriving, time-sensitive tasks based on a task-specified deadline in the edge-cloud collaborative environment.
- We conducted evaluations and performance comparisons by employing the CloudSim-Plus kit to simulate the scheduling problem of dynamically arriving tasks in edge-cloud collaborative computing scenario. The experimental results have proven the superiority of the TSGS strategy compared to recent studies.

The remainder of this paper is organized as follows: related work has been summarized in Section 2. In Section 3, the system model and edge-cloud collaborative computing architecture are described. Section 4 introduces the overall process and prediction method of the dynamic time-sensitive scheduling algorithms. The simulation experiment results are discussed in Section 5. Finally, Section 6 concludes the whole paper.

2. Related Work

Edge-cloud collaborative computing is advanced distributed computing paradigms that is still in its early stages and will require more time to be mature [19]. There has been some work undertaken on task scheduling and resource allocation over the past few years [20–22]. Due to the popularity and commercial adoption of these two distributed computing paradigms, researchers of task processing in edge and cloud computing pay more attention to deadlines [23] and being cost-aware [24]. In this section, on the one hand, we discuss task scheduling in edge and cloud computing paradigms in smart factories to define the research gaps. Furthermore, we investigated a few task scheduling and load balancing techniques, along with their limitations.

2.1. Edge-Cloud Collaborative Computing In IIoT

As the Industrial Internet of Things (IIoT) is an important data source and an application scenario for edge-cloud collaborative computing, IoT-related cloud and edge computing research has become prevalent [25–29]. Research in edge-cloud collaborative computing technology tends to focus on task offloading [14], task scheduling , and real-time data processing [30] in a hierarchical and hybrid environment [21,29,31,32].

Wang et al. [33] were introduced to edge computing in industry , which emphasized the prominent contribution of edge computing nodes (ECNs) to real-time computing and the protection of security beyond computation. However, our work combines edge and cloud, redesigning the edge-cloud collaborative framework to provide real-time computing for IIoT devices and CPS-based applications that are time-sensitive. Plenty of related work on task offloading has been done from energy efficiency [34,35]. Different models of task offloading in edge computing and the hierarchical relationship between edge computing and cloud computing are investigated in [36], which also provides various methodologies for task offloading under diverse task requirements. In [35], the original random problem is transformed into a deterministic optimization problem by using stochastic improvement technology, and then an energy-efficient dynamic offloading algorithm called EEDOA is proposed, which makes a task-offloading decision with polynomial time complexity and is similar to the minimum transmission energy consumption. In [37], an SDN-based edge-cloud interaction is presented to address streaming big data in IIoT environment, in which SDN provides efficient middleware support by leveraging Tchebycheff decomposition for SDN flow scheduling and routing. As a result, the trade-off between energy efficiency and latency has been decreased, as has the trade-off between energy efficiency and bandwidth.

However, these works do not describe how offloaded tasks are then handled, nor how to carry out task scheduling as well as resource allocation for offloaded processes from user′ perspectives, such as SLA violation and cost. When researchers study the scheduling and resource allocation of offloaded tasks in industrial scenarios, they often focus on the heterogeneity of the offloaded tasks requirements (energy, cost, and so on) and have limited MEC capabilities. Therefore, considering the deadline from the user′s perspective is a major breakthrough.

Deng et al. [38] apply artificial intelligence to optimize resource allocation in mobile edge computing, maximizing the trustworthiness gain of services by dynamically generating appropriate resource allocation schemes using the reinforcement learning method, ultimately improving network performance by 20%. In another work, Sodhro et al. [2] demonstrate a forward central dynamic and available approach (FCDAA) by adapting the operating duration of sensing and transmission processes in IoT-based portable devices and a system-level battery model by evaluating the energy dissipation in IoT devices. They succeeded in enhancing energy efficiency and battery lifetime. In [23], Ranesh et al. described a hierarchical framework of fog-cloud computing, as well as the pass-through fog device to the cloud server workflow for tasks generated from endpoints. However, the complicated hierarchy leads to a cumbersome architecture. Moreover, they presented resource allocation and provisioning algorithms based on resource ranking, which achieved some optimizations for latency and processing time, but the efficiency of meeting user deadlines is not obvious in the experiment.

2.2. Task Scheduling Technology

The research on task scheduling in the edge cloud scenario has attracted an increasing amount of attention. However, there is still a lack of more effective strategies for dynamic task arrival and dynamic changes in user needs. In addition, we integrate the hierarchical and hybrid nature, as well as the load balancing of edge servers, by considering the communication strain arising from the high throughput of cloud computing, which is unprecedented.

There are currently a number of efforts that have been made regarding links in the edge-cloud collaborative computing workflow, which usually consider optimization from the points of view of energy [35], bandwidth [20], cost [24], etc. Table 1 summarizes some important research on edge-cloud collaborative computing issues.

Table 1. Comparison of existing dynamically algorithms.

Recent Works	**Main Concerns**	**Techniques**
Zhang et al. (2019) [14]	Offloading reliability	Deep Q-Learning
Gu et al. (2019) [39]	Energy consumption	Matching-theoretic
Ibrahim et al. (2020) [29]	Energy consumption	AES and balancing algorithm
Li et al. (2019) [31]	Energy consumption	Two-phase algorithm
Wang et al. (2020) [18]	Energy consumption	0–1 Integer programming
Zhao et al. (2019) [21]	Energy and service time	Deep reinforcement learning
Kaur et al. (2019) [40]	Energy and interference	Integer linear and multiobjective optimization
Puthal et al. (2018) [41]	Balancing efficiency	EDCs for efficient load balancing
Luo et al. (2020) [17]	Cost-effective	Federated edge learning (HFEL)
Xia et al. (2019) [24]	Cost-effective	EDD-IP and EDD-A
Wu et al. (2018) [42]	Processing time	Extended Lyapunov technique
Manaouil et al. (2020) [43]	Processing time	Microservice scheduling
Wang et al. (2019) [21]	Capacity constraints	Deep reinforcement learning
Deng et al. (2020) [38]	Services' trustworthiness	Reinforcement learning
Naha et al. (2020) [23]	Deadline constraints	Resource ranking and provision of resources
Kannan et al. (2019) [44]	SLA, high throughput	Regression prediction, sequence
Zhang et al. (2021) [45]	QoS	Machine learning
The proposed	Processing time, SLA violation, and cost	TSGS and greedy strategy

Li et al. [31] describe a fog-cloud computing model based on an SDN controller, which assigns tasks to servers after ranking. However, as indicated by [23], the complex layered structure leads to redundant latency, and their implementation of single-threaded file transfers generates redundant data transfer times and does not take into account the impact of the greedy algorithm on server load pressure, which may be less friendly to the server and to time-sensitive applications that arrive later. In [40], Kaur et al. provided a Kubernetes-based energy and interference-driven scheduler (KEIDS), to manage containers on edge-cloud nodes while reducing energy consumption. They also formulated task scheduling using integer linear programming based on multi-objective optimization. However, the huge time overhead associated with iteration may be unbearable for users.

GrandSLAm [44] makes an effort to execute each request queued in the microservice phase in a manner that maximizes sharing while maintaining end-to-end latency guarantees. Dynamic batching and slack-based (SLA, the partial deadlines at each microservice stage, which every request needs to meet at so that end-to-end latency targets are not violated) request reordering are two methods employed by GrandSLAm to maximize throughput. However, the hyperparameters SLA confront a tough adjustment issue during the computation of microservice stage slack, which leads to instability in the correct scheduling rate. GrandSLAm [44], Vanilla Kubernetes scheduler [43], and Sinan [45] introduce scheduling frameworks based on machine learning and deep learning that enhance scheduling methods under different task grouping protocols in microservices. In addition, most of the current work is focused on energy consumption during task execution. It is necessary to consider alternative ways to optimize real-time task processing in terms of task scheduling to better meet the time-sensitive needs of dynamic tasks, which yields a challenging problem of a combinatorial nature. Based on the edge-cloud collaborative hierarchical model for processing tasks in smart factories, we optimize the architecture of this model, and design a time-sensitive task scheduling algorithm based on this model, which considers the highly time-sensitive nature of dynamically changing user requests, whole load-balancing reduces the pressure on the server and congestion of waiting tasks. This methodology is designed to suit the demands of users in terms of meeting deadlines, improving task success rates, and reducing task processing time and cost. It compensates for several shortcomings in prior work to a significant extent.

3. System Model

In this paper, we provide a hybrid hierarchical architecture for edge-cloud collaborative computing. This section covers the structure of the edge-cloud mechanism, the task scheduling strategy for edge-cloud collaborative computing, and the details of the overall system architecture for scheduling optimization.

3.1. Overview of Edge-Cloud Collaborative Computing

It is a highly efficient approach to choose a suitable server for the task to be processed with an excellent task scheduling strategy. In this method, we can reduce the latency of cloud computing while ensuring that the task is completed within the stringent deadline. Dispatching tasks to the Cloud could be avoided, and the extra bandwidth overhead and execution cost could be economized if we choose an appropriate edge server to handle time-sensitive processes.

