*electronics*

MDPI

*Article*

# Dynamic Application Partitioning and Task-Scheduling Secure Schemes for Biosensor Healthcare Workload in Mobile Edge Cloud

**Abdullah Lakhan [1], Jin Li [2], Tor Morten Groenli [1], Ali Hassan Sodhro [3,4], Nawaz Ali Zardari [5], Ali Shariq Imran [6], Orawit Thinnukool [7] and Pattaraporn Khuwuthyakorn [7,*]**

1 Mobile Technology Lab, Department of Technology, Kristiania University College, 0107 Oslo, Norway; abdullahrazalakhan@gmail.com (A.L.); tor-morten.gronli@kristiania.no (T.M.G.)
2 School of Computer Science, Beijing Jiaotong University, Beijing 100044, China; jinli71@gmail.com
3 Department of Computer Science, Kristianstad University, SE-291 88 Kristianstad, Sweden; ali.hassan_sodhro@hkr.se
4 Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518000, China
5 Faculty of Electrical Engineering, University Technology, Johor Bahru 81310, Malaysia; engrnawaz06@gmail.com
6 Department of Computer Science (IDI), Norwegian University of Science and Technology (NTNU), 2815 Gjøvik, Norway; ali.imran@ntnu.no
7 Research Group of Embedded Systems and Mobile Application in Health Science, College of Arts, Media and Technology, Chiang Mai University, Chiang Mai 50200, Thailand; orawit.t@cmu.ac.th
* Correspondence: Pattaraporn.khuwuth@cmu.ac.th



**Citation:** Lakhan, A.; Li, J.; Groenli, T.M.; Sodhro, A.H.; Zardari, N.A.; Imran, A.S.; Thinnukool, O.; Khuwuthyakorn, P. Dynamic Application Partitioning and Task-Scheduling Secure Schemes for Biosensor Healthcare Workload in Mobile Edge Cloud. *Electronics* **2021**, *10*, 2797. https://doi.org/10.3390/electronics10222797

Academic Editor: Khaled Elleithy

Received: 9 October 2021
Accepted: 10 November 2021
Published: 15 November 2021



**Abstract:** Currently, the use of biosensor-enabled mobile healthcare workflow applications in mobile edge-cloud-enabled systems is increasing progressively. These applications are heavyweight and divided between a thin client mobile device and a thick server edge cloud for execution. Application partitioning is a mechanism in which applications are divided based on resource and energy parameters. However, existing application-partitioning schemes widely ignore security aspects for healthcare applications. This study devises a dynamic application-partitioning workload task-scheduling-secure (DAPWTS) algorithm framework that consists of different schemes, such as min-cut algorithm, searching node, energy-enabled scheduling, failure scheduling, and security schemes. The goal is to minimize the energy consumption of nodes and divide the application between local nodes and edge nodes by applying the secure min-cut algorithm. Furthermore, the study devises the secure-min-cut algorithm, which aims to migrate data between nodes in a secure form during application partitioning in the system. After partitioning the applications, the node-search algorithm searches optimally to run applications under their deadlines. The energy and failure schemes maintain the energy consumption of the nodes and the failure of the system. Simulation results show that DAPWTS outperforms existing baseline approaches by 30% in terms of energy consumption, deadline, and failure of applications in the system.

**Keywords:** failure; dynamic application partitioning; task scheduling; offloading; energy consumption; MD5; MECCA

## 1. Introduction

Currently, the use of digital mobile applications in practice to deal with different life activities, e.g., digital shopping, digital booking, digital healthcare, etc., is growing. These applications have distributed runtime such as JAVA JVM for execution on different platforms, known as application partitioning. In recent years, emerging technologies such as edge computing and wireless networks for digital healthcare applications have been widely used for applications. Among these digital applications, mobile workflow applications are becoming increasingly popular [1,2]. The applications are healthcare ones, where

standalone mobile devices cannot run the applications locally. Mobile Edge-Cloud Computing Environment (MECCA) is a promising paradigm that allows resource-constrained devices to offload tasks to edge servers. Application partitioning is how a heavyweight application process divides local execution and external execution in the same mobile edge-cloud network. Offloading is the approach that divides the applications between the mobile device and accessible edge-cloud computing for execution [3]. In MECCA, many goals can optimize, for example, energy consumption, response time, latency, bandwidth, and resource use [4]. However, security is the biggest issue in application partitioning when it runs on different computing nodes and shares data between connected computing nodes for execution [4]. Many security schemes have been suggested in MECCA to support application data in the network [5]. For instance, RSA, CBC, SHA-256, and MD5 are based on both symmetric and asymmetric schemes. Many studies [5–10] have adopted these methods in MECCA for the application-partitioning problem in a distributed network. The primary goal of this study was to undertake encryption and decryption on the local machine to ensure the security of data before offloading to the edge-cloud network. Public and private keys are shareable in the network, and all connected devices can encrypt and decrypt data based on shared keys. However, existing studies have widely ignored local device energy and resource consumption during the implementation of security schemes at the local machine. Furthermore, existing studies have widely missed the deadline constraints of applications in the application-partitioning problem.

Dynamic application partitioning, which involves separating mobile application runtimes between local execution and remote-cloud execution to optimize total cost, is a vital component of the offloading system [5]. Edge-cloud data centers are a subset of remote-cloud data centers and have fewer computational capabilities, lower processing speed, and less memory capacity than remote-cloud data centers [6]. There are two runtimes for program execution in the offloading system; however, present research, which focuses on mobile energy savings, ignores cloud resource energy use. It is evident that cloud resources are critical to performance; however, if the offloading system does not effectively control resources, it may not be reliable enough to execute an application. After partitioning the program into local and edge-cloud execution, scheduling all jobs on the mobile device and edge-cloud resources is critical. Existing papers [7–9] have not included task-scheduling and security aspects in their suggested system, because variability in cloud resources, network connections, and mobile devices make static offloading with fixed bandwidth and resource values unreliable. Security inside the mobile edge-cloud network has been ignored in existing application-partitioning schemes in the system. This research looks at the mobile offloading system's deadline-constrained energy-aware dynamic application-partitioning and job-scheduling challenges in diverse contexts. When performing the offloading system, the goal is to reduce mobile and cloud resource energy consumption. The problem is divided into two sub-problems: first, we divide the application into two parts: local and edge-cloud execution. The application-partitioning decision takes into account network bandwidth, cloud resources, and job size. Then, all partitioned jobs are scheduled on local devices as well as the edge cloud. The aim of this study is to reduce the amount of power used by mobile devices and edge servers and still meet application deadlines. The failure of nodes is also considered in this study of healthcare applications. The study is going to solve many issues of existing application-partitioning methods. However, all existing application methods and frameworks have faced the following issues.

- Computation offloading plays a vital role in achieving optimal energy efficiency at the resource-constrained device, where computation-intensive workloads are offloaded to the server for execution. However, many parameters are considered during proposed application partitioning, such as bandwidth, execution time, and application deadline. Many existing offloading schemes during application partitioning consider pre-calculated parameters. However, many parameters can change during the runtime and require a dynamic and adaptive offloading scheme in the proposed architecture.

- Task scheduling is an NP-complete issue that involves scheduling tasks with heterogeneous resources based on a given constraint. It is never easy to schedule all jobs for complex resources so that the entire execution time stays within the constraints. Two types of failure could occur in a high-performance offloading system: intermittent network failure, and edge-cloud failure. Whenever a mobile user makes a dynamic decision to offload in a high-performance offloading system, the network connection or edge-cloud resource status becomes unstable during the offloading process. All rejected projects must be repaired within a set application deadline or completed locally if the available resources are sufficient. As mentioned earlier, a dynamic resubmission system, capable of correcting all rejected assignments while maintaining generosity, should be made available to address the problem.

The study makes the following contributions to the state of art to answer the aforementioned questions.

1. The task-scheduling problem in application partitioning has been widely solved based on HEFT (Heterogeneous Earliest Finish Time) approaches [5]. DAPWTS is the proposed algorithm scheme where many heuristics work together to achieve these goals. Initially, all applications are partitioned based on different metrics such as time, energy, and deadline based on the proposed min-cut algorithm. After partitioning, all tasks are prioritized based on their deadlines. All tasks are scheduled and maintain their failure status in the execution based on different schemes.
2. The DAPWTS algorithm framework, in which data generated by sensors and profiling technologies develop real-time contents of the wireless network due to the real-time system, is explored. We offer a new task-scheduling mechanism that maps all mobile healthcare applications depending on their requirements. The proposed work scheduling approach relies heavily on the application deadline division and sequencing criteria.
3. The study suggests an efficient failure-aware rescheduling approach that manages any failure with guaranteed performance in the proposed work.
4. The secure min-cut algorithm is a weighted graph of mobile workflow applications that divides the graph into local and remote executions to minimize the makespan and energy of edges. Before delivering their data to the system cloud nodes, tasks are encrypted and decrypted locally. The secure min-cut algorithm's primary purpose is to divide mobile workflow applications between local and remote-cloud execution depending on energy and time restrictions. The existing min-cut algorithms [7,9,11,12] focused solely on the security of jobs between their performance on various nodes, not on the safety of applications between local and cloud execution.
5. The study devises the two phase-enabled security methods, ensuring data transaction security and storage security in the distributed mobile edge-cloud network.

The following is a breakdown of the manuscript structure. Current application-partitioning strategies and their execution in mobile edge-cloud networks are discussed in Section 2. The system description, problem formulation, and mathematical aspects of the problem are demonstrated in Section 3. Section 4 depicts each stage of the proposed DAPWTS algorithm framework, where each heuristic plays a role in solving the problem. Section 5 describes the study's performance evaluation, which compares all baseline schemes and suggested work against the target function. The study's findings and future efforts are presented in Section 6.

## 2. Related Work

The number of biosensor-enabled digital healthcare applications is growing progressively. Generally, these applications are accessible on client devices such as mobile and Internet of Things (IoT) healthcare sensors. However, due to their resource constraints, the application-partitioning mechanism has gained significant attention for healthcare applications. We have analyzed closely related efforts of application-partitioning schemes.

### 2.1. Task Partitioning Offloading Schemes

In [1], the study devised a static application-partitioning scheme for healthcare applications on mobile and cloud nodes. Execution and task division is done during the design of the applications. An optimal offloading system could be helpful in complex workflow applications. There are three types of offloading scheme considered in mobile edge-cloud computing: non-offloading, full offloading, and partial offloading. Many efforts have been made with the offloading plan to achieve multiple objectives such as energy consumption and response time for interactive applications while minimizing the total cost. Regarding the entire offloading scheme, ref. [2] proposed a framework for application partitioned-based dynamic profiling and static analysis, in which an application is partitioned into a thread-level fine-grained application virtual machine and offloaded to the remote-cloud clone for execution. Dynamic adaption and migration was not considered in this paper. Based on current network status, the energy model is proposed in [6]. This aims to predict the offloading accuracy of either application workload offload, or not to minimize energy consumption. Network bandwidth-related parameters are taken into consideration in the proposed work. The mobile cloud runs a time-based framework in the dynamic offloading decision in which parts of the application are offloaded to the cloud or parts are locally executed (see [7]). MAUI [8] proposed a fine-grained application-partitioning framework. The goal was to reduce programmer efforts during application partitioning. Using hybrid analysis (i.e., static and dynamic), runtime offloading could improve energy consumption more effectively than full coarse-grained offloading. Furthermore, workload scheduling in the mobile and cloud energy-based strategy was proposed in [9]. The central theme of this strategy balances the workload between mobile and cloud resources to minimize energy consumption.

### 2.2. Task Allocation Schemes in Application Partitioning

Regarding diversity and the optimal finding of cloud resource for task allocation, a rule-based scheduling hyper-heuristic scheduling strategy was proposed in [10]. It effectively and dynamically searches for optimal solutions for resource allocation among all candidate solutions, and this process continues until the final optimal solution is produced. An energy-efficient workflow and independent task scheduling based on cloud speed and power consumption-based frameworks is proposed in [11,13–16]. The main goal of these studies is to minimize energy consumption and idle time while performing task scheduling.

### 2.3. Security Schemes in Application Partitioning

Security-enabled application partitioning was investigated by these studies [12,17–20]. These studies considered geographically distributed cloud data centers and applied workload migration techniques to minimize the energy of running tasks inside the system. Dynamic application partitioning with failure and resource constraint-enabled schemes were suggested in these studies [21–27]. The studies considered fine-grained and coarse-grained healthcare workloads in their models. The mobile and edge cloud was considered in the system, and the objective was to minimize the energy consumption of mobile devices and minimize the response time of applications in the network. To the best of our knowledge, dynamic application partitioning and task scheduling of biosensor healthcare workflow in secure mobile edge clouds has not been studied yet. Closely related studies [1,3,5,9,21–27] have focused on dynamic application partitioning and minimizing the energy use of mobile devices. These studies only considered bandwidth and resource constraints. However, security, deadline, and failure constraints have been widely ignored in the studies for the application-partitioning problem of healthcare applications. These studies [21–27] considered healthcare application security, deadline, and failure constraints in a mobile edge-cloud network. The objective is to minimize the energy consumption of all computing nodes in the mobile edge-cloud architecture. The study suggested energy-efficient offloading and secure task-scheduling schemes in the system for healthcare applications.

