remote sensing

MDPI

Article

Managing Time-Sensitive IoT Applications via Dynamic Application Task Distribution and Adaptation

Harindu Korala *, Dimitrios Georgakopoulos, Prem Prakash Jayaraman and Ali Yavari

Department of Computer Science and Software Engineering, Swinburne University of Technology, Melbourne 3122, Australia; dgeorgakopoulos@swin.edu.au (D.G.); pjayaraman@swin.edu.au (P.P.J.); ayavari@swin.edu.au (A.Y.)

* Correspondence: hkorala@swin.edu.au

Abstract: The recent proliferation of the Internet of Things has led to the pervasion of networked IoT devices such as sensors, video cameras, mobile phones, and industrial machines. This has fueled the growth of Time-Sensitive IoT (TS-IoT) applications that must complete the tasks of (1) collecting sensor observations they need from appropriate IoT devices and (2) analyzing the data within application-specific time-bounds. If this is not achieved, the value of these applications and the results they produce depreciates. At present, TS-IoT applications are executed in a distributed IoT environment that consists of heterogeneous computing and networking resources. Due to the heterogeneous and volatile nature (e.g., unpredictable data rates and sudden disconnections) of the IoT environment, it has become a major challenge to ensure the time-bounds of TS-IoT applications. Many existing task management techniques (i.e., techniques that are used to manage the execution of IoT applications in distributed computing resources) that have been proposed to support TS-IoT applications to meet their time-bounds do not provide a sophisticated and complete solution to manage the TS-IoT applications in a manner in which their time-bounds are guaranteed. This paper proposes TIDA, a comprehensive platform for managing TS-IoT applications that includes a task management technique, called DTDA, which incorporates novel task sizing, distribution, and dynamic adaptation techniques. DTDA's task sizing technique measures the computing resources required to complete each task of the TS-IoT application at hand in each available IoT device, edge computer (e.g., network gateways), and cloud virtual machine. DTDA's task distribution technique distributes and executes the tasks of each TS-IoT application in a manner that their time-bound requirements are met. Finally, DTDA includes a task adaptation technique that dynamically adapts the distribution of tasks (i.e., redistributes TS-IoT application tasks) when it detects a potential application time-bound violation. The paper describes a proof-of-concept implementation of TIDA that uses Microsoft's Orleans Actor Framework. Finally, the paper demonstrates that the DTDA task management technique of TIDA meets the time-bound requirements of TS-IoT applications by presenting an experimental evaluation involving real time-sensitive IoT applications from the smart city domain.

Keywords: IoT; time-sensitive IoT applications; distributed IoT data analysis; distributed systems

Citation: Korala, H.; Georgakopoulos, D.; Jayaraman, P.P.; Yavari, A. Managing Time-Sensitive IoT Applications via Dynamic Application Task Distribution and Adaptation. *Remote Sens.* **2021**, *13*, 4148. https://doi.org/10.3390/rs13204148

Academic Editor: Akram Al-Hourani

Received: 10 August 2021
Accepted: 12 October 2021
Published: 16 October 2021

1. Introduction

The Internet of Things (IoT) is a novel expansion of the Internet that forms a network of heterogeneous physical objects such as sensors, video cameras, mobile phones, and industrial machines (all of which we refer to as IoT devices) that can communicate and exchange data with each other over the Internet [1–3].

In recent years, the quantity of data generated from IoT devices (we refer to this data as IoT data) has grown significantly and there has been a great deal of interest in extracting valuable insights from these data [4]. To accomplish this, IoT applications collect IoT data from appropriate IoT devices and produce high-value information by analyzing the collected IoT data.