As shown in Figure 1, IoT applications deal with users' processing requirements and serve them according to their demands. Users from other devices such as machines in smart factories, and automobiles, will produce service requests, and the required metadata are provided by sensors and actuators. Edge servers will manage the underlying middleware and process data. In the edge-cloud collaborative architecture, the cloud infrastructure is responsible for the long-term storage of the processed data and application outcomes. To process data at the edge, the middleware must have the capability of real-time data handling. In this processing architecture, the connections between various types of devices (end devices, edge servers, and the cloud) are maintained mostly through industrial networks (i.e., wired/wireless network).

The edge servers act as the main processing module. On the one hand, they must receive input data from edge sensors and perform pre-processing, and on the other hand, they are responsible for using available resources or retrieving the necessary information from the cloud to process the data and return the results to users. Edge servers in close proximity process data from end devices, but their storage, Ram, bandwidth, and processing capacity are limited. When an edge server receives a task-processing request, it will initially attempt to use its currently available information for processing. If the available resources are completely utilized by the running applications, the task may be forwarded to neighboring peer edge servers for processing. In this situation, the edge servers should periodically transmit a "heartbeat" to the broker to update the broker on their status and available resources. In the three-tier architecture of edge-cloud collaborative computing, the end devices are connected to the edge server and the edge servers are connected to the cloud.

Table 2. Definition of different types of time in edge-cloud collaborative computing system.

Types of Time	Definition
T_{wait}	Time for the task to wait on the server
T_{exec}	Time for the task to execute on the server
$T_{transfer}$	Time for the task to transfer data to the server
$T_{totalExec}$	The time it takes for the task to be completed
T_{exec_Edge}	Task execution time on the edge server
T_{exec_Cloud}	Task execution time on cloud
T_{comu_Edge}	Communication time between devices and edge server
T_{comu_Cloud}	Communication time between edge server and cloud

3.2. Task Execution Model

The edge server requires Datacenter as the primary container to run, and the Host is established to run the server utilizing virtualization software to provide the CPU and memory resources necessary for the edge server. Correspondingly, edge servers also need to get access to graphics card, Datastore and Network connectivity from their higher-level Host. In the Space-Shared scenario, tasks are executed according to the first-in-first-out (FIFO) rule, a machine can only process one task at a time, and if a task is executing on this machine when the other tasks arrive, then the task must wait for the task on the machine to finish before it can start executing [46].

In edge-cloud collaborative environment, tasks generated by mobile devices (MDs) can be offloaded to MEC servers for computing locally or leveraged to cloud resources. Table 2 shows different types of time in edge-cloud collaborative computing system. However, sending tasks to the cloud requires extra communication time and creates bandwidth pressure. The temporal structure of edge-cloud collaborative computing is shown in Figure 2, and the total task execution time (from the initiation of the request to receiving a response) can be formulated as Equation (1),

$$T_{totalExec} = T_{comu\_Edge} + T_{exec\_Edge} + T_{comu\_Cloud} + T_{exec\_Cloud} \tag{1}$$

where $T_{totalExec}$ is the total time from when the task request is sent to the time the user receives the result of the calculation. The tasks need to spend T_{comu_Edge} to establish a connection with the server. After the task reaches the server through the connection, it will not start until all the previous tasks are finished, if there are tasks being executed on a server. After the task starts execution, the actual execution time T_{exec_Edge} varies due to the size and difficulty of the task and the processing power of the server. If there is no server nearby that can complete the task within the user's desired deadline, the server will request resources from the cloud, which increases the time it takes for establishing connection with the cloud T_{comu_Cloud}, and the task will take T_{exec_Cloud} to execute the remaining tasks while still on the cloud.

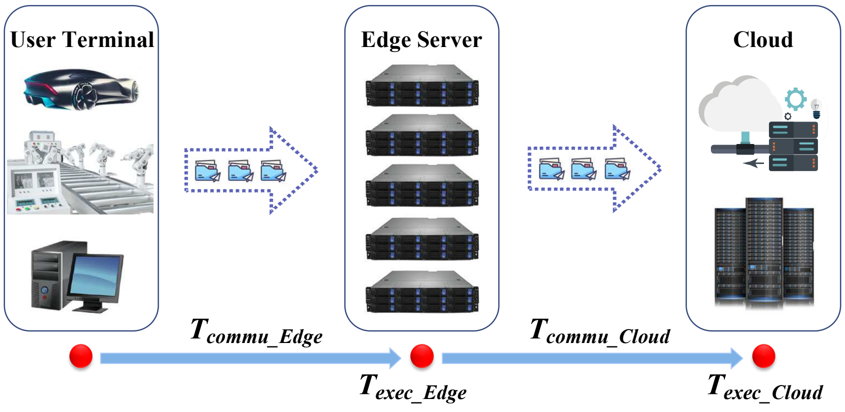


Figure 2. Task execution phase.

With the processing power and bandwidth available, we need to reduce the network traffic of the core network as much as possible by having the computations finished at the edge. Especially for time-sensitive applications, we should finish the task and return the result before the deadline of the task. When a task is to be executed on an edge server, the execution time of the task is calculated by Equation (2).

$$T_{exec\_Edge} = T_{wait} + T_{transfer} + T_{exec} \tag{2}$$

T_{comu} includes the time to send the request and the time to receive the result,

$$T_{comu} = T_{Device\_to\_Server} + T_{Server\_to\_Server} + T_{Server\_to\_Device} \tag{3}$$

When a task request is submitted, it must first spend $T_{Device\_to\_Server}$ to establish a connection with the allocated edge server and then provide user data to it as execution parameters. Since we use a separate thread to transfer data in our model, the $T_{transfer}$ is almost negligible. Allocating the task to an appropriate server for execution must be implemented after being offloaded.In our scheduling system, disposing it on the nearest machine for execution is prioritized, which cuts down on communication time. If the execution cannot be completed on the nearest server within the specified stringent deadline, the agent scheduling center must dispatch the application to a nearby server that can perform the task in time for immediate execution, utilizing a server communication. Furthermore, selecting a short and quick task-waiting queue to reduce the task-waiting time is a sensible choice as it takes into consideration the time that it takes to transfer data to the server before tasks begin executing. Despite the fact that jobs must wait for CPU, data transfer happens through multiple separate threads immediately as soon as jobs are submitted. The last $T_{Server\_to\_Device}$ is the time it takes for the computation results to be returned to the user from the server once task execution is complete. When an edge server receives a request from a user, it consults the cloud for necessary information for application processing. To cope with time-sensitivity, this information-retrieval operation will be executed as a separate process. Resource allocation and task scheduling are implemented by the agent center, and the edge server only needs to focus on processing tasks and does not need to spend time on security and other matters. Hence, this paper integrates the complete task-processing flow of edge computing and improves the high availability and efficiency of computing scenarios through model optimization. On the basis of the optimized model, an effective scheduling strategy is developed to further improve the efficiency of task processing.

4. Proposed Scheduling Algorithm

In this section, we present the specific implementation of the dynamic time-sensitive scheduling algorithm and the dynamic time-sensitive scheduling algorithm with a greedy strategy.

Table 3. Notations.

Symbol	Meaning
s_i	The available storage of the server S_i
B_i	The available bandwidth of the server S_i
R_i	The available Ram of the server S_i
C_t^{λ}	Different types of cost, $\lambda =$ {CPU, RAM, Storage, Bandwidth}
PT_i	The prediction time of $task_i$
CEC	The comprehensive execution capability of servers
$task_i$	The dynamically arrived task that need to be submitted to a server
$MIPS$	The available MIPS of the server S_i
$STNum_j$	The number of tasks waiting on $server_j$
$TasksList$	The list of tasks that have been submitted without ranking $Task = \{task_1, task_2, ..., task_n\}$
$ServersList$	The list of servers that have been submitted without ranking $S = \{S_1, S_2, ..., S_n\}$
$TasksWaitingList$	Submitted tasks that are waiting to be executed on the server

4.1. The Overview of the Algorithm

In this section, we present a solution for the dynamical arrival applications in the edge-cloud collaborative environment. Table 3 shows all notations. To meet the deadline objective while taking into account dynamic client requirements in the edge-cloud collaborative environment, scheduling optimization will be performed according to the following steps.

1. *Submission strategy*. Submitting tasks to the cloud for execution, and the edge server requesting resources from the cloud, massively increase the communication time and cost overhead; we need to maximize the use of the edge server to execute tasks, and to try to intercept tasks to complete execution on edge servers to avoid submitting to the cloud. When all servers are unable to accomplish the work before the user's desired deadline, the edge server must request resources from the cloud considering the processing capacity, the available bandwidth, and the response time of those resources. It is necessary to consider when resources are completely unavailable within the edge infrastructure. In such a case, the agent center tries to submit the application to the cloud before generating the resource unavailability message.
2. *Server's comprehensive execution capability*. There are many factors that influence the server's processing speed, so we need to consider the bandwidth, memory, processing capabilities of the server, and degree of load balancing, to obtain the CEC of the server.
3. *Execution time prediction*. Predict the execution time based on the server's comprehensive execution capability and assign tasks to the appropriate edge server or cloud for execution so that tasks are completed before the user's desired deadline.
4. *Dynamic task scheduling*. (1) The dynamic time-sensitive scheduling algorithm—DSOTS. Different tasks in a smart factory have varying time sensitivity; therefore, we select specific servers according to the requirement of different types of tasks, maximize the use of computing resources, and design the DSOTS to meet the needs of users from a deadline-aware perspective. (2) A dynamic time-sensitive scheduling algorithm with a greedy strategy—TSGS. Depending on the DSOTS, we designed the TSGS based on the greedy strategy after sorting the servers according to the defined rules to schedule dynamic tasks and load-balance the servers.