## 3. Problem Description

This study devises Dynamic Application Partitioning Healthcare Workload Task-Scheduling Secure (DAPWTS) schemes, as shown in Figure 1. The architecture consists of biomedical sensors, workflow application interfaces, and resource layers. Many biomedical sensors are connected to the application interface. Each sensor generates data and runs inside application workflows at the user level. Heavyweight data requires a lot of resources to run applications. However, due to resource constraints, mobile devices running locally cannot satisfy the requirements. Therefore, this study devises a DAPWTS algorithm framework consisting of different schemes, such as secure min-cut algorithm, searching node, energy, and failure-scheduling schemes. The goal is to minimize energy consumption of nodes and divide the application between local nodes and edge nodes by applying the min-cut algorithm. After partitioning the applications, the node-search algorithm optimally searches to run applications under their deadlines. The energy and failure schemes maintain the energy consumption of the nodes and the failure of the system. After execution, all the task data will be stored based on a dynamic message-digest algorithm (MD5). The security validation of data in workflow applications is necessary. All tasks of one application are run on different nodes and share their data without backtracking in the system.

The study denotes the notations of the problem in Table 1.

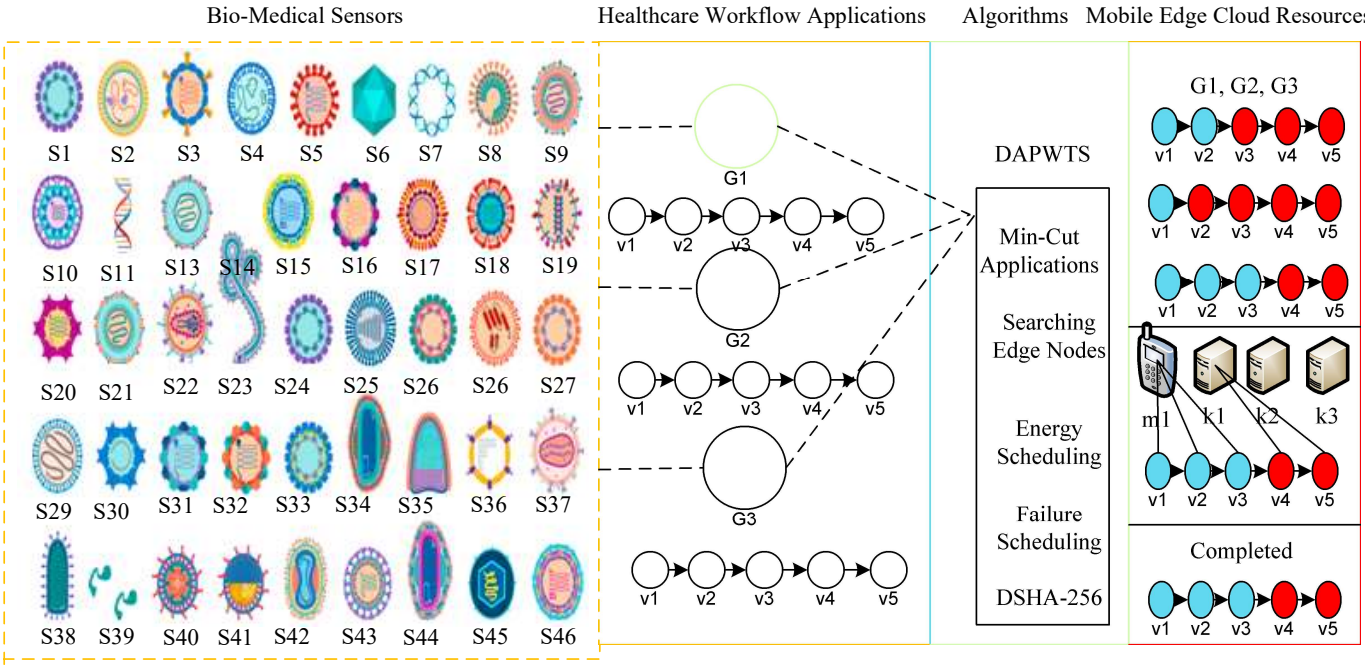



**Figure 1.** Biosensor-enabled mobile edge-cloud architecture: DAPWTS.

### *3.1. System Model and Problem Formulation*

The proposed mobile edge-cloud computing architecture is a combination of the wireless network, mobile edge server, and cloud server. Mathematical notations are marked in Table 1. In the proposed architecture, a mobile healthcare application is modeled as a consumption-weighted Directed Acyclic Graph (DAG) i.e., G (V, E). However, each task $v_i$ is represented by a node $v_i \in V$. An edge $e(v_i, v_j) \in E$ represents communication between $v_i$ to $v_j$. A task $v_i$ could be started when associated all predecessors complete [20]. $v_1$ and $v_n$ are two dummy tasks (i.e., entry task and exit task). A task could be started when associated predecessors complete. In short, $v_j$ cannot be started in anticipation of $v_i$ accomplishing the job and $i < j$. A set of servers has denoted by, e.g., $K = \{k_1, \ldots, k_n\}$. We presuppose that each $k$ server holds a different virtual machine type, that all virtual machine instances are heterogeneous, and that every virtual machine (VM) has dissimilar computation speed which is illustrated as $\zeta_j = (j = 1, \ldots, M)$. A set of virtual machine instances can be

shown by $VK = \{vk_1, \ldots, vk_n\}$, in which $K_i^{vk}$ is the virtual machine assignment for task $v_i$. Each workflow application task has workload $W_i = \{i = 1, \ldots, N\}$ with deadline $D_G$. To minimize the power consumption of the submitted workflow tasks, we assign each application task to the lower-speed VM while meeting the deadline $D_G$, because the lower-speed VMs always leads to lower power consumption. Since a task $v_i$ can only be performed by one VM $j$, a decision variable $x_{ij} \in \{0, 1\}$ is used, $x_{ij}$=1 only if the task $v_i$ is assigned to the VM $V_j$. The task $v_i$ has two execution costs (i.e., local and cloud), on cloud execution time is determined by the speed $\zeta_j$ and power consumption $P_w$, e.g., $T_i^e$, i.e., $T_i^e = \sum_{j=1}^{\mathcal{V}} x_{ij} \times \frac{W_i}{\zeta_j} \times P_{cw}$ and task execution time on mobile $M_i^e = \sum_{m=1}^{\leftarrow} x_{ij} \times \frac{W_i}{\zeta_m} \times P_{mw}$.

**Table 1.** Mathematical notation.

| Notation | Description |
|---|---|
| $N$ | Number of tasks $v$ |
| DAG | Directed Acyclic Graph of Application |
| $\alpha_1$ | Reschedule the rejected tasks on mobile |
| $\alpha_2$ | Reschedule the rejected tasks on edge cloud |
| $\mathcal{V}_M$ | Number of virtual machines $\mathcal{V}$ |
| $\mathcal{V}_j$ | *jth* virtual machine in edge server |
| $v_i$ | Workflow application task |
| $loc$ | Local set of tasks |
| $c$ | Edge-Cloud tasks |
| $W_i$ | Weight of the each task |
| $\lambda_i = v_i \in V$ | All tasks arrival rate |
| $\lambda$ | Arrival rate at cloudlet virtual machine |
| $\lambda_0$ | Arrival rate at mobile device |
| $\mu_r$ | Current Cloudlet speedup factor rate |
| $\mu_m$ | The mobile service rate |
| $\mu_c, \mu_w$ | Cellular and WLAN bandwidth rate |
| $\Omega$,$\lambda$ | Application partitioning factor during offloading |
| $\zeta_{\mu_j}$ | Speed rate of *jth* virtual machine |
| $\zeta_{\mu_m}$ | Speed rate of mobile processor $v_i$ |
| $T_i^e$ | Execution cost of task $v_i$ on edge cloud $k$ |
| $M_i^e$ | Execution cost of task $v_i$ on mobile |
| $P_{cw}$ | Power consumption rate at edge-cloud virtual machine |
| $P_{mw}$ | Power consumption rate at mobile device |
| $x_{ij}$ | Assignment of task $v_i$ on virtual machine $j$ |
| $B_U$ | Upload bandwidth |
| $B_D$ | Download bandwidth |
| $B_i$ | Begin time of the task $v_i$ |
| $F_i$ | Finish time of the task $v_i$ |
| $G(V, E)$ | Call graph of application $G$ |
| $D_G$ | Deadline of the application $G$ |
| $AV$ | Most Tightly Connected Vertices |
| $a$ | Arbitrary of vertex $G$ |
| $wt$ | weight of each task on graph |
| $s, t$ | source and sink in call graph |
| $R_i$ | Recover time of a task during failure |
| $FR_i$ | Failed ratio of a task $v_i$ |

### *3.2. Application Energy Consumption*

The total power consumption of a workflow application is an amalgamation of computation time and communication time. Since computation cost could include location and remote execution after application partitioning, the communication cost is determined by the weight of data transport and available network bandwidth. The average power consumption of workflow application due to offloading is expressed as follows in Equation (1).

$$EG_{total} = \sum_{v \in V} Pr_v \times EG_v^{loc} + \sum_{v \in V} (1 - Pr_v) \times EG_v^c + \sum_{e(v_i, v_j) \in E} Pr_e \times T_e^{trans}. \quad (1)$$

Equation (1) describes the total power consumption of the workflow application during application partitioning and scheduling. The mathematical notations are denoted in Table 1. The mobile workflow healthcare application is modeled as Directed Acyclic Graph (DAG), i.e., G (V, E). However, each task $v_i$ is represented by a node $v_i \in V$. An edge $e(v_i, v_z) \in E$ represents the communication between $v_i$ to $v_z$. A task $v_i$ could be started when associated all predecessors are complete [12]. Furthermore, the $v_1$ and $v_n$ are two dummy tasks (i.e., entry task and exit task). A task could be started when associated predecessors are complete. In short, $v_z$ cannot be started in anticipation of $v_i$ accomplishing the job and $i < z$. A set of servers can be represented by $K = \{k_1, \ldots, k_n\}$. We presuppose that each $k$ edge cloud holds a different virtual machine type such that all virtual machine instances are heterogeneous, and that every virtual machine (VM) has dissimilar computation speed illustrated as $\zeta_j = (j = 1, \ldots, M)$. A set of virtual machine instances can be shown by $VK = \{vk_1, \ldots, vk_n\}$, in which $K_i^{vk}$ is the virtual machine assignment for task $v_i$. Each workflow application task has workload $W_i = \{i = 1, \ldots, N\}$ with deadline $D_i$. To minimize the power consumption of the submitted workflow tasks, we assign each application task to the lower-speed VM while meeting the deadline $D_i$, because the lower-speed VMs always lead to lower power consumption [16]. Since a task $v_i$ can only be performed by one VM $j$, a decision variable $x_{ij} \in \{0, 1\}$ is used, $x_{ij} = 1$ only if the task $v_i$ is assigned to the VM $V_j$. The task $v_i$ has two execution costs (i.e., local and cloud); cloud execution time is determined by the speed $\zeta_j$ and power consumption $P_{cw}$ e.g., $T_i^e$, i.e., $T_i^e = \sum_{j=1}^{\mathcal{V}} x_{ij} \times \frac{W_i}{\zeta_j} \times P_{cw}$ and task execution time on mobile determined in the following way, e.g., $M_i^e = \sum_{m=1}^{\leftarrow} x_{ij} \times \frac{W_i}{\zeta_m} \times P_{mw}$. In the same way, the vector $Y = \{y_{ij} : v \in V, j \in M\}$ with variable $x_{ij}$ indicates either task offload or not, namely determined in Equation (2)

$$x_{ij} = \begin{cases} 1, & \text{if } v_i \in V_{loc} \\ 2, & \text{if } v_i \in V_c \end{cases} \quad (2)$$

According to Equation (3), the communication cost is determined by $E_{cut} = 1$, otherwise the same location tasks have no communication cost.

$$Pr_e = \begin{cases} 1, & \text{if } e \in E_{cut} \\ 0, & \text{if } e \notin E_{cut} \end{cases} \quad (3)$$

The study formulated the application partitioning and task-scheduling energy consumption of the problem in Equation (4).

$$\min Z = \sum_{G=1}^{A} \sum_{i \in G}^{N} \sum_{m=1}^{\psi} \sum_{j=1}^{M} x_{ij} \lambda_i \times \zeta_{\mu_j} \times P_{cw} \times T_i^e + T_i^{trans} + x_{ij} \lambda_i \times \zeta_{\mu_m} \times M_i^e \times P_{mw} \times EG_{total}. \quad (4)$$

Equation (5) determines the energy consumption of mobile devices.