In this paper, we focus on IoT applications that must complete the tasks of collecting IoT data and analyzing them within an application specific time-bound to produce high-value results. If this is not achieved, the value of such applications and the results they produce depreciates. We refer to such applications as Time-Sensitive IoT (TS-IoT) applications and such time-related requirements of the data analysis as time-bound requirements. We can consider a vehicle accident prediction application that must (1) gather IoT data generated from sensors located at traffic lights, vehicles, and traffic cameras deployed at intersections, (2) analyze the collected data to predict a possible accident using many machine learning techniques [5,6], and (3) prevent the accident by informing the corresponding driver in near real-time (e.g., within a 30 ms time-bound). If there is any additional time (i.e., more than the application time-bound) involved in completing the data analysis, the predicted accident information will not be beneficial to prevent the accident. To discuss further the problem of satisfying time-bound requirements, consider that TS-IoT applications are comprised of a set of tasks and are intrinsically distributed. Each of these tasks may need to perform one of the following: collect IoT data from heterogeneous IoT devices, process the collected data using various techniques such as stream processing and resource-intensive machine learning and statistical techniques, manage the storage requirements for stateful data analysis, and maintain the data analysis pipelines (i.e., results produced by a task may be used as an input to another task/tasks in the same application). Currently, TS-IoT applications that are comprised of such tasks are executed in distributed IoT environments.

The IoT environments consist of various heterogeneous distributed computing resources [3,7]. These computing resources include: (1) IoT devices that are directly connected with sensors, which generate IoT data; (2) edge-computers, i.e., any computing resource that is closer than the cloud to the IoT devices and is not directly connected to them (e.g., network gateways, dedicated edge computers); and (3) cloud data centers. These computing resources are connected to each other and/or to the Internet via multiple heterogeneous networks (e.g., narrowband/NB-IoT, LoRa, BLE, and Wi-Fi). Moreover, IoT environments tend to be volatile due to (1) unpredictable IoT data generation rates, (2) uncertain availability of resources caused by mobility, connection issues, etc., and (3) overloaded resources ascribed to multitenancy (i.e., multiple applications utilizing the same set of resources).

Ensuring the time-bound requirements of TS-IoT applications is largely determined by the total application execution time. This can be computed as the summation of the total data processing time and the total data communication time. The total data communication time is influenced by the relevant network delays involved in moving IoT data to corresponding resources to process them, whereas the total data processing time is influenced by the computing resource where the data analysis is performed. Therefore, guaranteeing the time-bound requirements depends significantly on the choice of suitable resources from the IoT environment. Nevertheless, there are trade-offs in the selection of the cloud virtual machines, edge-computers, and/or IoT device resources to execute a TS-IoT application [8,9]. Although analyzing IoT data on IoT devices yields the lowest communication delays, IoT devices have limited computing resources. Edge computers have greater computing resources than IoT devices, but they are likely to suffer from more communication delays than IoT devices. Cloud virtual machines offer virtually unlimited resources [8]; therefore, several researchers [10] have used cloud computing resources to process large quantities of sensor data, such as remote sensing data. However, processing these data in the cloud induces significant communication delays when IoT data is transferred to the cloud [11,12]. Moreover, each task of the TS-IoT application also has diverse resource requirements. For example, a machine learning classification task may require more computing resources than a simple data aggregation task. Therefore, although it is usually feasible to meet the time-bound requirements of each TS-IoT application by distributing tasks for execution in the IoT devices, edge computers, and the cloud, we must determine the best possible distribution of tasks from the perspective of communication and computing resource constraints. However, determining the task distribution for TS-IoT

applications is more difficult than for other applications due to the volatility of the IoT environment and the unpredictability of IoT data streams. Therefore, task distributions of TS-IoT applications may need to be dynamically adapted to deal with possible time-bound violations caused by the unpredictable and volatile nature of the IoT environment. Thus, this has become a major research challenge.

Many existing studies in IoT have proposed various task management techniques (i.e., techniques that are used to manage the execution of IoT applications in distributed computing resources) to address the research challenge of supporting the time-bound requirements of TS-IoT applications. These task management techniques include: (1) task sizing techniques that determine the amount of computing and networking resource needed to complete the execution of TS-IoT application tasks; (2) task distribution techniques that distribute and execute TS-IoT application tasks in the IoT environment; and (3) task adaptation techniques that dynamically adapt the distribution of TS-IoT application tasks to mitigate possible time-bound violations. However, existing task sizing techniques [13–15] have depended on simulation tools [16] or include limited testbed implementations. Thus, they cannot accurately estimate the suitable quantity of resources required for TS-IoT application tasks. Existing task distribution techniques have employed computationally expensive complex optimization techniques [17–20] because they have been built for use with distributed real-time systems that include controlled execution environments, unlike volatile IoT environments, and most of them cannot scale with a large number of IoT devices. Thus, existing task distribution techniques are not suitable to use with TS-IoT applications that have stringent time-bound requirements. Task adaptation techniques [18,21] that are found in the literature involve higher overheads and most of them adapt the distribution of tasks in a reactive manner (i.e., after noticing the application's execution time exceeded the time-bound requirements). Finally, there is a lack of comprehensive task management techniques that integrate task sizing, task distribution, and task adaptation techniques to collectively facilitate the TS-IoT applications to meet their time-bound requirements.