On the other hand, the problem of a user's dynamic behavior may be handled through the following steps:

- **Step 1:** Estimating the time spent on edge computing versus cloud computing.
- **Step 2:** Ranking the available resources and the comprehensive execution capability of the edge servers.
- **Step 3:** Building a virtual waiting queue of uncompleted tasks and pushing the dynamic task in.
- **Step 4:** Ranking the estimated execution time of tasks considering the anticipated execution time in the waiting queue.
- **Step 5:** Assigning the task to the server with the shortest predicted time.

4.2. Prediction of Execution Time

When obtaining the CEC of the server, we consider the server's storage, ram, bandwidth, and MIPS, and assign different weights α, β, μ, ν to $MIPS$ (Mip_i), bandwidth (B_i), storage (S_i), ram (R_i), respectively. According to their importance, their relationship can be expressed by the following equation:

$$CEC_i = \alpha Mip_i + \beta B_i + \mu S_i + \nu R_i \tag{4}$$

To determine the computational hyperparameters (α, β, μ, ν) of CEC , we need to first train a deep learning LSTM model based on a priori knowledge and use the model to fit to obtain these four hyperparameters. CEC is equivalent to an evaluation of the server's execution performance.

Predicting the execution time of $task_i$ on $server_j$ is the inevitable content of the dynamic scheduling task to make it finish within the desired deadline. We employ Algorithm 1 to predict the execution time by considering the server and task's own conditions comprehensively.

- We require the waiting time and execution time for the task that is waiting at $server_j$ to predict how long it will take the server until $task_{i-1}$ is finished; this is called the waiting time. Then, add the execution and response time of the $task_i$ itself. $WaitingTime$ can be stored and calculated in two ways: as a task property, or by calculating the execution time of all the previous tasks in each cycle to estimate the difference between them in terms of space cost and time cost. Due to the extreme urgency, the separate threads for data transfer allow $T_{transfer}$ to be ignored, reducing the impact of space costs. In this paper, all this information ($WaitingTime$ and $ExecutionTime$) is set as the properties of $task$.
- Whenever the status of the task is changed, for example, when it enters the execution state from the waiting queue, the system will update this information and send a "heartbeat" to the agent center to update its latest status. Of course, this information is not known until it is submitted to a server because it is necessary to consider the server's comprehensive execution capacity and waiting queue information to know $WaitingTime$ and $ExecutionTime$.

Algorithm 1 Prediction of execution time.

Input: *TasksWaitingList <task>: tasks waiting on the server; ServersSubmitedList <server>: servers that handle requests; $task_i$: the i-th arriving task*

Output: $ServersList < server >$

```
1:  $TasksList < task > \leftarrow TasksWaitingList < task >$
2:  $ServersList < server > \leftarrow ServersSubmitedList < server >$
3:  if (ServersList is null ) then
4:      return $Failure$
5:  else
6:      $PT_i \leftarrow 0$
7:      for each $task \in TasksWaitingList$ do
8:          $PT_i \leftarrow PT_i + getExecutionTime(task)$
9:      end for
10:     $PT_i \leftarrow PT_i + getExecutionTime(task_i)$
11:     $PT_i \leftarrow PT_i + getResponseTime(task_i)$
12:     Sort($ServersList < server >$) according to $PT_i$ and CEC
13: end if
14: return $ServersList < server >$
```

4.3. Dynamic Time-Sensitive Scheduling Algorithm—DSOTS

To deal with deadline-aware difficulties, we proposed a dynamic scheduling algorithm based on time-sensitive prioritization, as Algorithm 2 shows. When a task is dynamically submitted, we choose a better server to perform it on. The user can set a desired deadline for a task as long as the task is created. When the agent center receives a task processing request, DSOTS can help to choose the server with the best chance of meeting the user's requested deadline. Figure 3 shows the server selection model of DSOTS, {1 s, 2 s, 3 s, 4 s, 5 s} is the predicted time it takes for servers $\{S_2, S_3, S_5, S_1, S_4\}$ to process the $task_i$, and we sorted the servers into $\{S_2, S_3, S_5, S_1, S_4\}$ according to PT_i. On the other hand, $Deadline_1$ to $Deadline_3$ is the deadline set for the task, and finishing the task before the deadline as much as possible is a guarantee that the failure rate will not be too high. DSOTS will be implemented according to the time priority line in Figure 3, according to the following steps:

- First, $ServersList$ is ranked according to CEC of the server; then, sort the ranked $ServersList$ again according to the predicted time PT_i that the task $task_i$ will take to execute on it. In other words, the server with the shorter task-waiting queue for the user will be given priority on machines with the same CEC. As shown in Figure 3, if there are five available edge servers nearby, the $ServersList$ will be sorted as $\{S_2, S_3, S_5, S_1, S_4\}$.
- When the user-defined deadline is later than the execution time required by the slowest server ($Deadline_3$ is later than the predict time on S_4), then all the servers can satisfy the user's criteria and complete the task ahead of the deadline. In this case, if the task is highly time-sensitive, it is a good idea to choose the server with the shortest predicted execution time to execute. If the urgency of the task is low, we can choose to schedule it on the slowest server, which can save enough server resources for time-sensitive tasks that may occur later. For tasks that have modest time sensitivity, a server can be randomly selected for execution. The amount of seconds in the center, as shown in Figure 3, represents the projected time for various servers to complete the current task. If a task's deadline is $Deadline_3$, larger than the whole expected execution time, we can insert time-relaxing tasks in S_4, time-urgent tasks in S_2, and moderately time-sensitive tasks in $\{S_1 \vee S_3 \vee S_5\}$.
- Nevertheless, if the deadline (such as $Deadline_1$ in Figure 3) is shorter than any server's expected execution time, then all the servers will be unable to complete this task within the specified time. Regardless of the time urgency of the task at this point, the agent center must select the server with the strongest comprehensive execution capability and the shortest predicted completion time to serve the task, just like S_2, and retrieve resources from the cloud platform to complete the task before the user receives the task failure message. If the defined deadline is relatively short like $Deadline_2$ but not too loose, then randomly choosing a server that is just appropriate for the task would suffice.

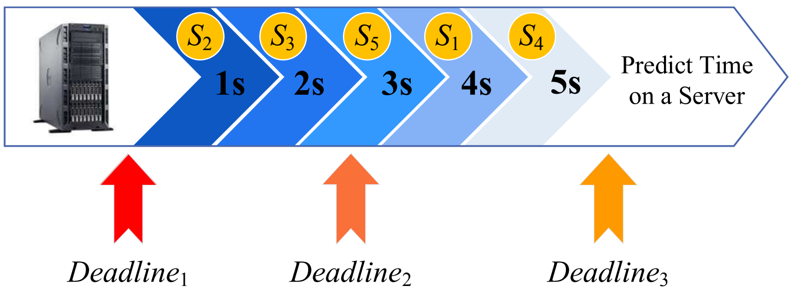


Figure 3. Time-sensitive greedy strategy.

Algorithm 2 DSOTS algorithm.

```
Input: $TasksWaitingList < task >; ServersSubmitedList < server >; task_i;$
1: $TasksList < task > \leftarrow TasksWaitingList < task >$
2: $ServersList < server > \leftarrow ServersSubmitedList < server >$
3: if $(ServersList \neq null)$ then
4:     $deadline \leftarrow$ The user's expected execution time for $task_i$
5:     $TasksList \leftarrow TasksWaitingList.add(task_i)$
6:     sort ($ServersList$) according to each server's CEC
7:     for each $server \in ServersList$ do
8:         $PT_i \leftarrow$ The estimated execution time of $task_i$ on the $server$ by Algorithm 1
9:         $server \leftarrow PT_i$
10:    end for
11:    sort ($ServersList$) according to $PT_i$ that task spend on server
12:    $RSL \leftarrow RankedServersList$
13:    if $deadline \geq PT_{last}$ then
14:        if $task_i$ is Time-Sensitive then
15:            Submit $task_i$ to the first server of the RSL
16:        end if
17:        if $task_i$ is Time-Moderate then
18:            Submit $task_i$ to a random server of the RSL
19:        end if
20:        if $task_i$ is Time-Relaxation then
21:            Submit $task_i$ to the last server of the RSL
22:        end if
23:    end if
24:    if $deadline \leq PT_{first}$ then
25:        Submit $task_i$ to the first server of the RSL
26:    end if
27:    if $deadline$ in ( $PT_{first}, PT_{last}$ ) then
28:        if $task_i$ is Time-Sensitive then
29:            Submit $task_i$ to the first server of the RSL
30:        end if
31:        if $task_i$ is [Time-Moderate $\vee$ Time-Relaxation] then
32:            Submit $task_i$ to the last server that can meet the user's
               expected deadline of the $task_i$
33:        end if
34:    end if
35: end if
36: Update the $ServersList$
```

4.4. Dynamic Time-Sensitive Scheduling Algorithm with Greedy Strategy—TSGS

As shown in Algorithm 3, in order to better distribute tasks and perform better load balancing of the server clusters, we seek to improve the dynamic scheduling algorithm based on the greedy algorithm. The servers are ranked initially. At the same time, we must examine the queue of submitted tasks in order to improve the overall performance and optimization of the server cluster and task-processing queue at the level of the entire task batch. Following DSOTS algorithm processing, the sequence of servers and task intent processors already have a better allocation decision, based on which the task intent servers are adjusted using the greedy algorithm, and the load balancing of servers is performed.