$$\sum_{v \in V} Pr_v \times EG_v^{loc} = M_i^e = \sum_{i=1}^{N} x_{ij} \lambda_i \times \frac{W_i}{\zeta_{\mu_m}}, \quad (5)$$

Equation (6) determines the energy consumption of edge-cloud nodes.

$$\sum_{v \in V} (1 - Pr_v) \times EG_v^c = T_i^e = \sum_{i=1}^{N} x_{ij} \lambda_i \times \frac{W_i}{\zeta_{\mu_j}}, \quad (6)$$

Equation (7) determines the communication energy between nodes.

$$\sum_{e(v_i,v_j)\in E} Pr_e \times T_e^{trans} = w(e(v_i,v_j)) = \frac{in_{ij}}{B_U} + \frac{out_{ji}}{B_D}, \tag{7}$$

Equation (8) shows that the initial energy consumption of the task becomes zero.

$$T_{j,0} = 0, \tag{8}$$

The begin time of the task at the busy machine is determined in Equation (9), if machine has already executed any tasks, and the next task waits until it finishes its first execution.

$$B_i = T_{i,j} = T_{i-1,j} + \sum_{i=1}^{N} x_{i,j}\lambda_i T_i^e, \tag{9}$$

The real execution time of tasks is determined in Equation (10).

$$T_i^e = \sum_{i=1}^{V_c} x_{ij}\lambda_i \times \frac{W_i}{\zeta_{\mu_j}}, M_i^e = \sum_{i=1}^{V_{loc}} x_{ij}\lambda_i \times \frac{W_i}{\zeta_{\mu_m}}, \tag{10}$$

The actual finish time of tasks is determined in Equation (11).

$$F_i = \sum_{j=1}^{M} T_{i,j}x_{i,j}, \tag{11}$$

Failure and secure time is determined in Equation (12).

$$\sum_{i=1}^{N} x_{i,j} = 1, \sum_{j=1}^{M} x_{i,j}\lambda = 1, \sum_{m=1}^{\psi} x_{i,j}\lambda_0 = 1, \tag{12}$$

All tasks must be finished under their deadlines as determined in Equation (13).

$$\sum_{G=1}^{A}\sum_{i=1}^{N} F_i \leq D_G, \tag{13}$$

Each task and each computing node can assign and schedule one task at a time, as determined in Equation (14).

$$x_{i,j}\{0,1\}. \tag{14}$$

## 4. Proposed Algorithm DAPWTS Framework

To solve the joint application partitioning and task-scheduling problem, we propose the Dynamic Application-Partitioning Workload Task-Scheduling Secure (DAPWTS) framework, consisting of different phases, such as the secure min-cut, task-sequencing phase, task-scheduling phase, and failure-aware scheduling phase. We run the mobile workflow applications onto heterogeneous resources via different phases to achieve robust and seamless execution. Algorithm 1 shows the entire process of the application executions. It takes all mobile workflow applications as input and processes them via different phases. All particular stages will explain in their subsections.

**Algorithm 1:** DAPWTS framework.

```
Input  : G ∈ A, V ∈ G ;
Output : min_Z;
begin
    Z ←0;
    Application-Partitioning Phase;
    Task-Sequencing Phase;
    Power-Efficient VM Searching Phase;
    foreach (v_i ∈ V ∈ G) do
        foreach (V_j ∈ Q_vm) do
            Task-Scheduling Phase;
            Z* ← v_i ← j;
            Z ← Z*;
        end
        Task-Failure Scheduling Phase;
    end
    return Z;
end
```

### *4.1. Secure Min-Cut Algorithm*

The secure min-cut algorithm is a weighted graph of mobile workflow applications that divides the graph into local and remote executions. The goal is to minimize the makespan and energy of edges. Before delivering their data to the system cloud nodes, tasks are encrypted and decrypted locally. The secure min-cut algorithm's primary purpose is to divide mobile workflow applications between local and remote-cloud execution depending on energy and time restrictions. The existing min-cut algorithms [7,9,11,12] focus solely on the security of jobs between their performance on various nodes, not on the safety of applications between local and cloud execution as shown in Figure 2. The primary goal of the secure min-cut phase is to partition the application into a set of local tasks and offloaded jobs with the security schemes. Network contents, e.g., bandwidth, signal, inference, secure, CPU resources, and execution time parameters, are considered for the application partitioning. Algorithm 2 divides workflow applications into local $V_{loc}$ and $V_c$ sets with an optimal cut in order of task size, available bandwidth, resources, and speed. A secure min-cut function always returns an optimal solution in a dynamic and adaptive environment and applies security based on the task size and available node resources. With the help of profiling techniques such as network, mobile-device status, and program status, the available edge-cloud speed is calculated dynamically before and after partition. Based on Equation (1), the workflow application is divided into two disjoint sets of the task, i.e., local and cloud tasks. However, Non-offloadable jobs are randomly generated $\lambda_0$ and mapped locally on mobile computing, processing $\mu_m$ using the dispatcher $\alpha_1$. Cloud tasks $\lambda_V$ are offloaded by the dispatcher $\alpha_2$ to cloudlet computing via wireless communication. In the dynamic offloading system, two kinds of connection are employed—WLAN and cellular—with rates $\mu_c$ and $\mu_w$, respectively. The variable $\eta$ is an impeding factor list of all available links for communication, and $\xi$ is the differentiating factor. If one connection receives an external signal or bandwidth, it changes the status of the network. However, $\Omega.\lambda$ shows either tasks successfully executed or those which fail to be assigned to any resource.

In Algorithm 2, step 2 shows that $\min_{cut}$ in the given graph can be more than one until $\infty$. Application partition into local $V_{loc}$ and $V_c$ execution is shown by steps 3–5. All offloaded tasks $V_c$ and non-offloaded tasks $V_{loc}$ nodes are merging into a single node in steps 5–6. Steps 6–8 show the $\min_{cut}$-phase function in a given application graph. Steps 9–11 elaborates a minimum $\min_{cut}$ among all given cuts. Finally, step 12 returns the minimum $\min_{cut}$ group list from all given graphs. The suggested dynamic message digest of an up to 256-byte security scheme encrypts all data at the local device with a public

key (PK). At the same time, the nodes decrypt data on a private key (PV) provided by the system in the network. All the nodes can encrypt and decrypt data with both public and private keys in the system.

**Algorithm 2:** Min-Cut Algorithm.

**Input** : $(G \in A, w)$;
**begin**
  $w_t(\min_{cut}) \Leftarrow \infty$;
  **for** $(w = 1$,*length(source-vertices))* **do**
    Partition the application according to Equation (1);
    Disjoint sets $V_c$ and $V_{loc}$;
    $(G, w_t)$= merging$(G, w_t)$, source-vertices, source-vertices(m))Merging Function;
    **while** $\mid v \in V \mid > 1$ **do**
      $[C_{ut}(AV - t, t), s, t]$ = $\min_{cut}$-Phase$(G, w_t)\min_{cut}$ Phase Function;
      **if** $w_t(c_{ut}(AV - t, t)) < w_t(\min_{cut})$ **then**
      **end**
      $\min_{cut} \Leftarrow c_{ut}(AV - t, t)$;
    **end**
    merging$(G, a, w_t, s, t)$;
    Call DMD5-256 methods to secure data;
    $G \leftarrow DMD5 \leftarrow PKandPV$;
  **end**
  **return** $\min_{cut}$, $\min_{cut}$-Grouping-List;
**end**

Figure 2 shows the procedure of a secure min-cut algorithm for a healthcare application. The study assumes that the application consists of different workflow tasks, e.g., $v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9$. All tasks are taking data from different biosensors such as $s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9$. The secure min-cut algorithm divides the applications based on available resources, encryption and decryption time, and task deadline, where 50/100 denotes edge-cloud execution time and mobile execution time. The blue nodes are those tasks being executed on mobile devices, and red tasks are being executed on the edge-cloud nodes. Each task at the node encrypts data with a message-digest algorithm (MD5) and decrypts data with the same algorithm. The data size is measured in kilobytes (KB) between nodes during the execution of applications. The study modified the existing MD5 [16] with dynamic encryption and decryption schemes. It can handle failure of tasks due to resources or service unavailability in the system.

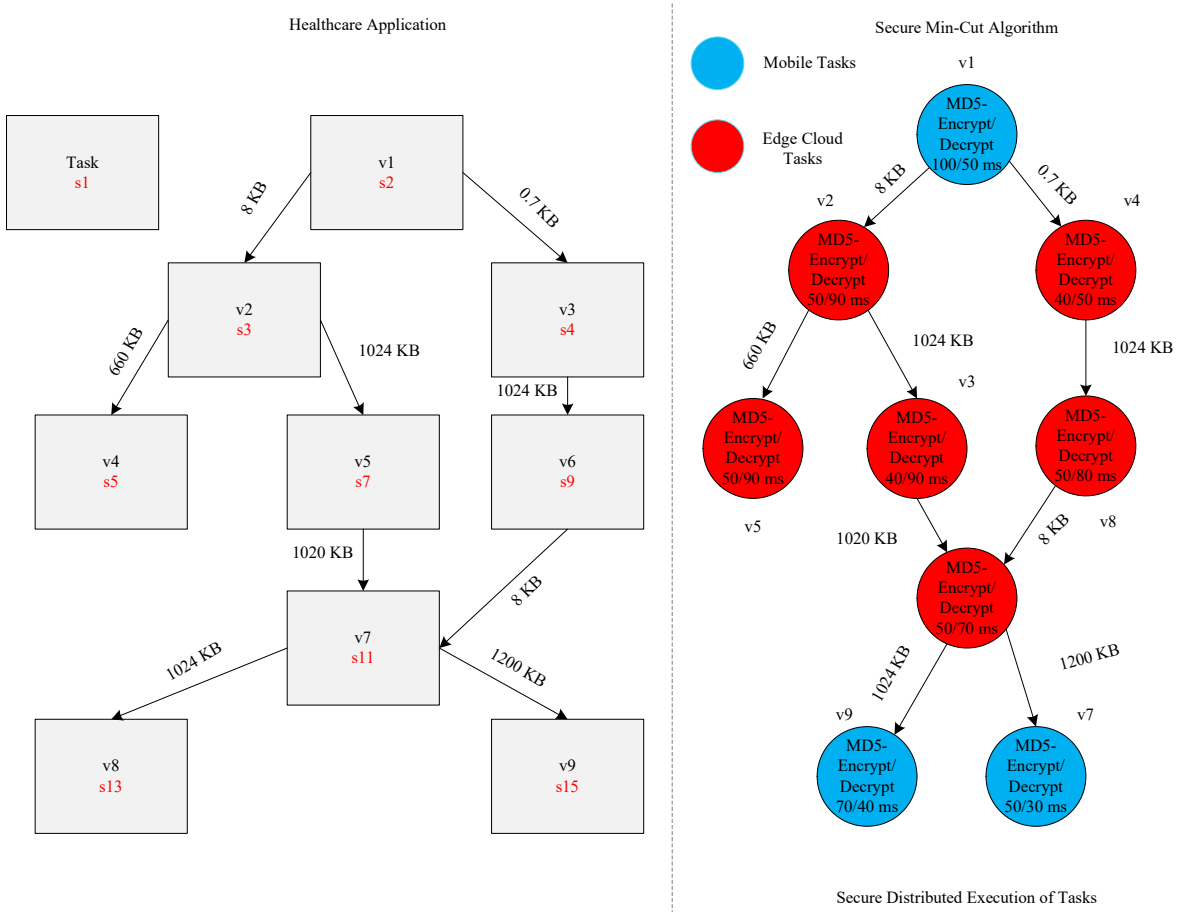



**Figure 2.** Secure min-cut algorithm.

### *4.2. Task Sequencing*

The deadline of a workflow application $G$ can be met if all its tasks are completed earlier than their deadline requirements. We introduce advisable deadlines for all tasks, which poise the time for each task based on their workload magnitudes. We obtain the deadline as follows.

$$ratio = \sum_{G=1}^{A} \frac{G_D}{Z}, \tag{15}$$

Equation (15) determines the total execution of all tasks of all applications by dividing their deadline by execution time.

$$\sum_{i=1}^{V} T_i^{e'} = T_i^e \times ratio, \tag{16}$$

The deadline of each task is assumed to be the finish time of the task based on Equation (11).

$$M_i^{e'} = M_i^e \times ratio, \tag{17}$$

Equation (17) determines the deadline for local and cloud execution based on Equation (11).

$$T_e^{trans'} = T_e^{trans} \times ratio, \tag{18}$$

The transmission time also considers additional time inside the deadline as determined in Equation (18).

$$vc_{di} = \min(\{v_{dj}\}) - T_i^{e'}(v_j) - T_e^{trans'}(ij) \quad \forall v_i \in V_c \exists v_j \in successor(v_i). \tag{19}$$

Equation (19) determines that the deadline must satisfy all predecessors and successors in all tasks.

$$vm_{di} = \min(\{v_{dj}\}) - M_i^{e'}(v_j) \quad \forall v_i \in V_{loc} \exists v_j \in successor(v_i). \tag{20}$$

Equation (20) determines that each task is to be executed within the given deadline on the particular node. The mobile workflow applications should be sequenced based on given partitioning sets [1] in the considered problem. A dynamic offloading system submitted the workflow tasks $V_{loc}$, $V_c$ to the scheduling system at the same time, and each application workflow is bounded by deadline $vc_{di}$. All workflow applications are submitted randomly to the cloud server and mobile device with priority. All tasks must complete their performance before their given deadlines. We consider the slack time only for offloaded tasks (i.e., offloadable tasks) because local tasks are lightweight and require the local mobile device for execution. The slack time $T_i^{slack}$ of the offloaded task $v_i$ is determined by $T_i^{slack} = F_i - vc_{di}$. The finish time $F_i$ of the task $v_i$ on a mobile device and the virtual machine is determined by $T_i^e$ and $M_i^e$, respectively. The $T_i^{slack}$ is mathematically determined in Equations (21) and (22).

$$T_i^{slack} = vc_{di} - F_i \tag{21}$$

$$\overline{F}_i = \sum_{i=1}^{V_c} x_{ij} \times \frac{W_i}{\zeta_{\mu_j}} + \sum_{i=1}^{V_{loc}} x_{ij} \times \frac{W_i}{\zeta_{\mu_m}}, \tag{22}$$

We rank all tasks onto a virtual machine based on the HEFT [28] heuristic, and added weight to all nodes and edges according to their priorities. All data offloaded to the system follow a first-come-first-served policy and are sequenced by all proposed methods as shown in Figure 3. We define a set of sequence rules as follows.