To overcome this shortcoming, in this paper, we propose a novel Time-Sensitive IoT Data Analysis platform called TIDA that includes a task management technique, called DTDA, which utilizes the computing resources available in the IoT devices, edge computers, and the cloud for meeting the time-bound requirements of each TS-IoT application when the entire pool of available computing resources is sufficient to collectively achieve this. Specifically, this paper significantly extends our earlier work [3] and makes the following novel contributions:

1. A novel task management technique called Dynamic Task Distribution and Adaptation (DTDA) that integrates the following techniques:
    - A task sizing technique that utilizes an unsupervised machine learning technique to extract the most important features from task sizing measurement data (i.e., computing and network resources required to complete each task of a time-sensitive IoT application).
    - A greedy task distribution technique that distributes and executes the TS-IoT application in the IoT environment, which was presented in [3].
    - A novel task adaptation technique that dynamically adapts and re-distributes TS-IoT application tasks when it predicts a time-bound violation (i.e., the TS-IoT application cannot meet its time-bound requirements).

Our earlier work [3] presented a greedy task distribution technique that is unable to dynamically adapt the task distribution in real-time to cope with the volatile changes in the IoT environment. Hence, the proposed DTDA technique presented in this paper aims to address this key limitation of the greedy task distribution technique.

2. A complete implementation of the improved task sizing and novel task adaptation techniques using Microsoft's Orleans Actor framework.

3. A comprehensive experimental evaluation using a real-world smart city use case and related dataset that includes: (a) an experimental setup that forms four clusters of computing resources with heterogeneous system configurations to validate the DTDA task management technique; and (b) a comprehensive comparison with two current state-of-the-art task management techniques that shows how well the TIDA platform, which implements the above techniques, meets the time-bound requirements of TS-IoT applications.

The remainder of the paper is organized as follows. Section 2 presents the related work, Section 3 presents a motivating use case scenario, Section 4 describes the system model and problem formulation, Section 5 discusses the dynamic task distribution and adaptation technique, and Section 6 presents the design and implementation of the TIDA platform. Section 7 presents the experimental evaluation results and Section 8 concludes the paper and outlines potential future work.

2. Related Work

TS-IoT applications are executed in an IoT environment that connects a variety of IoT devices such as sensors, mobile phones, cameras, and industrial machines with each other and/or to the Internet. Intrinsically, this environment is highly distributed. Traditional distributed real-time systems consist of a set of computer resources interconnected by a real-time communication network [22]. Generally, distributed real-time systems include a controlled execution environment (i.e., an environment where applications are executed). In addition, distributed real-time systems found in power plants and factory control systems are not connected to the Internet. In these distributed real-time systems, attributes of the execution environment, such as latencies of networks, data generation rates, and available computing resources, can be determined in advance [23,24]. On the contrary, although the IoT environment is highly distributed, it has unique characteristics that impose additional challenges for the execution of TS-IoT applications in such a manner that their time-bound requirements are met compared to traditional distributed real-time systems [3,8,25]. We further discuss these additional challenges of the IoT environment in Section 3 via a motivation scenario, and in Section 4 by presenting a formal model for the IoT environment.