- To account for dynamically arriving tasks, the processing done here is constructing a virtual processing queue *TasksList* by adding dynamically arriving tasks $task_i$ together with previously submitted tasks to the waiting queue *TasksWaitingList* to better predict the overall execution time; then, the *TasksList* needs to be ranked according to the length of tasks.

- A two-dimensional array needs to be created to store the estimated execution time of tasks on each server based on the mapping of servers to tasks. For each task, the predicted execution time on a particular server can be found using the mapping table. For each task, we can iterate through all servers where the task can be stored.
- The policy of the greedy algorithm is to select the machine with the strongest execution capability and the shortest predicted execution time to execute the current task each time. When traversing to a server, even if there is no task executing on the current server, the task should not be placed on the server directly without considering the processing capacity of the server, and $task_i$ could be submitted to $Server_j$ only when the expected execution time is less than the minimum time in the current record. It will select the one with the shorter task-waiting queue for servers with the same expected execution time to achieve the load-balancing effect.

The TSGS greatly improves the efficiency and accuracy of task scheduling by comprehensively considering the deadline requirements of tasks and server load.

Algorithm 3 TSGS algorithm.

```
Input: $TasksWaitingList < task >; ServersSubmitedList < server >; task_i$
1:  $TasksList < task > \leftarrow TasksWaitingList < task >$
2:  $ServersList < server > \leftarrow ServersSubmitedList < server >$
3:  if $(ServersList \neq null)$ then
4:      $deadline \leftarrow$ The user's expected execution time for $task_i$
5:      $TasksList \leftarrow TasksWaitingList.add(task_i)$
6:      sort ($ServersList$) according to each server's CEC
7:      sort ($TasksList$) according to $task_i$ that task spend on server
8:      for each $server \in ServersList$ do
9:          $PT_i \leftarrow$ The estimated execution time of $task_i$ on the $server$ by Algorithm 1
10:         $server \leftarrow PT_i$
11:     end for
12:     sort ($ServersList$) according to $PT_i$ that task spend on server
13:     $tempServer \leftarrow Server_1$ // update $tempServer$
14:     $minTime \leftarrow$ the optimal value of the current task assignment
15:     for $j$ = 1 to $ServerList.length$ do
16:         if The TasksWaitingList in $Server_j$ is empty then
17:             if $minTime \geq Time(i, j)$ then
18:                 Submit $task_i$ to the $Server_j$ and update $tempServer$
19:             end if
20:         end if
21:         if $minTime > Time(i, j)$ then
22:             Submit $task_i$ to the $Server_j$ and update $tempServer$
23:         else
24:             // Simple load balancing
25:             if $minTime = Time(i, j)$ and $STNum_j < S_{temp}TNum$ then
26:                 Submit $task_i$ to the $Server_j$ and update $tempServer$
27:             end if
28:         end if
29:         Update the task execution queue of servers
30:     end for
31: end if
32: Update the $ServersList$
```

5. Experiments and Result Analysis

In terms of task-processing time; success; and cost, this section provides performance metrics and assessment methodologies for evaluating the performance of edge computing in the intelligent industrial factory. Firstly, we provide the experimental settings and simulation parameters. Afterwards, we produce a simulation program to simulate the dynamic generation of time-sensitive tasks in industrial production, submit them to edge and cloud servers for processing, and evaluate the performance of the proposed algorithm based on the implementation of the edge computing hierarchical model designed in Section 3.

5.1. Experiment Scenario and Configuration

Application services hosted under the cloud computing paradigm have cumbersome provision, composition, configuration, and deployment requirements. It is challenging to evaluate the performance of cloud-provisioning policies, the application of workload models, and resource-performance models in a repeatable fashion under varying system and user configurations and requirements. We reconstructed the functionality of the CloudSimPlus toolkit [47], which is extensively employed in various distributed computing simulations, to simulate a realistic edge computing scenario in smart industrial production and then measure the effectiveness of our approach. Experiments were carried out by changing the number and fluency of application submissions, as well as the deadline strain.

5.2. Baseline Approaches

In order to evaluate the performance of **TSGS**, the time-sensitive greedy scheduling strategy proposed in this paper, we compared it with the following four baseline approaches.

- **DSOTS**, dynamic scheduling based on a time-sensitivity algorithm proposed in this paper.
- **OEC-RR**, an ordinary round-robin edge computing scheduling, which cyclically selects the next server from the broker server list based on a round-robin policy.
- **SAE-CEC** [31], a two-phase algorithm based on threshold strategy with latency constraints, which is one of the most recent works latency-aware algorithm.
- **GrandSLAm** [44], a microservice execution framework that can accurately estimate the completion time of the requested microservice phase, which focuses on achieving high throughput while maintaining SLAV.

The latency-awareness mainly concerns the algorithm energy consumption and satisfaction degree. In our proposed technique, we consider the available bandwidth, the available process power, and the response time of the device. For task scheduling, we take into account application requirements as well as dynamic changes in the deadlines specified by users when scheduling tasks.

5.3. Metrics

We randomly generate different types of tasks with different priorities from 100 terminals at different times to ensure that the waiting time and processing time of each task are random and set a randomly reasonable processing time for them as the deadline. In order to evaluate the performance of the proposed edge-cloud collaborative model and algorithm, we measured the task-processing success rate, the delay, the processing time, and the processing cost. Since the edge computing is specifically for handling time-sensitive tasks, latency and processing time are of vital importance to us. If a task can be completed before its required deadline, the task will be regarded as having been effectively accomplished. Otherwise, even on the highest-performing edge server, we must borrow a peer edge server or request resources from the cloud to accomplish it. The edge server hierarchical model is an innovation for us; we tier the edge servers according to the CEC mentioned above. Tasks are first submitted to the lower level servers for processing, and if the bottom processor does not have enough processing resources, it can apply resources from the higher level servers or submit the service to the upper level. If even then the task cannot be completed, we shall abort the task execution and notify the user of the failure. The SLA violation rate is defined as the proportion of unsuccessfully completed tasks among all tasks submitted during a period of time.

(1) *Delay*: this is the time interval between when the task is submitted to the edge or cloud server and the start time of the task execution d_t, which will vary with the allocation of communication and computing resources.

$$d_t = d_t^{device} + d_t^{edge} + d_t^{Cloud} \tag{5}$$

The latency of a task on a server can be thought of as the execution time of all the previous tasks that are waiting, such as $d_i^{edge} = \sum\limits_{k=0}^{i-1} T_k^{wait}$. Therefore, we need to calculate our total delay time as follows:

$$d_i = \sum_{terminal}^{Cloud} \sum_{k=0}^{i-1} T_k^{wait} \tag{6}$$

(2) *Processing Time*: this needs to be calculated based on the task characteristics (type, deadline, size, and so on) and the CEC of the server.

For processing costs, we took into account the cost of storage and processing and divided it into four parts: the monetary cost of using each Megabyte of RAM in the Datacenter, the monetary cost per second of CPU, the monetary cost to use each Megabyte of storage, and the monetary cost to use each Megabit of bandwidth.

$$C_t = \sum C_t^{CPU} + C_t^{RAM} + C_t^{Storage} + C_t^{Bw} \tag{7}$$

(3) *SLA Violation*: this refers to the proportion of failed tasks in a batch of tasks to the total tasks.

$$SLA = \frac{\sum_N^{j=1} T_{fail_j}}{\sum_N^{i=1} T_i} \tag{8}$$

5.4. Parameter Configuration

In this experiment, we built up 10 edge servers, which have three data centers to provide adequate resources for the host; each host has 2 PEs, and each server can occupy one PE to execute tasks.Because almost all the tasks are performed on the edge servers, we set up only two clouds to provide additional resources, and their topology is random and homogeneous. With such hardware resources, we simulate the task-processing requests sent by intelligent control machines in a genuine smart industrial factory production and test that 100 to 1300 tasks were submitted according to the task submission pattern of [23,48]. The task sent by the user, and the server processing the task, are the two main objects in our experiment. Table 4 shows the resource configuration of the edge server and cloud server, and Table 5 shows the setting of tasks. To ensure the unobstructed operation of the experiment and the accuracy of the data, we keep the smooth network connection and dynamic resource allocation according to [38]. The deadline is the return time expected by the user. In the experiment, the deadline setting is related to the magnitude of the task itself, and a reasonable deadline is essential to ensure the accuracy and adaptability of the experimental results. We set the estimated execution time of the task on the server with average performance as the standard deadline (the value of standard deadline constraint is 1, which represents 45 ms). The other values (0.8, 1.2, etc.) of the deadline represent different levels of demand; the smaller the value, the more lenient deadline, and the easier it is to be satisfied.