- Shortest deadline first (SDF): We sort the set of workflow applications based on their deadline. The smaller deadline application is sorted first and the bigger one later. If the deadlines are the same, then FCFS policy will be applied.
- Shortest slack time first (SSTF): The application tasks are sorted according to the task slack time (TST). The task which has the shortest slack time is scheduled first.
- Shortest weight first (SWF): The applications are sequenced based on the weight of all tasks, the shorter weight application arranged first and the bigger one later.

We introduce task-sequencing rules based on Equations (21) and (22) that define the priority of all tasks from task entry $v_i$ to exit $v_n$ by considering all predecessors and successors of the given application. Initially constructed task sequences using different methods are shown in Figure 3.

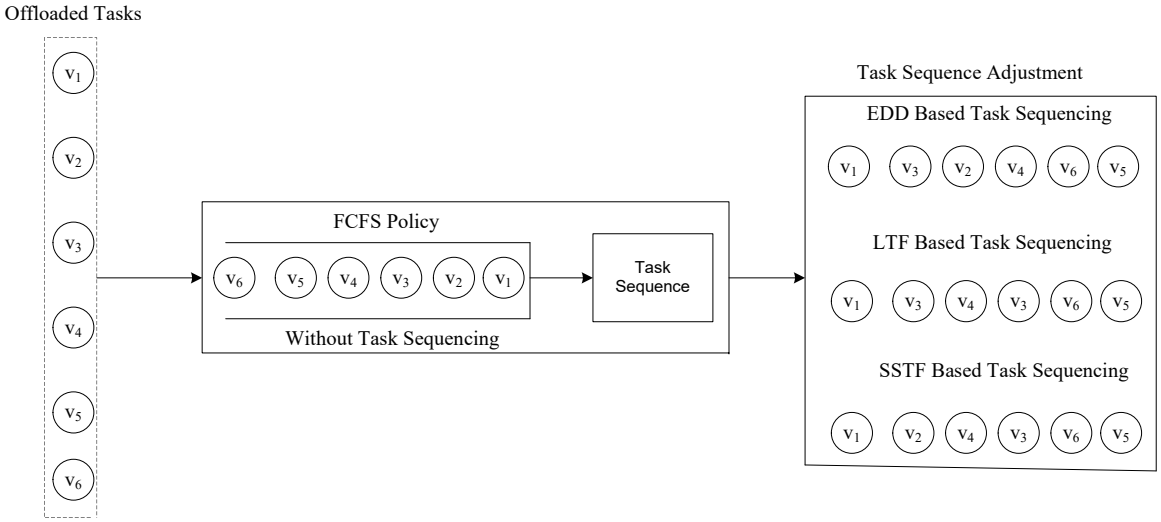



**Figure 3.** Task-Sequencing Rules.

### *4.3. Optimal Power-Efficient Edge Node Searching*

We are searching for a virtual machine based on its power efficiency. We have sorted all the virtual machines according to speed in descending order. The lower-speed virtual device consumes less power as compared to the higher-speed virtual machine. The proposed Algorithm 3 searches all lower-powered devices, then checks the finish of all tasks so that they are completed within a given deadline. This process will check until an appropriate machine is found that meets the requirements.

$$\zeta^*_{\mu_j} = \frac{W_i}{\zeta_{\mu_j}}. \tag{23}$$

**Algorithm 3:** Optimal edge-cloud node searching.

**Input:** ($v_i \in V$) to Scheduling;
**begin**
  $Q_{vm} \leftarrow$ Sort all Edge Nodes by their power consumption $\zeta^*_{\mu_j}$ based on Equation (23) with descending order;
  $\mathcal{V} \leftarrow$Null;
  **foreach** $\mathcal{V}_j \in Q_{vm}$ **do**
    $T_{j,0} \leftarrow 0$;
  **end**
  **foreach** $\mathcal{V}_j \in Q_{vm}$ **do**
    Calculate $T^e_i$ of $\mathcal{V}_j$ based on Equations (2)–(5);
    **if** $T_{j,i-1} + T^e_i \leq vc_{di}$ **then**
      Calculate the $C_{j,i}$ of $\mathcal{V}_j$ by Equation (4);
      $\mathcal{V} \leftarrow \mathcal{V}_j$;
      break;
    **end**
    Calculate the average Energy Consumption $Z_i$ of workflow application based on Equation (1);
  **end**
  **return** $Z_i, \mathcal{V}$
**end**

In Algorithm 3, in step 2, virtual machines were sorted by lower-powered speed. Step 3 initializes the value of any VM as zero. Steps 4–6 calculate each task execution on

each virtual machine. Steps 8–10 says that each assigned VM task must be completed before the application deadline with lower power consumption. The average power consumption of the complete task is calculated by step 12.

### *4.4. Energy-Efficient Task Scheduling*

After the initial task scheduling in Algorithm 3, guaranteed deadlines constrain mapping. Still, we can swap tasks within the same slots of machines and mobile cores in Algorithm 4. In this way, we can minimize the total energy consumption of all resources. Lines 2–6 in Algorithm 4 determine the swapping offloaded tasks based on their timeslot of virtual machine (e.g., virtual machine 1 to virtual machine 2) based on their power consumption. Lines 7–11 reschedule all applications inside mobile cores in a way that mobile energy is minimized. We exploited the Dynamic Voltage Frequency Scaling (DVFS) method [29] to reduce the consumption of power resources during rescheduling. Therefore, Algorithm 4 reschedules energy-efficient assignments of all these tasks without violating their deadline among resources.

**Algorithm 4:** Energy-efficient task scheduling.

**Input** : $\{G \in A, V \in G, v_i \in V_c, v_i \in V_{loc}\}$ ;
**begin**
  **foreach** $(v_i \in V_c \in V)$ **do**
    **if** $(v_i \leftarrow \mathcal{V}_2 \leq vc_{di})$ **then**
      Apply DVFS method;
      Swap higher power $v_i \leftarrow \mathcal{V}_1$ to lower power $\mathcal{V}_2$;
      Assign $v_i \leftarrow \mathcal{V}_1$;
    **end**
  **end**
  **foreach** $v_i \in V_{loc}$ **do**
    **if** $(T_i^M \leq vc_{di})$ **then**
      Apply DVFS method;
      Swap higher power mobile $P_{mw}$ to lower power core $P_{mw}$;
      Assign $v_i$ lower power core in the mobile device;
    **end**
  **end**
**end**

### *4.5. Failure-Aware Scenario*

This paper considers dynamic mobile cloud application partitioning and offloading, where network contents and resource values regularly fluctuate. In dynamic environments, the failure of applications due to network failure and resources often occurs. Therefore, how to handle the loss of tasks in a mobile cloud environment is a challenging question. As we know, checkpointing [18] is efficient in a task-failure situation, which recovers the from the point of failure regardless of resubmission. We exploit the checkpoint method to deal with any fault in our proposed scheme. We assume the scenario where a mobile user invoking a mobile workflow application during mobility requires two kinds of ubiquitous services such as network service and cloud service simultaneously to run the application tasks. The user may face application failure due to transient loss when requesting the cloud server and retrieving its result, as shown in Figure 4. Therefore, the proposed scheme offers seamless and robust services and executes all applications without any degrading performance.

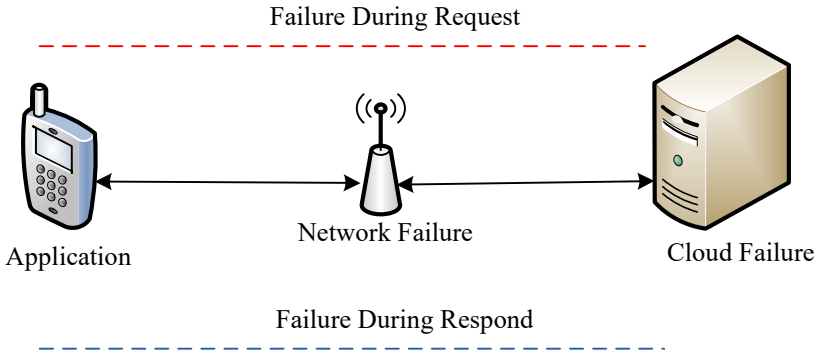



**Figure 4.** Task-failure scenario.

### *4.6. Failure Task Scheduling*

Figure 5 illustrates the failure-aware situation in the dynamic mobile cloud environment. A user can move from one place to another and invoke mobility-conscious network and cloud services concurrently. Presently, the cellular network is omnipresent, but it has low speed compared to a WiFi network. Notwithstanding, a cellular network allows users to invoke cloud services during trips. However, the failure of tasks often occurs during mobility and when invoking services for execution. We proposed a failure-aware task-scheduling algorithm to handle any failure during offloading and scheduling to cope with this situation.

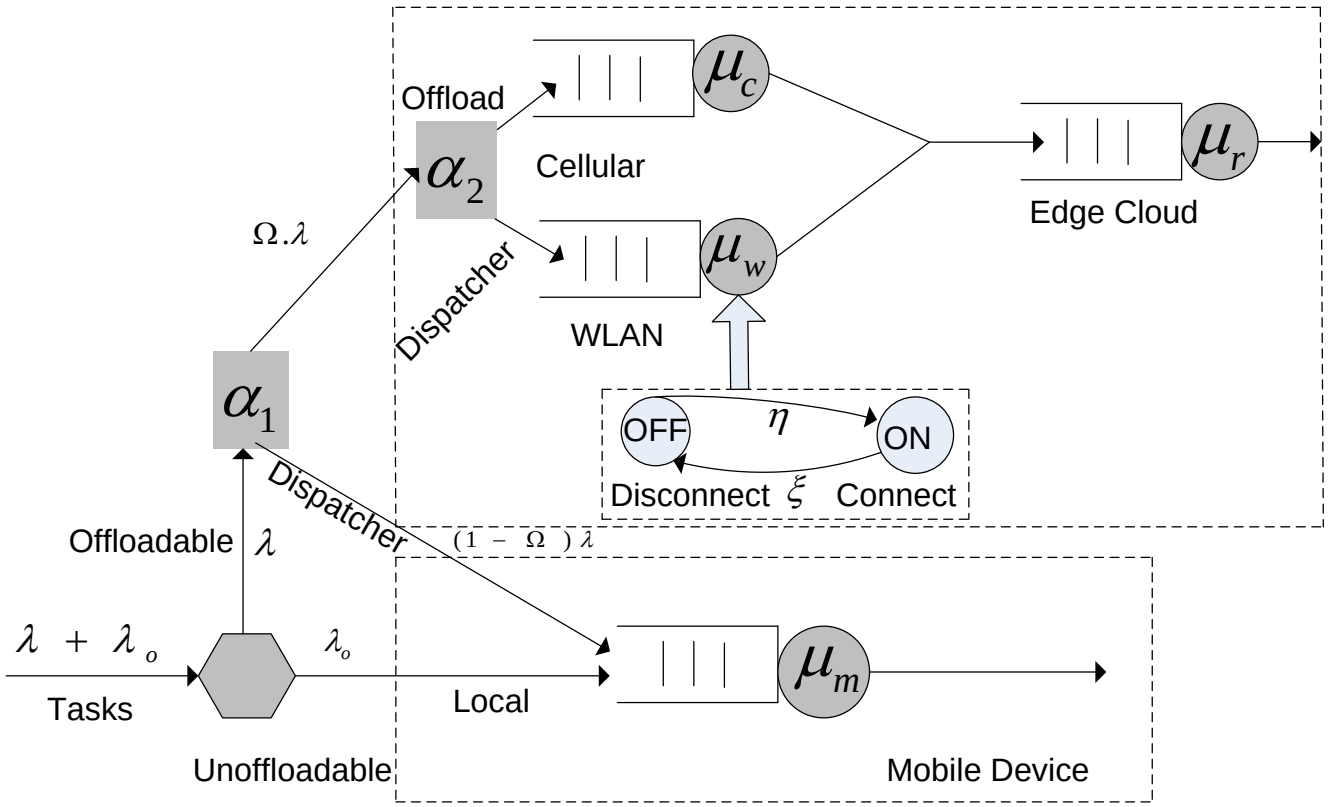



**Figure 5.** Failure-aware environment in the proposed work.

We explain all the steps of the proposed Algorithm 5 as follows.