Many techniques can be found in the literature that have been developed for distributed real-time systems and aim to optimize the execution of real-time applications in terms of execution time [26], cost of execution [27], and energy consumption [28]. Furthermore, due to the controlled execution environment of the distributed real-time systems, these techniques largely consist of optimization-based scheduling techniques [22,29–31] that can effectively determine which task needs to be executed, at what time it needs to be executed, and the resource that should be used for the execution. However, such optimization-based techniques are only effective in controlled and predictable execution environments. More specifically, in this paper, we focus on techniques for dynamically distributing and adapting the tasks of TS-IoT applications by determining the relevant communication delays involved and the needed computing resource capacities. However, adaptive scheduling techniques found in distributed real-time systems determine which tasks need to be executed, at which times, to meet the deadlines of the system. Therefore, the task scheduling techniques developed for distributed real-time systems cannot be used for TS-IoT application task distribution. Due to this reason and the unique characteristics of the IoT environment (such as heterogeneity and volatility) compared to traditional distributed real-time systems, techniques developed for distributed real-time systems cannot be effectively used with TS-IoT applications to meet their time-bound requirements. Therefore, many studies have been recently conducted to devise suitable techniques for facilitating TS-IoT applications to meet their time-bound requirements.

Meeting the time-bound requirements of TS-IoT applications is challenging due to the heterogeneous and volatile nature of the IoT environment and the time-bound requirements of such applications [8,32]. In one of our previous works [3], we proposed an approach for

dealing with these challenges that involves distributing TS-IoT applications in a collection of interrelated tasks and selecting the appropriate IoT computing and network resources to execute the tasks of each application in such a manner that they collectively meet the application's time-bound requirements. To enable such task distribution, a task sizing technique was proposed for estimating the computing and network resources required by the tasks of TS-IoT applications. Related work in determining the most suitable IoT resources for computing IoT application tasks includes [13,14], which investigated (1) how to estimate the computing resources required by cloud-based IoT applications based on historical performance metrics, and (2) evaluated various techniques for achieving this via the Cloudsim simulator. Zeng et al. [15] presented a simulator called IOTSim to analyze IoT applications to understand how they would perform in cloud computing environments. In [16], the researchers proposed a technique for measuring the performance of computing resources when different IoT application tasks are executed, whereas Korala et al. [33] introduced a platform to experimentally evaluate the performance of TS-IoT applications. Alhamazani et al. [34] proposed another approach for measuring performances of IoT applications across multiple cloud environments. Souza et al. [35] proposed a novel framework to take measurements of IoT microservices-based applications deployed across edge and cloud environments.

Most related research in task distribution has considered this problem as an optimization problem and proposed various optimization techniques (such as linear programming, non-linear programming, and heuristic techniques) for this purpose. For example, Taneja et al. [36] proposed a technique for efficient distribution of application tasks across cloud and edge resources in a resource-aware manner. Skarlat et al. [37] proposed an optimization technique that generates a task execution plan for IoT applications. Hong et al. [17] introduced a technique for optimizing the scheduling of IoT application tasks in edge devices. Yousefpour et al. [18] formulated IoT application distribution as an Integer Non-Linear Problem (INLP). The authors then used INLP to minimize the cost of resource usage while satisfying the QoS requirements of the applications. The optimization techniques proposed by [12,19,20] determine appropriate computing resource selection for meeting the QoS requirements of IoT applications. Related computing frameworks and tools, such as those in [38–42], have employed similar techniques to manage the distribution of TS-IoT applications. Zhang et al. [43] proposed a recommender system for dealing with the heterogeneity of cloud computing resources.

The IoT is subject to uncertainties, including unpredictable IoT data, volatile and mobile computing resources, and overloaded resource-constrained computing resources, due to multitenancy [35]. Therefore, task execution plans may need to be adapted dynamically to deal with possible time-bound violations. To address this, many related studies have presented dynamic task redistribution techniques, which involve generating new task execution plans periodically and/or in instances where certain computing resources become overloaded. Yousefpour et al. [18] introduced a technique to dynamically adapt the task execution techniques by periodically releasing and deploying tasks when there are QoS requirement violations. Skarlat et al. [21] also proposed a technique to dynamically redistribute the tasks when certain resources are overloaded and/or disconnected from the IoT environment. Xu et al. [44] proposed a solution that can recommend appropriate virtual machines for IoT application workloads in edge-cloud environments using a tree-based machine learning algorithm to predict performance metrics for these IoT application workloads.