Table 4. Resource configuration.

Parameter	Edge Servers Configuration	Cloud DC Configuration
Number	10	1
MIPS (Millions of Instructions Per Second)	8500	50,000
No. of PEs	2	2
No. of Host	2	2
No. of Datacenter	2	10
Bandwidth (Mbps)	10,000	100,000
RAM (GB)	4	128

Table 5. Task parameter setting.

Task Conf.	Value
Average task length (MI)	300 ± 20
Data size (MB)	55 ± 10
Deadline	random
Sensitivity type (priority)	random
PEs Number	2

To ensure the preciseness of the experimental results, we conducted the proposed algorithm using more extreme data than in previous work, testing the optimization capability of the algorithm by increasing the number of tasks, the task-generation frequency, and other factors. In addition, we conform to the Poisson distribution [49], as in Equation (9), when generating random data to ensure the independence and randomness of the task. Where t represents the time interval and n represents the number of tasks expected to be generated at a certain time interval. Since, as mentioned earlier, our standard deadline is 45 ms, we set $\lambda = 45$. Thus, $n \times P(N(t) = n)$ is the number of tasks generated in time t.

$$P(N(t) = n) = \frac{(\lambda t)^n e^{-\lambda t}}{n!} \tag{9}$$

5.5. Experimental Results

This section shows the experimental results under different conditions and demonstrates the comparison of a TSGS strategy with a default policy (OEC-RR), DSOTS, latency-aware SAE-CEC algorithms, and SLA-aware GrandSLAm algorithms. The SLA violation rate, processing time, and task processing cost are investigated, respectively, under varying numbers of tasks, task-arrival frequency, and deadline fluctuation rate. Since GrandSLAm is sensitive to SLA at high throughput; we compare the results with SLAV of GrandSLAm in each set of experiments.

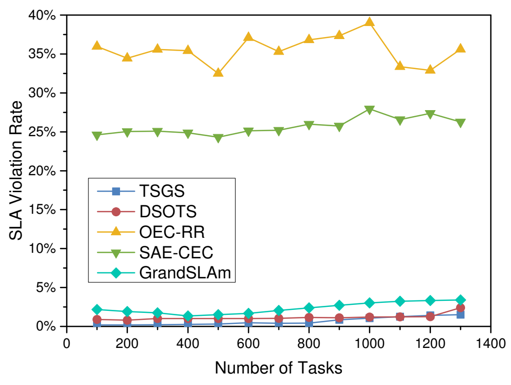
5.5.1. The Impact of Task Number

Figure 4a depicts the progression of the SLA violation rate as the number of tasks increases and when tasks are assigned according to different scheduling policies. If the server fails to complete a task before the user-specified deadline, the task is considered to be failed, and the service-level agreement is violated. The SLA violation rate is the percentage of all tasks that failed. A high SLA violation rate tends to cause task congestion and server overload, which is a devastating blow to the real-time performance of tasks in smart factories. Data show that SAE-CEC does have some optimizations over the default scheduling algorithm, but the DSOTS algorithm reduces the violation rate by 22.13%, and the greedy-optimized TSGS even reduces the SLA violation rate to 1.8%. The violation rate of all other algorithms increases as the task volume rises, while the TSGS and DSOTS violation algorithm considering time sensitivity decreases instead. Since the violation rate of TSGS, DSOTS, and GrandSLAm is too low, it needs to be magnified with Figure 4b to show their detailed values. With the increment of task number, the rates of OEC-RR and SAE CEC are both above 19.26% and hardly changed; TSGS can reduce the violation rate to less than 2%, and the optimization efficiency is better when the number of tasks more significant.As GrandSLAm focuses more on ensuring the improvement of throughput under SLA, the advantage of GrandSLAm will be revealed when the number of tasks increases.As Figure 4b shows, the SLAV ratio of GrandSLAm approaches that of TSGS after the number of tasks exceeds 1100. Each microservice stage slack of the directed acyclic graph (DAG) is calculated by GrandSLAm, and it is then reordered in accordance with $slack_m$. However, due to the influence of the hyperparameter SLA, the largest sharing degree (the actual batch size during execution) after reordering is not stable when computing $slack_m$. However, when the number of instantaneous tasks becomes larger, the impact of the hyperparameter on the largest sharing degree decreases, leading to an increase in efficiency.

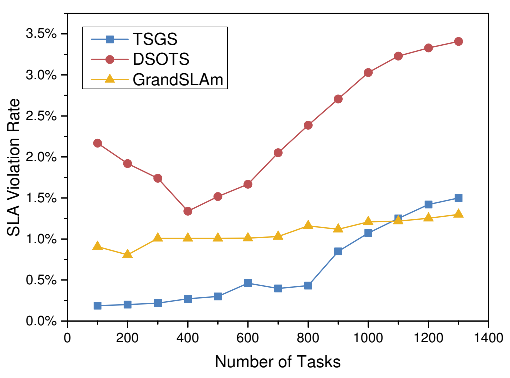
This is because the server is chosen after considering the deadline and time sensitivity of the tasks, which can satisfy the user's demand to a greater extent in TSGS. In addition, the execution and load-balancing design of the control time-insensitive tasks also release adequate processing resources for the tasks arriving later, which ensures the task-waiting queue on the server does not get excessively lengthy when the number of tasks increases, causing the tasks to fail.

The optimization of the task-processing time and task-waiting time of our algorithm is shown in Figure 4c,d. The overall average delay is 63.7% and 59.0% lower in the TSGS algorithm compared with the OEC-RR and SAE-CEC algorithms, respectively. The TSGS strategy can execute the same number of tasks under the SAE-CEC policy in only 64.1% of the time. Since TSGS chooses the appropriate edge server, tasks do not need to wait too long after they are submitted to the server, and the reduced latency results in a shorter execution time of the whole task. Independent threads for data transfer reduce the task execution time by reducing $T_{transfer}$ as well. Tasks are completed in less time, which is the primary reason why TSGS is able to significantly reduce the violation rate and increase the success probability. Moreover, Figure 4e depicts the results of the task-processing-cost trend as the number of tasks increases. Similarly, TSGS and DSOTS exhibit better performance than the other scheduling strategies. Because SAE-CEC only considers the MIPS and ignores other combined processing capabilities, including bandwidth, network congestion may be encountered or a more distant server selected to run the task, which adds to the amount of time it takes to transform data and increases the cost of the bandwidth.

As the number of tasks increases, the time and cost of various task-scheduling strategies increases, but TSGS algorithms perform excellently in terms of user satisfaction; processing time; and cost, which increases users' satisfaction and and provides better service and cost savings for real-time work in smart factories.

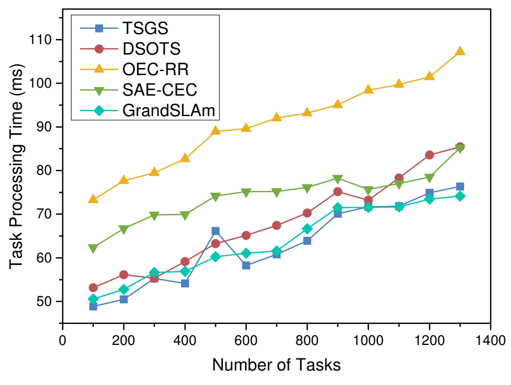


(**a**)

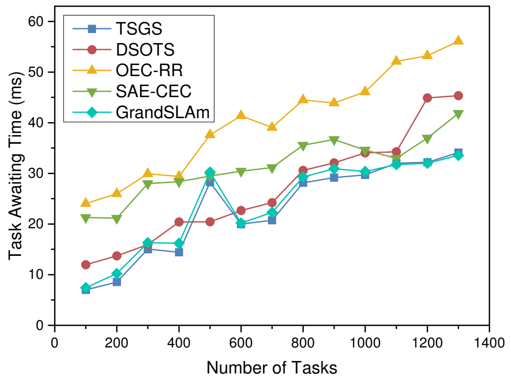


(**b**)

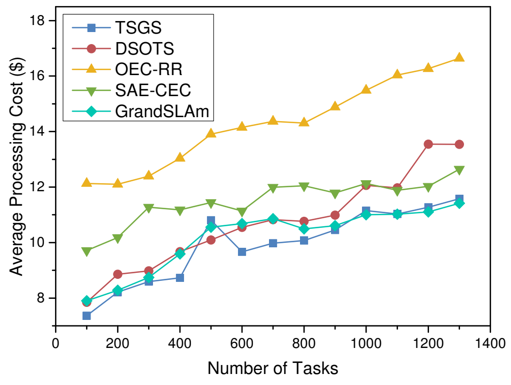
Figure 4. *Cont.*



(**c**)



(**d**)



(**e**)

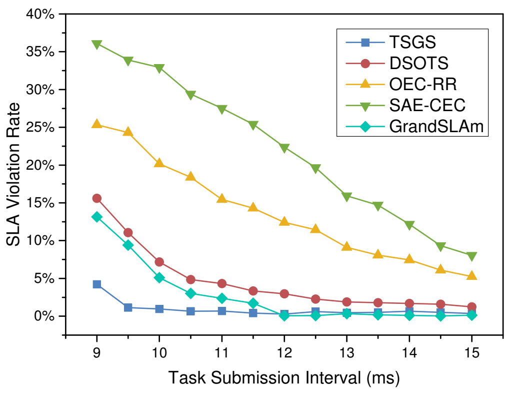
Figure 4. The impact of task number on SLA violation rate, time, and cost. (**a**) SLA violation rate; (**b**) SLA violation rate (DSOTS vs. TSGS vs. GrandSLAm); (**c**) task processing time; (**d**) task waiting delay; and (**e**) task processing cost.