**Algorithm 5:** Failure-Aware Scheduling.

**Input:** $(\alpha_1, \alpha_2, \lambda, \lambda_0, \mu_m, \mu_w, \mu_c, \mu_r, 1-\Omega, \eta, \xi, R_i, FR_{vi})$;
**begin**
  $Qm \leftarrow$ Mobile tasks queue;
  $Qw \leftarrow$ wireless tasks queue;
  $Qc \leftarrow$ cellular tasks queue;
  $Qr \leftarrow$ cloud tasks queue;
  **foreach** $(\lambda_i \leftarrow Qm)$ **do**
    **if** $(T_i^e + T_i R_i \leq vc_{di})$ **then**
      Apply Checkpoint Method $FR_{vi}$;
      Call Dispatcher $\alpha_1$ to handle $\lambda \leftarrow$ Unoffloaded tasks $v_i$;
    **end**
  **end**
  **foreach** $(\lambda_0 \leftarrow Qw)$ **do**
    **if** $(T_i^e + T_e^{trans} + R_i \leq vc_{di} \quad \& \eta = 1)$ **then**
      set $\xi$ = 1 connect;
      Apply Checkpoint Method $FR_{vi}$;
      Call Dispatcher $\alpha_2$ to handle $(1-\Omega)\lambda_0 \leftarrow$ offloaded tasks $v_i$;
    **end**
    **else**
      $\xi$ = 0 disconnect;
      $\eta$ = 1;
    **end**
  **end**
  **foreach** $(\lambda_0 \leftarrow Qc)$ **do**
    **if** $(T_i^e + T_e^{trans} + R_i \leq vc_{di} \quad \& \eta = 1)$ **then**
      set $\xi$=1 connect;
      Apply Checkpoint Method $FR_{vi}$;
      Call Dispatcher $\alpha_2$ to handle $(1-\Omega)\lambda_0 \leftarrow$ offloaded tasks $v_i$;
    **end**
    **else**
      $\xi$ = 0 disconnect;
      $\eta$ = 1;
    **end**
  **end**
  **foreach** $(\lambda_0 \leftarrow Qr)$ **do**
    **if** $(T_i^e + T_e^{trans} + R_i \leq vc_{di} \quad \& \eta = 1)$ **then**
      set $\xi$ = 1 connect;
      Apply Checkpoint Method $FR_{vi}$;
      Call Dispatcher $\alpha_2$ to handle $(1-\Omega)\lambda_0 \leftarrow$ offloaded tasks $v_i$;
    **end**
    **else**
      $\xi$ = 0 disconnect;
      $\eta$ = 1;
    **end**
    End All conditions;
  **end**
  End Loop;
**end**
End Main;

- We manage the failure of tasks into the four queues, i.e., $\{\mu_m, \mu_w, \mu_c, \mu_r\}$. $\mu_m$ denotes the mobile queue of failure tasks, $\mu_w$ illustrates the failure queue at the wireless network. $\mu_c$ and $\mu_r$ denote failure queues of cellular and cloud computing. $\lambda$ denotes the failure of tasks at the mobile device, and $\lambda_0$ shows the failure of offloaded tasks.

- Lines 6–19 show unoffloaded tasks that failed at the mobile device, and we apply the checkpointing technique to recover the task from the point of failure. If the recovery time and current execution are still less than the deadline, the dispatcher reads from the queue and tries to recover it. This process will continue until failed tasks recover successfully.
- Lines 10–14 show offloaded tasks that failed at the cellular network, and we apply the checkpointing technique to recover the task from the point of failure. If the recovery time, current execution, and communication time are still less than the deadline, the dispatcher reads from the queue and tries to recover it. This process will continue until failed tasks recover successfully. Lines 15–17 show the user changes its connections between cellular and wireless during mobility, and network status varies between on and off.
- Lines 18–22 show offloaded tasks that failed at the wireless network, and we apply the checkpointing technique to recover the task from the point of failure. If the recovery time, current execution, and communication time are still less than the deadline, the dispatcher reads from the queue and tries to recover it. This process will continue until failed tasks recover successfully. Lines 23–25 show that the user changes its cellular and wireless connections during mobility, and network status varies between on and off.
- Lines 26–30 show offloaded tasks that failed at the cloud resource, and we apply the checkpointing technique to recover the task from the point of failure. If the recovery time, current execution, and communication time are still less than the deadline, the dispatcher reads from the queue and tries to recover it. This process will continue until failed tasks recover successfully. Lines 31–36 show that the user changes its cellular and wireless connections during mobility, and network status varies between on and off. When all tasks are retrieved from the failed queue, the condition, loop, and main loop will be terminated.

### *4.7. Time Complexity*

The running time of DAPWTS is divided into three different parts—application partitioning, task sequence, and task scheduling. For application partitioning, we exploit $O(V \times E)$ time complexity. $V$ is the number of call-graph nodes, and $E$ is the edges among dependent tasks. Our partitioning algorithm follows the same rules as existing min-cut algorithms. However, we only consider application partitioning in the call-graph applications. We exploit $O(nlogm)$ for the task-sequencing component. $n$ is the number of tasks of call graph $G$, and $m$ denotes the number of sorted sequences for all tasks is equal to $O(N \times N^3) = n^4$. The time complexity of the task-scheduling algorithm is equal to $O(VlogM)$, as $V$ is the number of iterations for searching tasks, and $M$ is the number round for allocating all tasks onto the computing resources. The DAPWTS algorithm is iterative. Previously proposed application-partitioning frameworks [11,13–15] have $n^6$ and $n^7$ time complexities to reduce the energy and response time of applications. The proposed work has the $n^5$ time complexity mentioned above while solving the offloading and scheduling problem of mobile workflow applications. Still, $n^5$ time complexity of the proposed framework is high due to security components, partitioning components, task sequences, and failure-aware scheduling components. However, $n^4$ can reduce if we exploit blockchain-enabled frameworks and distributed symmetric keys in the network. The main limitation of the current system is that it cannot support blockchain technology with the current runtime in the system and cannot reduce the time complexity of the system for mobile workflow applications.

## 5. Performance Evaluation Simulation Setup

To evaluate the efficiency and effectiveness of the DAPWTS framework, we compare it with existing frameworks in terms of power consumption and applications quality-of-service requirements. We conduct an experiment to run mobile workflow applications

based on the simulation parameters given in Table 2. To verify the effectiveness of all methods, we exploited the ANOVA technique [16] in the experiment environment. We show the performance of the DAPWTS framework via different metrics into different subsections. Table 2 shows all simulation parameters in the experimental configuration file. The JAVA simulation environment for both local and edge computing is designed in the JAVA language using the java runtime machine. The study considered 2000 heterogeneous mobile devices with different configurations to install and run the mobile workflow healthcare applications. Different healthcare sensors $S_1, S_2, S_3, S_4, S_5$ are connected to the mobile devices to generate data for the mobile workflow applications. Communication technologies are fixed in the simulation environment, such as WiFi, with upload and download bandwidth determined in mbps (e.g., 200 to 300). The study fixed the edge cloud with the different configurations as shown in Table 2. Therefore, the resources are heterogeneous and have different computational speeds and power consumption in the study. Table simulation parameters which are exploited in the simulation configuration files as shown in the following Table 2.

**Table 2.** Simulation parameters.

| Simulation Parameters | Values | Symbol |
|---|---|---|
| Languages | JAVA | JVM |
| Applications | Healthcare | $G$ |
| $\lambda_i$ | Arrival of tasks | 5 s |
| Simulation Time | 6 h | - |
| Experiment Repetition | 14 | - |
| No. of Mobile devices | 2000 | - |
| WAN-WLAN Network Bandwidth | 20 to 300 mbps | $\alpha_1$ |
| WAN-Propagation Delay | 50 to 150 mbps | $\alpha_2$ |
| Upload/download data size of a task | 2000/150 KB | $W_i$ |
| Possibility offload to edge cloud | 80% | $v^c$ |
| Possibility offload to loc | 12% | $v^{loc}$ |
| VM processing speed cloudlet | 1200/22,000 MIPS | $\mu_r$ |
| No. VMS per cloudlet | $3/\infty$ | $\mathcal{V}_M$ |
| Mobile-Device Capability. | 64 GB | $\mu_m$ |
| VMs speed | 500–2500 MIPS | $\mathcal{V}_j$ |
| Mobile Devices | 10–4 GB RAM, 64 ROOM | - |
| Edge-Cloud-1 | Core i3 | - |
| Edge-Cloud-2 | Core i5 | - |
| Edge-Cloud-3 | Core i7 | - |
| Edge-Cloud-4 | Core i9 | - |
| Edge-Cloud-5 | Core GPU | - |
| $S1$ | Heartbeat Sensor | Arduino |
| $S2$ | SPO2 Sensor | Arduino |
| $S3$ | airflow Sensor | Arduino |
| $S4$ | body temperature Sensor | Arduino |
| $S5$ | ECG Sensor | Arduino |
| $S6$ | Glucometer Sensor | Aurdino |
| $S7$ | galvanic skin Sensor | Aurdino |
| $S8$ | EMG Sensor | Aurdino |
| $S9$ | EEG Sensor | Aurdino |
| $S10$ | MQ2 Sensor | Aurdino |
| $S11$ | Lead-1 Sensor | Aurdino |
| $S12$ | Lead-2 Sensor | Aurdino |

### *5.1. Baseline Approaches*

We deployed and executed mobile healthcare applications as the detail of workload analysis depicted in Table 3. We will evaluate the efficiency and effectiveness of the proposed architecture based on the components mentioned earlier by comparing it

with existing application-partitioning schemes with the execution time, security, deadline, and failure constraints.

1. Baseline1: This is a default min-cut scheme-enabled application partitioning and computational offloading method that allows running the entire workload locally or fully offloaded to the edge cloud. These kinds of schemes exploit many existing studies [7–9]. To evaluate the effectiveness and efficiency of DAPWTS, we implemented these environments for testing purposes and with our proposed system.
2. Baseline2 [19,20]: This is the computational offloading scheme that allows the entire clone of the application to be submitted to the cloud system for execution. Numerous studies [11,13,16] have exploited this scheme to run different mobile workflow applications in their frameworks. It is a widely exploited offloading scheme that is adopted by many studies in the mobile cloud system. It enables the mobile workflow application to run onto distributed computing resources to obtain energy-consumption objectives efficiently. We implemented similar offloading in our DAPWTS framework to simultaneously maintain the trade-off between mobile energy and cloud energy.

**Table 3.** Mobile Workflow Workload Analysis.

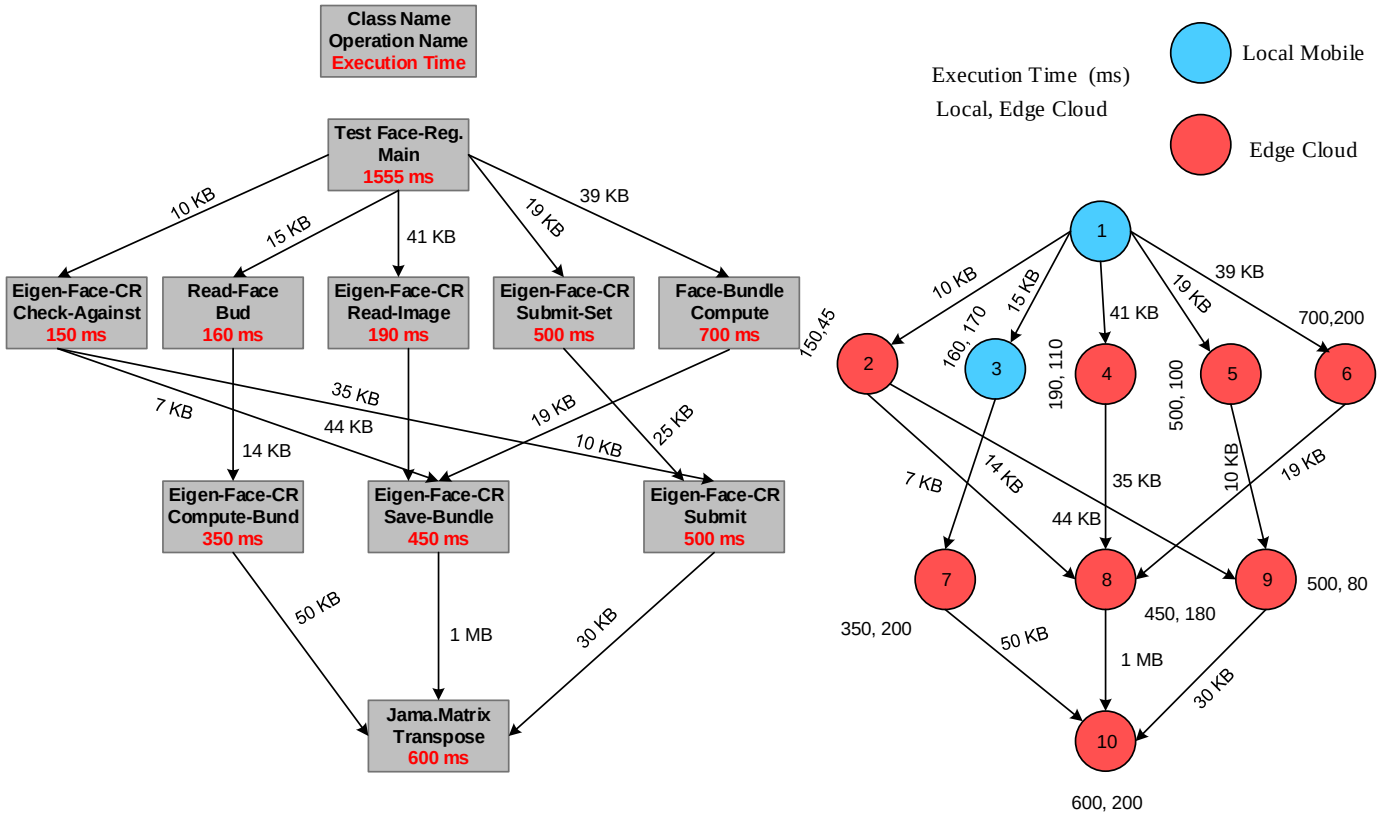
| Workload | Data Size (GB) | C.Ins. (MI) | No. of Tasks |
|---|---|---|---|
| G1 | 5 | 5.8 | 1000 |
| G2 | 3 | 6.8 | 900 |
| G3 | 4 | 7.8 | 800 |

### *5.2. Mobile Workflow Application-Partitioning System*

The workload of applications is shown in Table 3. The data size determined GB and then divided into kilobytes. The source code of the aforementioned mobile workflow applications is available at the following links: http://darnok.org/programming/face-recognition/ (accessed on 1 January 2016), https://github.com/mHealthTechnologies/mHealthDroid (accessed on 23 March 2020). We denote each workflow with four columns: workload name, size, required CPU instructions (CIns) to run it, and the number of tasks.

We partitioned the mobile workflow application into local sets of tasks (e.g., non-offloaded tasks) and edge-cloud tasks (e.g., offloaded tasks) as shown in Figure 6 in the following way.

After applying the min-cut algorithm on the workflow call graph, we produce two blues nodes (i.e., to be scheduled at the mobile device) and red notes (i.e., needs to be offloaded to the cloud for execution). Figure 6a shows that the system takes a call graph (e.g., mobile workflow application) as an input, and Figure 6b shows after applying the secure min-cut algorithm that we generated the partitioning of tasks with their node and edge weights. The partitioning is done on the mobile device; based on different criteria parameters, we made this partitioning decision: these parameters, task size, execution time, deadline, availability of resources, and network bandwidth.




(a) Face-Recog. Workflow Call Graph of Healthcare Application

(b) Weighted Graph of Dynamic Application Partitioning

**Figure 6.** Application Partitioning Results After Applying Algorithm 1.

### *5.3. Profiling Technologies*

The mobile workflow application is represented as a call graph, as we can scale applications via an open-source connection scrutiny tool which shows energy profiling for uploading and downloading the data. This is also available at www.speedtest.net/ (accessed on 1 May 2021), and for energy profiling you can this tool [11]: http://ziyang.eecs.umich.edu/projects/powertutor/, (accessed on 9 October 2011).

Dynamic Application-Partitioning Setup

The healthcare workflow application was partitioned into local and remote execution. Each task $v_i$ is represented by a node, and each node has exactly two costs (i.e., local execution cost and cloud execution cost). We partitioned the application under $F = 2$ speedup factor and available bandwidth of 1 M/B, respectively. However, each mobile workflow application is bound by its deadline. The deadline of each mobile workflow $D_w$ is expressed as follows:

To evaluate the performance of the proposed DAPWTS algorithm next to the healthcare heuristics-based DEA benchmark [19,20] that determines the effectiveness of the proposed algorithm. The calibration parameters of tasks are the same as a healthcare workflow; the performance evaluation of the DAPWTS is measured at different deadlines. The DAPWTS has RPD (relative percentage division), which is used to compare with existing schemes such as non-offloading, and is described as follows:

$$RPD\% = \frac{Z - Z^*}{Z} \times 100\%, \tag{24}$$

In Equation (24) $Z$ is the objective function of study, i.e., $EG_{loc}$, and $Z^*$ is the optimal objective function of total cost, e.g., $EG_{total} = \sum_{v \in V} Pr_v EG_v^{loc} + \sum_{v \in V}(1 - Pr_v) EG_v^{cl} + \sum_{e(v_i,v_j) \in E} Pr_e T_e^{trans}$.

### *5.4. Components Calibration of DAPWTS Framework*

The DAPWTS framework consists of the following components: Min-Cut partitioning method, task sequence rules, searching power-efficient virtual machines, task scheduling, and failure-aware task scheduling. We simulated all the results with 100% of energy and time. The *y*-axis represented energy-consumption level, i.e., 20 to 100%. The *x*-axis represented the number of tasks.

#### 5.4.1. Task-Sequencing Rules

The $\tau \in \{0.2, 0.4, 0.6, 0.8, 1\}$ variable is exploited as tightly valued to run workflow applications $G_w \in A$. Figure 6 shows that $G_w$ 0.8 is best among all adjustments and with the proposed task sequence rules. Based on the ANOVA technique, Figure 6 describes the mean plot of $\tau$ using 95.0% Tukey HSD intervals. It can be observed that RPD% reduces when $\tau$ increases from 0.2 to 0.4. Existing heuristics such as Baseline1 and Baseline2 were compared with proposed DAPWTS-based algorithm components. We evaluated the overall performance and validity based on the above components, and all RPD results proposed that the algorithm is better when compared to all benchmark heuristics bound by deadline constraints [28,30,31].

Table 4 determines the node specification concerning the energy consumption in Watt and speed of the simulation config file.

**Table 4.** Mobile Edge-Cloud Resource Specifications.

| Mobile Device | Edge-Cloud-1 | Edge-Cloud-2 | Edge-Cloud-3 | Edge-Cloud-4 | Edge-Cloud-5 |
|---|---|---|---|---|---|
| Core | 1 | 1 | 1 | 1 | 1 |
| MIPS/Core | 200 | 400 | 600 | 800 | 1000 |
| Power/Core | 50 W | 100 W | 150 W | 200 W | 250 W |

#### 5.4.2. Secure Min-Cut Algorithm

This part discusses healthcare application dynamic and secure partitioning before scheduling them onto the mobile devices and edge cloud nodes. In the initial phase, all tasks are divided between local execution and edge-cloud execution based on deadlines, execution time, and resource availability of the nodes. After partitioning, each node performs encryption and decryption based on dynamic message digestion (MD5) methods. As we already have shown in the application-partitioning process in Figure 7, the secure dynamic application partitioning may vary according to network bandwidth and resource availability. To run the mobile workflow applications, all users demand ubiquitous applications, wireless network services, and omnipresent cloud services. Therefore, existing application-partitioning schemes such as Baseline1 [1,4,7,11,12] and Baseline2 [21–27] cannot adopt any dynamic changes at the runtime and do not ensure any security on the data.

According to Figure 7, the proposed algorithm RPD% is preferred to existing heuristic techniques. The main reason for this is that DAPWTS (e.g., secure min-cut) adopts any environment changes (i.e., resources that speedup factor and network bandwidth) and improves application-partitioning performance iteratively in the system. The main goal is to reduce the trade-off energy consumption of both mobile and cloud resources simultaneously.

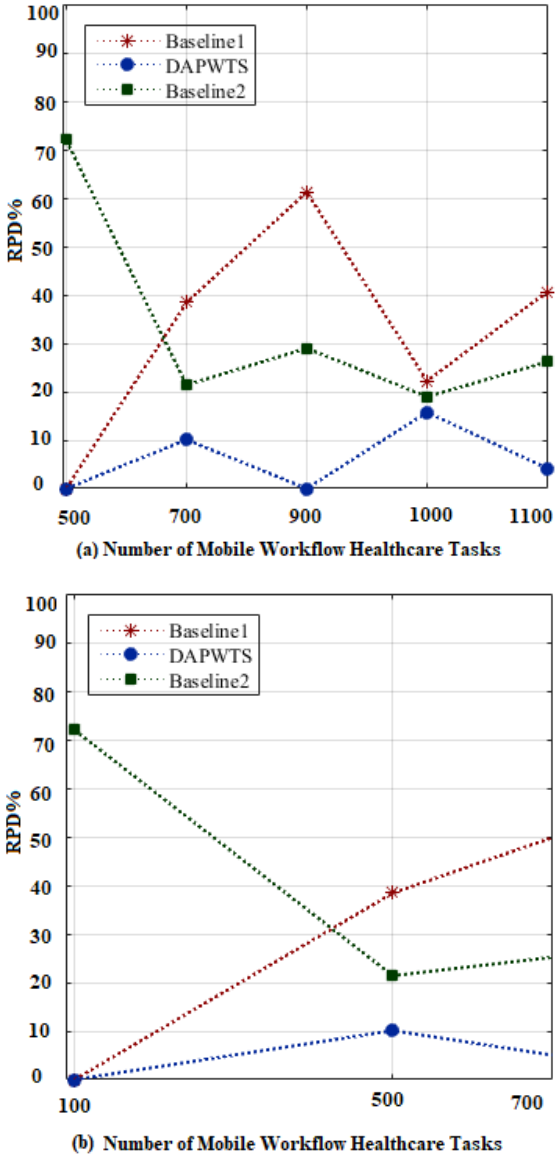



**Figure 7.** Application partitioning performance and task sequencing.

#### 5.4.3. Task-Scheduling Phase

We assume the mobile devices are resource-constrained; however, edge-cloud servers have unlimited on-demand resources. Our primary goal is to satisfy the quality-of-service requirements (e.g., deadline and failure management) of all applications and minimize the trade-off energy consumption of mobile devices and edge-cloud servers simultaneously when running to all forms in the proposed DAPWTS system. We compare the effectiveness of the DAPWTS (task scheduling) with existing frameworks regarding deadlines of the different requests and energy consumption of mobile and edge-cloud servers. We run the random workflow tasks of various applications and note DAPWTS satisfies all task deadlines during partitioning and scheduling. Hence, Figure 8a,b illustrates that DAPWTS outperforms existing studies in terms of user QoS requirements and improves the use of RPD% of used resources. DAPWTS transcends existing studies because Baseline1 and Baseline2 only focus on energy minimization of the mobile device; however, this does not focus on the QoS requirements of different application tasks during partitioning and scheduling.

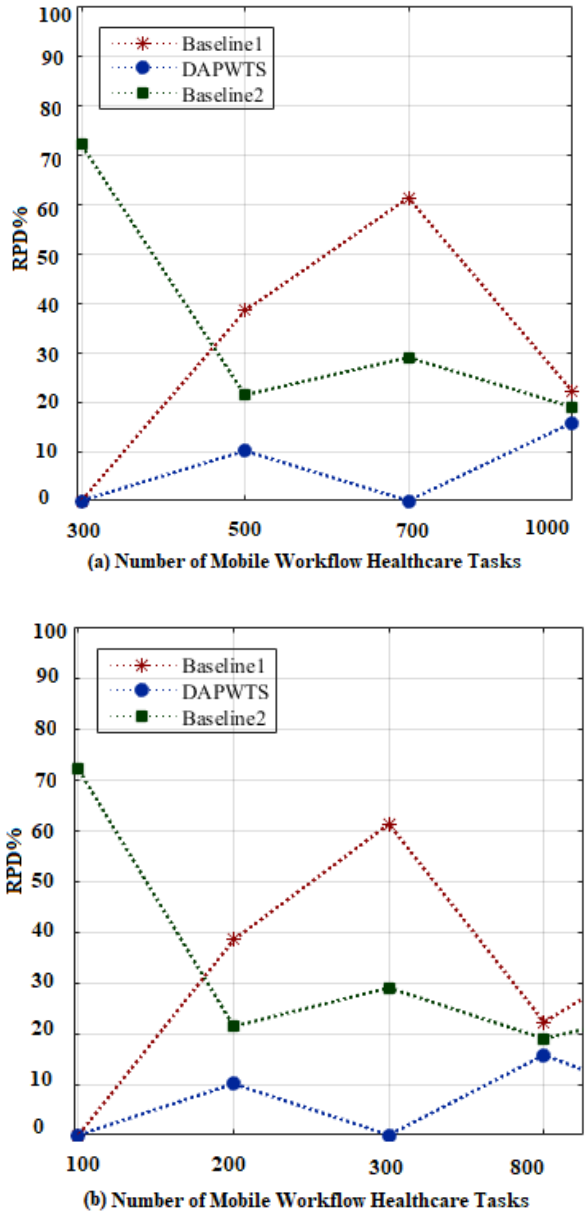



**Figure 8.** Workflow application no. of tasks completed within a given deadline.

Furthermore, Figure 8a,b prove that DAPWTS also outperforms existing studies when all workflow applications are simultaneously in the system. When we run all applications with baselines in a mobile cloud system, we note that these methods only improve the mobile energy consumption regardless of edge-cloud servers and quality-of-service requirements of applications. Hence, the proposed task scheduling and energy-efficient scheduling initially outperform during application partitioning and scheduling of different workflow applications.