5.5.2. The Impact of Task-Submission Frequency

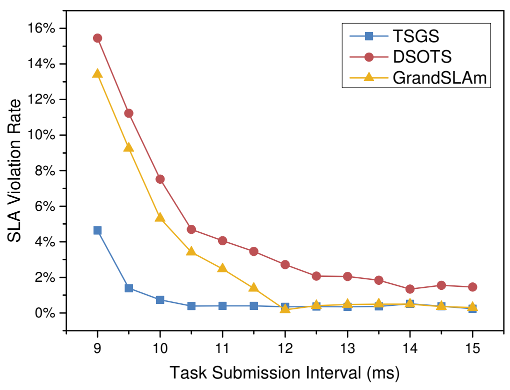
When the time interval of task arrival is reduced, this implies that a significant number of tasks are submitted in a short period of time, causing the server's task-waiting queue to become overburdened and task-waiting times to increase. At this time, it is especially important to choose the right server to handle the tasks and to balance the task-waiting queue of the server reasonably. By altering the time interval of each dynamic request to imitate the frequency of different task dynamic arrivals, we simulated 800 task requests to explore how different task densities affected the performance of the algorithm. The optimization of the TSGS and DSOTS algorithms for dynamic task processing is depicted in Figure 5.

As shown in Figure 5a, the violation rates of different scheduling algorithms all decrease when the frequency of task arrivals decreases. However, since OEC-RR just selects servers cyclically and makes no judgments on task or server attributes, the changes in task density have less of an influence on this method. It is observed that the TSGS algorithm reduces the failure rates by 10–20%, and the SLA violation rate is lower than the other algorithms when the time interval between task arrivals is minimal. The proposed two algorithms are capable of dealing with the situation of dense task arrivals and handling task-intensive workloads with ease, which proves that the adoption of TSGS strategies will lead to higher task-carrying capacity in task-intensive scenarios.

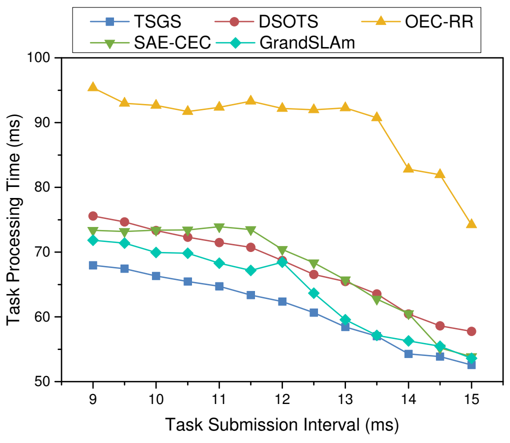
It is illustrated that the average processing time per task reduces as the task submission interval rises in Figure 5c. This is due to the fact that when the task interval increases, the blocking queue correspondingly loosens and the average waiting time for tasks decreases, shortening the task execution time. When the task intensity rises, the task processing cost increases as well, as seen in Figure 5e. Instead, the scheduling strategies of TSGS and DSOTS result in a decrease in the average processing cost per task as the tasks grow more intensive. This is because TSGS creates a separate thread for data transfer while the task is waiting, which reduces the bandwidth congestion cost and cache cost in the task-intensive case; this is very friendly for large-scale task processing in smart factories.



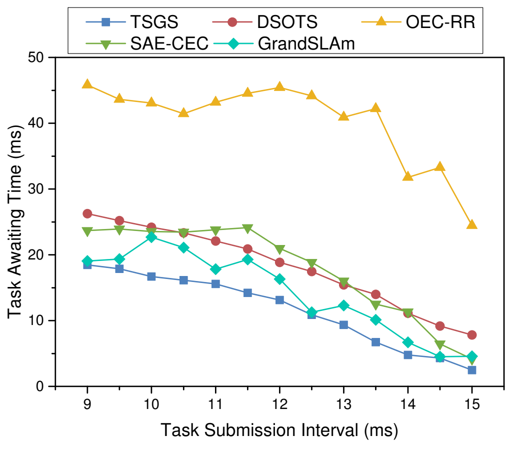
(**a**)



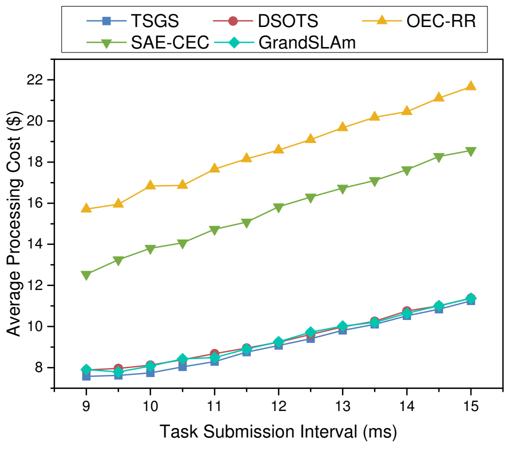
(**b**)



(**c**)



(**d**)



(**e**)

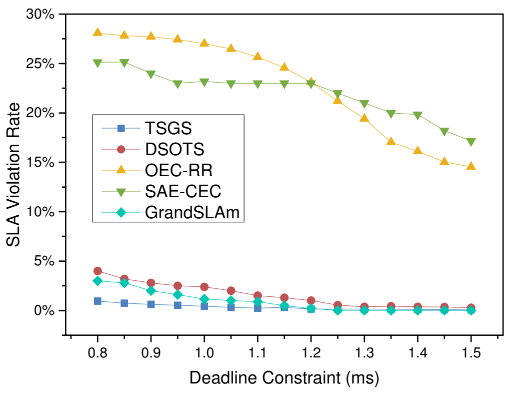
Figure 5. The impact of task submission frequency on SLA violation rate, time, and cost. (**a**) SLA violation rate; (**b**) SLA violation rate (DSOTS vs. TSGS vs. GrandSLAm); (**c**) average processing time; (**d**) average waiting delay; and (**e**) task processing cost.

The trend of each metric of task execution demonstrates the superiority of the TSGS strategy in satisfying task-intensive work scenarios, making it desirable for application to large-scale real-time workflows in smart industrial production.

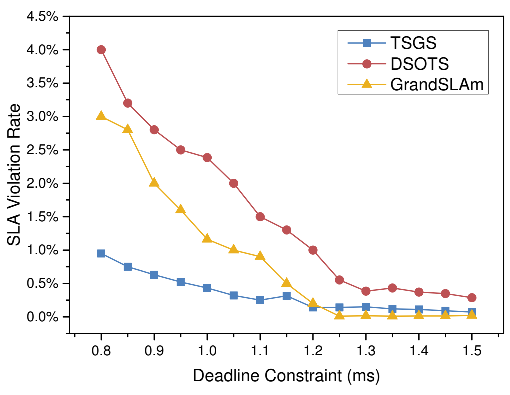
5.5.3. The Impact of Deadline Constraint

To obtain more details about the performance of all algorithms, the violation rate, average execution time, and average processing cost are compared under different degrees of deadline constraints, as Figure 6 shows. The number "1" (representing 45 ms) is set to the standard degree of deadline constraint, and the case of 500 dynamically submitted tasks is tested by varying the degree of deadline constraint. The results show that TSGS and DSOTS scheduling strategies outperform the other two algorithms in all cases. From Figure 6a,b, it is obvious that TSGS can respond flexibly to changes in the deadline strain to improve the processing success rate of time-sensitive applications, and it even reduces the failure rate to 0.95%, which means that TSGS can respond in a timely manner when users post urgent tasks. It is reasonable to accept that, except for the SAE-CEC algorithm, the violation rate of tasks will decrease as the deadline constraint gradually relaxes and the user's completion time requirement is relaxed. SAE-CEC is less affected by external influences, and SLAV decreases slowly. For the reason that the deadline is a demand made by the user, it has no effect on the average processing time or cost of the workflow. The TSGS reduces the processing time by 17.6% compared to SAE-CEC and by 31.3% compared to OEC-RR. However, the processing time and cost of the task do not differ much under the same scheduling policy as well.

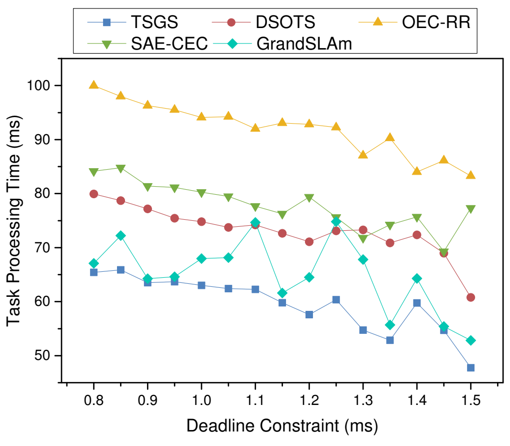
There is no doubt that the TSGS scheduling strategy significantly outperforms the other two algorithms because, in the same case, the TSGS algorithm is able to select a more suitable edge server to complete the task.