5.4.4. Energy-Efficient Task Scheduling

In this discussion, the study evaluated the performance of existing application-partitioning schemes and proposed schemes in terms of energy consumption with different deadlines, security, and failure constraints. Application-partitioning architecture and procedures focus mainly on minimizing energy consumption of resource-constrained mobile devices for healthcare applications. Application-partitioning methods as energy-hungry mobile application tasks need to be offloaded to cloud computing to reduce device energy and improve user experience. However, the main limitation of existing studies is that they

do not balance energy consumption between mobile devices and cloud machines when mobile workflows run onto the resources. It is unfair to the cloud resource to schedule energy-hungry offloaded tasks without energy minimization planning. The proposed algorithm DAPWTS fairly schedules all tasks between the mobile device and cloud resources and adjusts the total energy consumption of workflow applications.

Figures 9a,b and 10a,b show that initial task scheduling in the DAPWTS framework improves energy consumption of all distributed resources, including mobile device and edge-cloud servers, concurrently, as compared to the existing application-partitioning schemes. It can be seen that energy-efficient task scheduling further improves the energy consumption of shared resources via the DVFS technique during rescheduling. Therefore, DAPWTS is an efficient and effective framework that enhances the performance of different applications and minimizes the overall energy consumption of distributed resources.

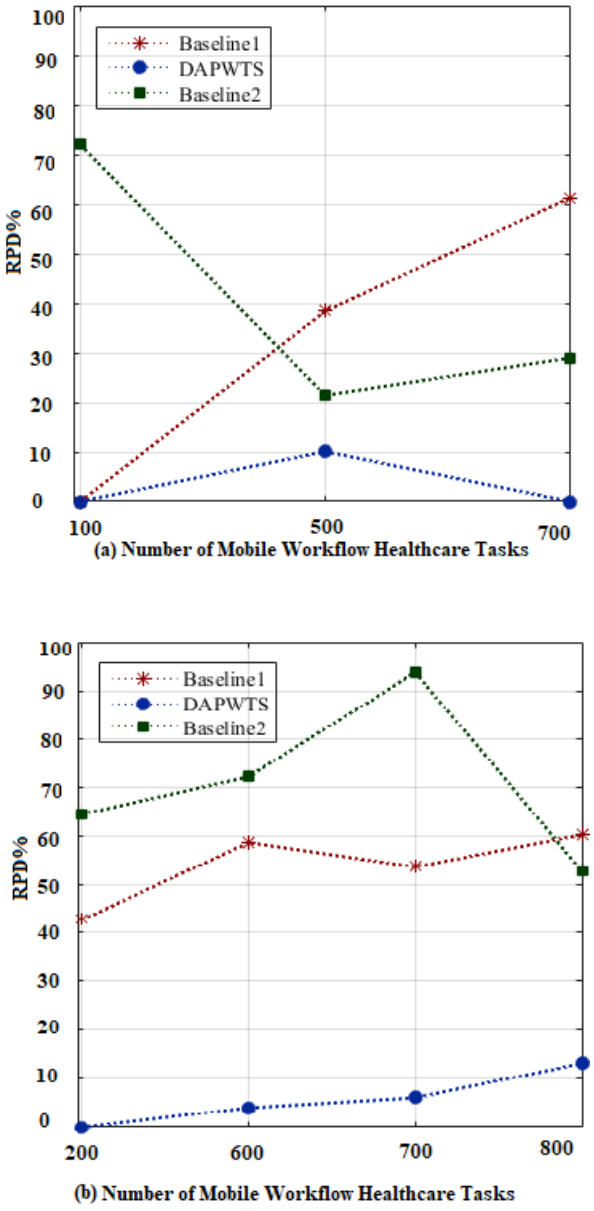



**Figure 9.** Multiple workflow application energy consumption at local devices.

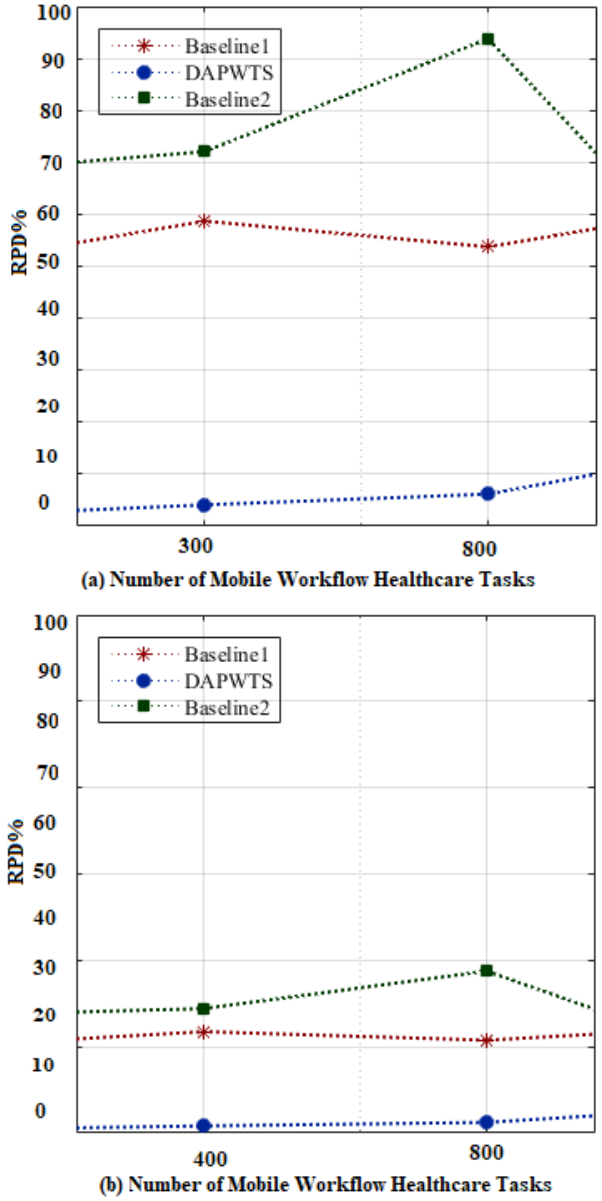



**Figure 10.** Multiple workflow applications at edge-cloud energy.

5.4.5. Failure-Aware Task Scheduling

We are formulating dynamic application and task-scheduling problems. We allow users to request and call cloud services and seamless wireless connection during mobility. However, due to the adaptive and dynamic environment, the transient failure of tasks often occurs due to resource-constrained mobile devices, wireless connection failure, or service busy failure. Thus, handling all kinds of failures is challenging without violating any QoS of tasks at runtime during offloading and scheduling. We exploited the checkpointing technique to recover a task failure from the point of collapse without any delay. DAPWTS offers a seamless and robust proposed work environment to run the mobile workflow applications without degradation of their performance. As can be observed from Figure 11, DAPWTS handled all kinds of failure with lower RPD% as compared to existing baseline offloading schemes. There are the following reasons why DAPWTS outperforms state-of-the-art studies: (i) We divided DAPWTS into four layers: mobile device, wireless network (e.g., cellular and WiFi), and edge-cloud computing. Therefore, it is easy to save the failure tasks into a particular queue at any layer; (ii) Transient failure is a temporary breakdown

that can be retrieved quickly before the given task deadline. The deadline division of the application among tasks is an efficient way to handle any failure with guaranteed performance. Hence, it is proved by Figures 11a,b, 12a,b and 13 that DAPWTS tackles all kinds of failure efficiently in the dynamic environment and encourages user interactivity and mobility features.

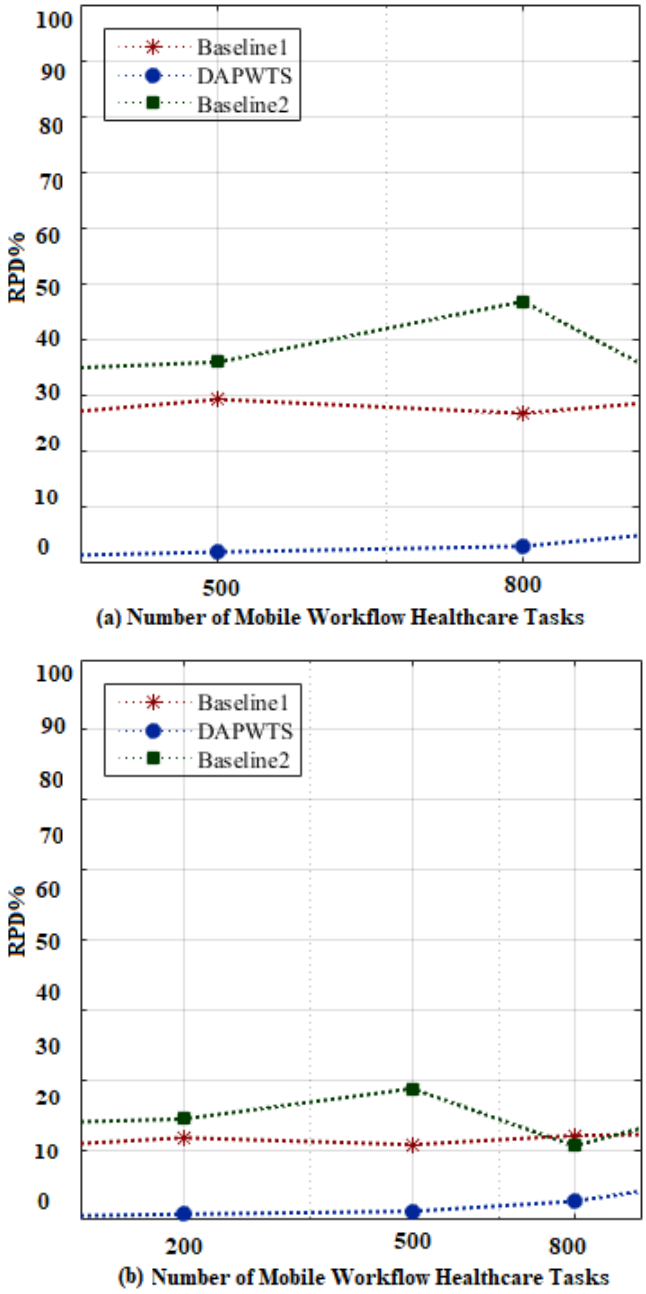



**Figure 11.** Multiple workflow application failure-aware scheduling.

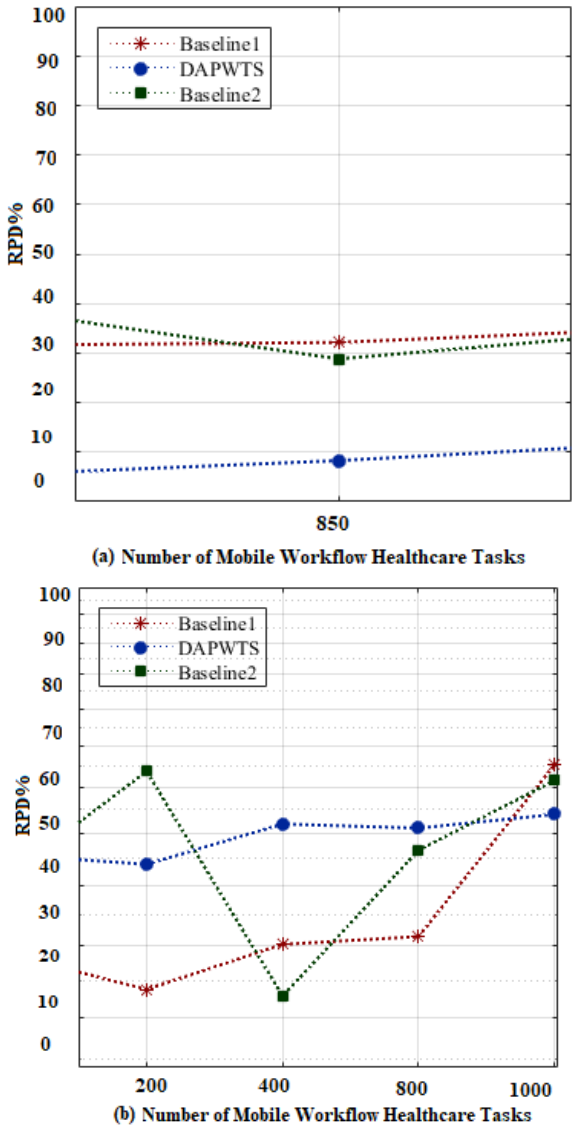



**Figure 12.** Failure energy-performance of tasks of applications G1.

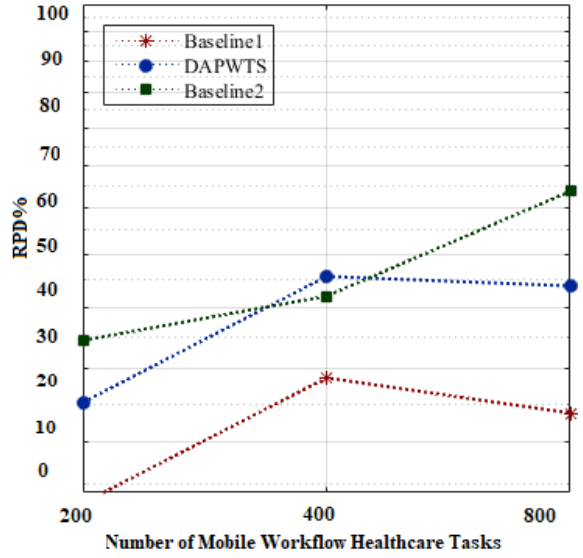



**Figure 13.** Failure energy-performance of tasks of applications G2.

### *5.5. Research Findings and Limitations*

This paper devised a dynamic application-partitioning workload and task-scheduling secure (DAPWTS) scheme for healthcare applications. The study considered heterogeneous mobile edge-cloud nodes with different speeds, resources, and power consumption. The goal was to minimize the power consumption of the nodes when running healthcare applications with deadlines, failure, and security constraints. The study obtained the optimal results during intermittent changes in resources during the runtime execution of applications. The main finding of this study is to assess the security, failure, and deadline of healthcare applications in dynamic and mobile environments. Another result is that all tasks were executed with the deadline, which means the proposed work satisfied the quality of services of applications. However, there are limitations to the study. (i) the data migration between nodes is insecure in the current work because data validation between nodes is missing. Due to many constraints in the study, it requires more resources and service availability, which means users may pay more for the application execution in the system. Therefore, cost-efficient task scheduling in terms of node price will be considered more in the following work.

## 6. Conclusions and Future Work

The study's goal was to simultaneously minimize the energy consumption of applications mentioned earlier at the resource-constrained mobile device and the edge-cloud servers. To deal with the problem, we devised a dynamic application-partitioning task scheduling (DAPWTS) framework, which consists of the following components: (i) Application-partitioning component; (ii) Task-sequencing component; (iii) Task-scheduling component; (iv) Failure-aware task scheduling of the mobile workflow in the proposed architecture. The performance evaluation results show that DAPWTS performs well in the dynamic environment and supports mobility and interactivity of users during movement. DAPWTS efficiently handles any failure during the processing of applications. Simulations showed that the DAPWTS minimized partitioning energy by 40%, resource-allocation energy by 38%, security energy by 39%, and failure energy by 29.9% for the workflow applications in the system. These results were discussed in the simulation and performance part and highlighted all findings and limitations of the study. The obtained results showed that the partitioning, scheduling, security, and failure processes consumed energy from the nodes in the system. However, existing studies only focused on the partitioning energy consumption of healthcare applications and widely ignored the security, scheduling, and failure energy of nodes in the mobile edge-cloud system.

In the future, we will add security and container-based microservices to the edge cloud to improve the system performance from the cloud side. We will formulate the future problem as a many-objectives convex optimization problem in the heterogeneous mobile edge-cloud environment.

**Author Contributions:** Data curation, A.L.; Formal analysis, A.L.; Methodology, A.L. and A.H.S.; Software, J.L.; Supervision, T.M.G.; Validation, O.T.; Visualization, P.K.; Writing—original draft, A.L. and A.S.I.; Writing—review and editing, N.A.Z. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work is financially supported by the Research grant of PIFI 2020 (2020VBC0002), Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences (SIAT, CAS), Shenzhen, China. Also, this work is partially supported by Chiang Mai University and the college of arts, media and technology.

**Data Availability Statement:** All the experimental data are generated at the local institution servers. Therefore, it cannot be made publicly available for other researchers.

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

## References


1. Kwon, D.; Yu, S.; Lee, J.; Son, S.; Park, Y. WSN-SLAP: Secure and lightweight mutual authentication protocol for wireless sensor networks. *Sensors* **2021**, *21*, 936. [CrossRef]
2. Lee, C.C. Security and privacy in wireless sensor networks: Advances and challenges. *Sensors* **2020**, *20*, 744. [CrossRef]
3. Woźniak, M. Advanced Computational Intelligence for Object Detection, Feature Extraction and Recognition in Smart Sensor Environments. *Sensors* **2020**, *21*,45.
4. Waseem, M.; Lakhan, A.; Jamali, I.A. Data security of mobile cloud computing, on cloud server. *Open Access Libr. J.* **2016**, *1*, 11. [CrossRef]
5. Shahbazi, Z.; Byun, Y.C. Integration of Blockchain, IoT and Machine Learning for Multistage Quality Control and Enhancing Security in Smart Manufacturing. *Sensors* **2021**, *21*, 1467. [CrossRef]
6. Lin, Y. An analytic computation-driven algorithm for Decentralized Multicore Systems. *Future Gener. Comput. Syst.* **2019**, *96*, 101–110. [CrossRef]
7. Liu, G.; Peng, B.; Zhong, X. A Novel Epidemic Model for Wireless Rechargeable Sensor Network Security. *Sensors* **2021**, *21*, 123. [CrossRef]
8. Khoso, F.H.; Arain, A.A.; Lakhan, A.; Kehar, A.; Nizamani, S.Z. Proposing a Novel IoT Framework by Identifying Security and Privacy Issues in Fog Cloud Services Network. *Int. J.* **2021**, *9*, 592–596.
9. Lakhan, A.; Li, X. Transient fault aware application partitioning computational offloading algorithm in microservices based mobile cloudlet networks. *Computing* **2020**, *102*, 105–139. [CrossRef]
10. Lakhan, A.; Xiaoping, L. Energy aware dynamic workflow application partitioning and task scheduling in heterogeneous mobile cloud network. In Proceedings of the 2018 International Conference on Cloud Computing, Big Data and Blockchain (ICCBB), Fuzhou, China, 15–17 November 2018; IEEE:Piscataway, NJ, USA, 2018; pp. 1–8.
11. Lakhan, A.; Li, X. Content Aware Task Scheduling Framework for Mobile Workflow Applications in Heterogeneous Mobile-Edge-Cloud Paradigms: CATSA Framework. In Proceedings of the 2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), Xiamen, China, 16–18 December 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 242–249.
12. Sharma, P.K.; Park, J.H.; Cho, K. Blockchain and federated learning-based distributed computing defence framework for sustainable society. *Sustain. Cities Soc.* **2020**, *59*, 102220. [CrossRef]
13. Mastoi, Q.-u.-a.; Ying Wah, T.; Gopal Raj, R.; Lakhan, A. A Novel Cost-Efficient Framework for Critical Heartbeat Task Scheduling Using the Internet of Medical Things in a Fog Cloud System. *Sensors* **2020**, *20*, 441. [CrossRef]
14. Lakhan, A.; Ahmad, M.; Bilal, M.; Jolfaei, A.; Mehmood, R.M. Mobility Aware Blockchain Enabled Offloading and Scheduling in Vehicular Fog Cloud Computing. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*,4212–4223. [CrossRef]
15. Pinto, M.F.; Marcato, A.L.; Melo, A.G.; Honório, L.M.; Urdiales, C. A framework for analyzing fog-cloud computing cooperation applied to information processing of UAVs. *Wirel. Commun. Mob. Comput.* **2019**, *2019*, 7497924. [CrossRef]
16. Garg, S.; Aujla, G.S.; Erbad, A.; Rodrigues, J.J.; Chen, M.; Wang, X. Guest Editorial: Blockchain Envisioned Drones: Realizing 5G-Enabled Flying Automation. *IEEE Netw.* **2021**, *35*, 16–19. [CrossRef]
17. Gill, S.S.; Tuli, S.; Xu, M.; Singh, I.; Singh, K.V.; Lindsay, D.; Tuli, S.; Smirnova, D.; Singh, M.; Jain, U.; et al. Transformative effects of IoT, Blockchain and Artificial Intelligence on cloud computing: Evolution, vision, trends and open challenges. *Internet Things* **2019**, *8*, 100118. [CrossRef]
18. Ferrag, M.A.; Shu, L.; Yang, X.; Derhab, A.; Maglaras, L. Security and privacy for green IoT-based agriculture: Review, blockchain solutions, and challenges. *IEEE Access* **2020**, *8*, 32031–32053. [CrossRef]
19. Kiwelekar, A.W.; Patil, P.; Netak, L.D.; Waikar, S.U. Blockchain-Based Security Services for Fog Computing. In *Fog/Edge Computing For Security, Privacy, and Applications*; Springer:Berlin, Germany, 2021; pp. 271–290.
20. Blasch, E.; Xu, R.; Chen, Y.; Chen, G.; Shen, D. Blockchain methods for trusted avionics systems. In Proceedings of the 2019 IEEE National Aerospace and Electronics Conference (NAECON), Dayton, OH, USA, 15–19 July 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 192–199.
21. Li, T.; Wang, Z.; Chen, Y.; Li, C.; Jia, Y.; Yang, Y. Is semi-selfish mining available without being detected? *Int. J. Intell. Syst.* **2021**, *2*, 1–21. [CrossRef]
22. Li, T.; Wang, Z.; Yang, G.; Cui, Y.; Chen, Y.; Yu, X. Semi-Selfish Mining based on Hidden Markov Decision Process. *Int. J. Intell. Syst.* **2021**, *36*, 3596–3612. [CrossRef]
23. Yu, X.; Wang, Z.; Wang, Y.; Li, F.; Li, T.; Chen, Y.; Tian, Y.; Yu, X. ImpSuic: A Quality Updating Rule in Mixing Coins with Maximum Utilities. *Int. J. Intell. Syst.* **2020**, *36*, 1182–1198. [CrossRef]
24. Li, T.; Chen, Y.; Wang, Y.; Wang, Y.; Zhao, M.; Zhu, H.; Tian, Y.; Yu, X.; Yang, Y. Rational Protocols and Attacks in Blockchain System. *Secur. Commun. Netw.* **2020**, *2020*, 8839047. [CrossRef]
25. Yang, G.; Wang, Y.; Wang, Z.; Tian, Y.; Yu, X.; Li, S. IPBSM: An optimal bribery selfish mining in the presence of intelligent and pure attackers. *Int. J. Intell. Syst.* **2020**, *35*, 1735–1748. [CrossRef]
26. Wang, Y.; Yang, G.; Li, T.; Zhang, L.; Wang, Y.; Ke, L.; Dou, Y.; Li, S.; Yu, X. Optimal mixed block withholding attacks based on reinforcement learning. *Int. J. Intell. Syst.* **2020**, *35*, 2032–2048. [CrossRef]
27. Liu, X.; Yu, X.; Zhu, H.; Yang, G.; Wang, Y.; Yu, X. A game-theoretic approach of mixing different qualities of coins. *Int. J. Intell. Syst.* **2020**, *35*, 1899–1911 [CrossRef]

28. Khan, G.; Jabeen, S.; Khan, M.Z.; Khan, M.U.G.; Iqbal, R. Blockchain-enabled deep semantic video-to-video summarization for IoT devices. *Comput. Electr. Eng.* **2020**, *81*, 106524. [CrossRef]
29. Rodrigues, T.A.; Patrikar, J.; Choudhry, A.; Feldgoise, J.; Arcot, V.; Gahlaut, A.; Lau, S.; Moon, B.; Wagner, B.; Scott Matthews, H.; et al. Data Collected with Package Delivery Quadcopter Drone. Carnegie Mellon University. Dataset. 2020. Available online: https://kilthub.cmu.edu/articles/dataset/Data_Collected_with_Package_Delivery_Quadcopter_Drone/12683453 (accessed on 10 February 2021).
30. Dovgal, V.A. Decision-Making for Placing Unmanned Aerial Vehicles to Implementation of Analyzing Cloud Computing Cooperation Applied to Information Processing. In Proceedings of the 2020 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM), Sochi, Russia, 18–22 May 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 1–5.
31. Yaqoob, S.; Ullah, A.; Awais, M.; Katib, I.; Albeshri, A.; Mehmood, R.; Raza, M.; Ul Islam, S.; Rodrigues, J.J. Novel congestion avoidance scheme for Internet of Drones. *Comput. Commun.* **2021**, *169*, 202–210. [CrossRef]