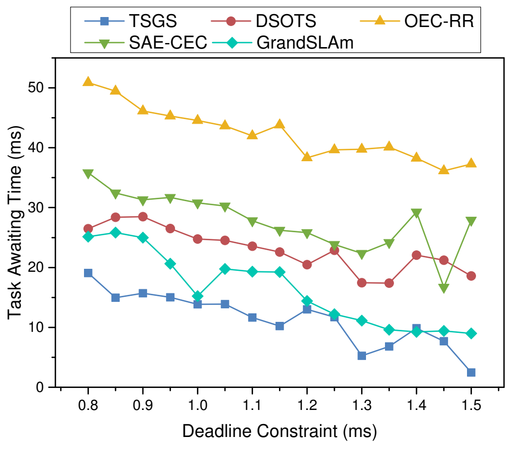
(**a**)



(**b**)

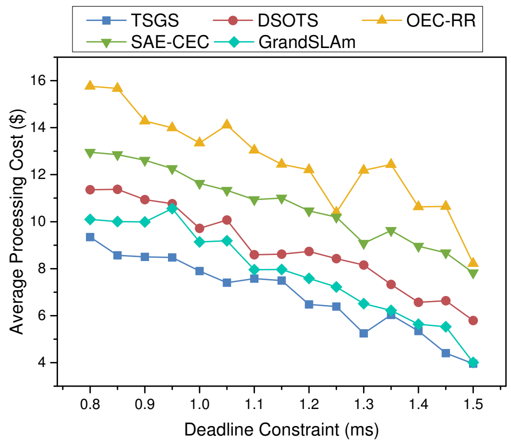


(**c**)



(**d**)

Figure 6. *Cont.*



(**e**)

Figure 6. The impact of deadline changes on SLA violation rate, time, and cost. (**a**) SLA violation rate; (**b**) SLA violation rate (DSOTS vs. TSGS vs. GrandSLAm); (**c**) average processing time; (**d**) average waiting delay; and (**e**) task processing cost.

6. Conclusions

Edge computing makes it possible for smart factories to handle time-sensitive applications with 5G support. However, appropriate scheduling algorithms are required to ensure the efficient execution of tasks. In this paper, we proposed a hybrid edge-cloud collaborative computing scheduling framework and designed a DSOTS scheduling algorithm to serve a TSGS strategy; finally, the TSGS scheduling algorithm is implemented by performing greedy optimization on DSOTS. Through conducting performance evaluations, we found that the TSGS scheduling policy has optimized the execution time and cost of time-sensitive tasks in the smart factory to a great extent under the edge-cloud collaborative environment. User satisfaction is improved, and computation consumption is saved.

However, there are several limitations that will be our future work. First, we can specify more deadline rules to ensure efficient scheduling. When real-time tasks become more urgent, more efficient algorithms than greedy (using machine learning or other algorithms) are required to make faster decisions. At the same time, the network's asynchronous, random, and unpredictable characteristics and design fault-tolerant techniques for anomalies in task scheduling must be taken into account.

Author Contributions: Conceptualization, Y.Z.; methodology, Y.Z.; validation, Y.Z. and J.L.; formal analysis, Y.Z., B.T. and J.L; investigation, Y.Z.; data curation, Y.Z. and J.L.; writing—original draft preparation, Y.Z.; writing—review and editing, B.T. and J.Z.; project administration, J.Z.; and funding acquisition, B.T. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by National Key R&D Program of China (No. 2018YFB1402800), the National Natural Science Foundation of China (Nos. 61872138 and 61602169), the Natural Science Foundation of Hunan Province (No. 2021JJ30278), and the Postgraduate Scientific Research Innovation Project of Hunan Province (No. QL20210242).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data that support the findings of this study are available from the corresponding author upon reasonable request.

Acknowledgments: We would like to thank all the anonymous reviewers for their valuable comments and suggestions for improving the quality of the paper.

Conflicts of Interest: The authors declared that they have no conflict of interest to this work.

Abbreviations

The following abbreviations are used in this manuscript:

CPS	Cyber physical system
IoT	Internet of Things
DSOTS	Dynamic time-sensitive scheduling algorithm
TSGS	Dynamic time-sensitive scheduling algorithm with greedy strategy
CEC	Comprehensive execution capability
IIoT	Industrial Internet of Things
ECNs	Edge computing nodes
SDN	Software-defined network
SLA	Service-level agreement
MDs	Mobile devices

References

1. Weyer, S.; Meyer, T.; Ohmer, M.; Gorecky, D.; Zühlke, D. Future modeling and simulation of CPS-based factories: An example from the automotive industry. *IFAC-PapersOnline* **2016**, *49*, 97–102. [CrossRef]
2. Sodhro, A.H.; Pirbhulal, S.; de Albuquerque, V.H.C. Artificial Intelligence-Driven Mechanism for Edge Computing-Based Industrial Applications. *IEEE Trans. Ind. Inform.* **2019**, *15*, 4235–4243. [CrossRef]
3. Kobusinska, A.; Leung, C.K.; Hsu, C.; Raghavendra, S.; Chang, V. Emerging trends, issues and challenges in Internet of Things, Big Data and cloud computing. *Future Gener. Comput. Syst.* **2018**, *87*, 416–419. [CrossRef]
4. Akkus, I.E.; Chen, R.; Rimac, I.; Stein, M.; Satzke, K.; Beck, A.; Aditya, P.; Hilt, V. SAND: Towards High-Performance Serverless Computing. In Proceedings of the 2018 Usenix Annual Technical Conference (USENIX ATC 18), Boston, MA, USA, 11–13 July 2018; pp. 923–935.
5. Tang, B.; Fedak, G. WukaStore: Scalable, Configurable and Reliable Data Storage on Hybrid Volunteered Cloud and Desktop Systems. *IEEE Trans. Big Data* **2022**, *8*, 85–98. [CrossRef]
6. Abbas, N.; Zhang, Y.; Taherkordi, A.; Skeie, T. Mobile Edge Computing: A Survey. *IEEE Internet Things J.* **2018**, *5*, 450–465. [CrossRef]
7. Guo, F.; Tang, B.; Tang, M. Joint optimization of delay and cost for microservice composition in mobile edge computing. *World Wide Web* **2022**. [CrossRef]
8. Tang, B.; Kang, L. EICache: A learning-based intelligent caching strategy in mobile edge computing. *Peer-to-Peer Netw. Appl.* **2022**, *15*, 934–949. [CrossRef]
9. Tan, H. An efficient IoT group association and data sharing mechanism in edge computing paradigm. *Cyber Secur. Appl.* **2023**, *1*, 100003. [CrossRef]
10. Arkian, H.R.; Diyanat, A.; Pourkhalili, A. MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications. *J. Netw. Comput. Appl.* **2017**, *82*, 152–165. [CrossRef]
11. Pudjianto, D.; Ramsay, C.; Strbac, G. Virtual power plant and system integration of distributed energy resources. *IET Renew. Power Gener.* **2007**, *1*, 10–16. [CrossRef]
12. Ali, J.U.A.B.W.; Kazmi, S.A.A.; Altamimi, A.; Khan, Z.A.; Alrumayh, O.; Malik, M.M. Smart Energy Management in Virtual Power Plant Paradigm with a New Improved Multilevel Optimization Based Approach. *IEEE Access* **2022**, *10*, 50062–50077. [CrossRef]
13. Kumar, V.S.; Alagappan, A.; Andrews, L.J.B. Cybersecurity challenges in energy sector (virtual power plants)—Can edge computing principles be applied to enhance security? *Energy Inform.* **2021**, *4*, 5. [CrossRef]
14. Zhang, K.; Zhu, Y.; Leng, S.; He, Y.; Maharjan, S.; Zhang, Y. Deep Learning Empowered Task Offloading for Mobile Edge Computing in Urban Informatics. *IEEE Internet Things J.* **2019**, *6*, 7635–7647. [CrossRef]
15. Tang, L.; Tang, B.; Zhang, L.; Guo, F.; He, H. Joint optimization of network selection and task offloading for vehicular edge computing. *J. Cloud Comput.* **2021**, *10*, 23. [CrossRef]
16. You, Q.; Tang, B. Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things. *J. Cloud Comput.* **2021**, *10*, 41. [CrossRef]
17. Luo, S.; Chen, X.; Wu, Q.; Zhou, Z.; Yu, S. HFEL: Joint Edge Association and Resource Allocation for Cost-Efficient Hierarchical Federated Edge Learning. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 6535–6548. [CrossRef]
18. Wang, Y.; Ru, Z.; Wang, K.; Huang, P. Joint Deployment and Task Scheduling Optimization for Large-Scale Mobile Users in Multi-UAV-Enabled Mobile Edge Computing. *IEEE Trans. Cybern.* **2020**, *50*, 3984–3997. [CrossRef] [PubMed]
19. Pham, Q.; Fang, F.; Ha, V.N.; Piran, M.J.; Le, M.; Le, L.B.; Hwang, W.; Ding, Z. A Survey of Multi-Access Edge Computing in 5G and Beyond: Fundamentals, Technology Integration, and State-of-the-Art. *IEEE Access* **2020**, *8*, 116974–117017. [CrossRef]
20. Meng, J.; Tan, H.; Xu, C.; Cao, W.; Liu, L.; Li, B. Dedas: Online Task Dispatching and Scheduling with Bandwidth Constraint in Edge Computing. In Proceedings of the 2019 IEEE Conference on Computer Communications (INFOCOM 2019), Paris, France, 29 April–2 May 2019; IEEE: New York, NY, USA, 2019; pp. 2287–2295.
21. Wang, J.; Zhao, L.; Liu, J.; Kato, N. Smart Resource Allocation for Mobile Edge Computing: A Deep Reinforcement Learning Approach. *IEEE Trans. Emerg. Top. Comput.* **2021**, *9*, 1529–1541. [CrossRef]

22. Cao, B.; Zhang, J.; Liu, X.; Sun, Z.; Cao, W.; Nowak, R.; Lv, Z. Edge–cloud Resource Scheduling in Space-Air-Ground Integrated Networks for Internet of Vehicles. *IEEE Internet Things J.* **2021**, *9*, 5765–5772. [CrossRef]
23. Naha, R.K.; Garg, S.; Chan, A.; Battula, S.K. Deadline-based dynamic resource allocation and provisioning algorithms in Fog-Cloud environment. *Future Gener. Comput. Syst.* **2020**, *104*, 131–141. [CrossRef]
24. Xia, X.; Chen, F.; He, Q.; Grundy, J.C.; Abdelrazek, M.; Jin, H. Cost-Effective App Data Distribution in Edge Computing. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 31–44. [CrossRef]
25. Lv, Z.; Xiu, W. Interaction of edge–cloud Computing Based on SDN and NFV for Next Generation IoT. *IEEE Internet Things J.* **2020**, *7*, 5706–5712. [CrossRef]
26. Medhane, D.V.; Sangaiah, A.K.; Hossain, M.S.; Muhammad, G.; Wang, J. Blockchain-Enabled Distributed Security Framework for Next-Generation IoT: An Edge Cloud and Software-Defined Network-Integrated Approach. *IEEE Internet Things J.* **2020**, *7*, 6143–6149. [CrossRef]
27. Sun, J.; Yang, T.; Xu, Z. Assessing the implementation feasibility of intelligent production systems based on cloud computing, industrial internet of things and business social networks. *Kybernetes* **2022**, *51*, 2044–2064. [CrossRef]
28. Verma, A.; Goyal, A.; Kumara, S.; Kurfess, T.R. Edge–cloud computing performance benchmarking for IoT based machinery vibration monitoring. *Manuf. Lett.* **2021**, *27*, 39–41. [CrossRef]
29. Zhang, W.; Elgendy, I.A.; Hammad, M.; Iliyasu, A.M.; Du, X.; Guizani, M.; El-Latif, A.A.A. Secure and Optimized Load Balancing for Multitier IoT and edge–cloud Computing Systems. *IEEE Internet Things J.* **2021**, *8*, 8119–8132. [CrossRef]
30. Liu, L.; Li, H.; Gruteser, M. Edge Assisted Real-time Object Detection for Mobile Augmented Reality. In Proceedings of the 25th Annual International Conference on Mobile Computing and Networking (MobiCom 2019), Los Cabos, Mexico, 21–25 October 2019; Brewster, S.A., Fitzpatrick, G., Cox, A.L., Kostakos, V., Eds.; ACM: New York, NY, USA, 2019; pp. 1–16.
31. Li, X.; Wan, J.; Dai, H.; Imran, M.; Xia, M.; Celesti, A. A Hybrid Computing Solution and Resource Scheduling Strategy for Edge Computing in Smart Manufacturing. *IEEE Trans. Ind. Inform.* **2019**, *15*, 4225–4234. [CrossRef]
32. Amiri, M.M.; Gündüz, D. Machine Learning at the Wireless Edge: Distributed Stochastic Gradient Descent Over-the-Air. *IEEE Trans. Signal Process.* **2020**, *68*, 2155–2169. [CrossRef]
33. Wang, T.; Ke, H.; Zheng, X.; Wang, K.; Sangaiah, A.K.; Liu, A. Big Data Cleaning Based on Mobile Edge Computing in Industrial Sensor-Cloud. *IEEE Trans. Ind. Inform.* **2020**, *16*, 1321–1329. [CrossRef]
34. Wang, J.; Pan, J.; Esposito, F.; Calyam, P.; Yang, Z.; Mohapatra, P. Edge Cloud Offloading Algorithms: Issues, Methods, and Perspectives. *ACM Comput. Surv.* **2020**, *52*, 1–23. [CrossRef]
35. Chen, Y.; Zhang, N.; Zhang, Y.; Chen, X.; Wu, W.; Shen, X. Energy Efficient Dynamic Offloading in Mobile Edge Computing for Internet of Things. *IEEE Trans. Cloud Comput.* **2021**, *9*, 1050–1060. [CrossRef]
36. Lin, L.; Liao, X.; Jin, H.; Li, P. Computation Offloading Toward Edge Computing. *Proc. IEEE* **2019**, *107*, 1584–1607. [CrossRef]
37. Kaur, K.; Garg, S.; Aujla, G.S.; Kumar, N.; Rodrigues, J.J.P.C.; Guizani, M. Edge Computing in the Industrial Internet of Things Environment: Software-Defined-Networks-Based edge–cloud Interplay. *IEEE Commun. Mag.* **2018**, *56*, 44–51. [CrossRef]
38. Deng, S.; Xiang, Z.; Zhao, P.; Taheri, J.; Gao, H.; Yin, J.; Zomaya, A.Y. Dynamical Resource Allocation in Edge for Trustable Internet-of-Things Systems: A Reinforcement Learning Method. *IEEE Trans. Ind. Inform.* **2020**, *16*, 6103–6113. [CrossRef]
39. Gu, B.; Zhou, Z. Task Offloading in Vehicular Mobile Edge Computing: A Matching-Theoretic Framework. *IEEE Veh. Technol. Mag.* **2019**, *14*, 100–106. [CrossRef]
40. Kaur, K.; Garg, S.; Kaddoum, G.; Ahmed, S.H.; Atiquzzaman, M. KEIDS: Kubernetes-Based Energy and Interference Driven Scheduler for Industrial IoT in edge–cloud Ecosystem. *IEEE Internet Things J.* **2020**, *7*, 4228–4237. [CrossRef]
41. Puthal, D.; Obaidat, M.S.; Nanda, P.; Prasad, M.; Mohanty, S.P.; Zomaya, A.Y. Secure and Sustainable Load Balancing of Edge Data Centers in Fog Computing. *IEEE Commun. Mag.* **2018**, *56*, 60–65. [CrossRef]
42. Wang, X.; Wang, K.; Wu, S.; Di, S.; Jin, H.; Yang, K.; Ou, S. Dynamic Resource Scheduling in Mobile Edge Cloud with Cloud Radio Access Network. *IEEE Trans. Parallel Distrib. Syst.* **2018**, *29*, 2429–2445. [CrossRef]
43. Manaouil, K.; Lebre, A. Kubernetes and the Edge? Ph.D. Thesis, Inria Rennes-Bretagne Atlantique, Rennes, France, 2020.
44. Kannan, R.S.; Subramanian, L.; Raju, A.; Ahn, J.; Mars, J.; Tang, L. GrandSLAm: Guaranteeing SLAs for Jobs in Microservices Execution Frameworks. In Proceedings of the Fourteenth EuroSys Conference 2019, Dresden, Germany, 25–28 March 2019; Candea, G., van Renesse, R., Fetzer, C., Eds.; ACM: New York, NY, USA, 2019; pp. 1–16.
45. Zhang, Y.; Hua, W.; Zhou, Z.; Suh, G.E.; Delimitrou, C. Sinan: ML-based and QoS-aware resource management for cloud microservices. In Proceedings of the ASPLOS'21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual, 19–23 April 2021; Sherwood, T., Berger, E.D., Kozyrakis, C., Eds.; ACM: New York, NY, USA, 2021; pp. 167–181.
46. Calheiros, R.N.; Ranjan, R.; Rose, C.A.F.D.; Buyya, R. CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services. *arXiv* **2009**, arXiv:0903.2525.
47. Filho, M.C.S.; Oliveira, R.L.; Monteiro, C.C.; Inácio, P.R.M.; Freire, M.M. CloudSim Plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness. In Proceedings of the 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, Portugal, 8–12 May 2017; IEEE: New York, NY, USA, 2017; pp. 400–406.

48. Taneja, M.; Davy, A. Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm. In Proceedings of the 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, Portugal, 8–12 May 2017; IEEE: New York, NY, USA, 2017; pp. 1222–1228.
49. Kogias, M.; Mallon, S.; Bugnion, E. Lancet: A self-correcting Latency Measuring Tool. In Proceedings of the 2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, 10–12 July 2019; Malkhi, D., Tsafrir, D., Eds.; USENIX Association: Washington, DC, USA, 2019; pp. 881–896.