In summary, optimization-based techniques that have been developed for distributed real-time systems cannot be used with TS-IoT applications to enable them to meet their time-bound requirements due to the volatile nature of the IoT environment. Task sizing techniques found in the literature have relied on simulation tools [13,14] or include limited testbeds [16] for sizing tasks using estimations. These techniques cannot effectively estimate the resources needed by TS-IoT application tasks because they do not deal with the heterogeneity and dynamic nature of the IoT environment. Most of the task distribution

techniques in the literature employ complex optimization techniques [17–20] in device task execution plans; furthermore, most of them do not consider task sizing and are expensive to compute. Existing dynamic task adaptation techniques [18,21] have mainly employed task redistribution techniques and they dynamically adapt the task execution plans in a reactive manner (i.e., after observing that the application has failed to meet the time-bound requirement or after observing the resources are overloaded), which is not an effective solution to guarantee the time-bound requirements. Due to these reasons, these techniques are not suitable for TS-IoT applications that have demanding time-bound requirements. On the contrary, TIDA includes a Dynamic Task Distribution and Adaptation (DTDA) task management technique that integrates: (1) a task sizing technique that measures the computing and network resources required by the tasks when they are executed in the IoT environment; (2) a greedy task distribution technique that uses the task sizing information to generate time-bound satisfying task execution plans to distribute tasks in the IoT environment; and (3) a dynamic task adaptation technique that utilizes a predictive machine learning model to accurately predict possible time-bound violations and make necessary adaptations to the task execution plans to ensure the time-bound requirements are met. Furthermore, TIDA was implemented by extending Microsoft Orleans and the greedy algorithm was evaluated using a real-world smart city application.

3. Smart City Passenger Counting Application—Motivating Scenario

Let us consider a smart city application that requires an accurate count of passengers for a public transport system in near real-time. The passenger count information is used by the transport service to improve planning and scheduling of buses, allocate buses or trains to meet the actual demand, and respond to unplanned incidents such as bus breakdowns and accidents. Figure 1 illustrates the motivating scenario, computing resources, and IoT data analysis tasks in this TS-IoT application.

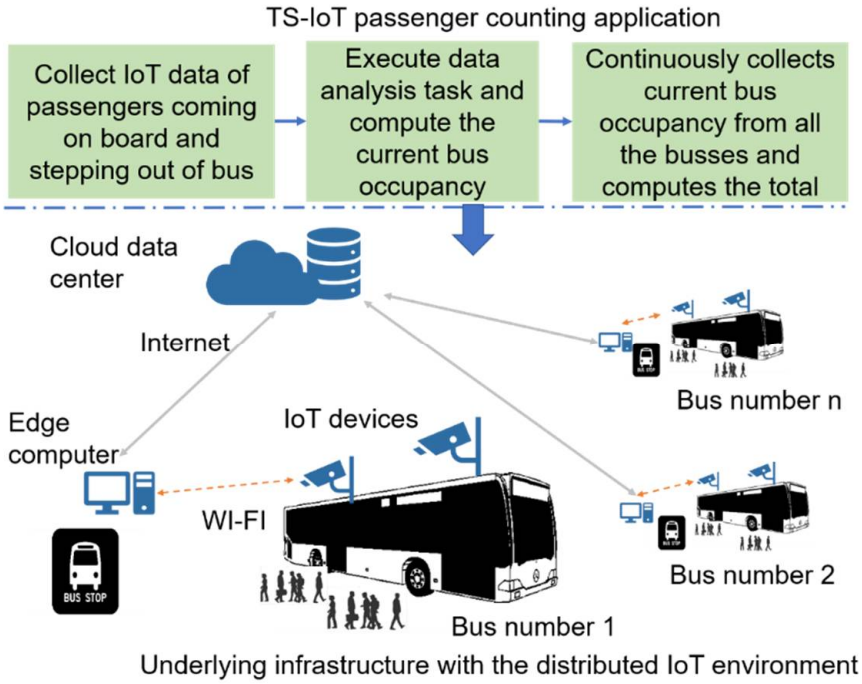

Figure 1. Illustration of motivating scenario.

To count passengers in this smart city environment we utilized the following IoT devices, edge computers, and cloud resources:

1. Orbbec Persee IoT devices providing a combination of RGB and infrared cameras with a fully functioning onboard computer were mounted above the doors of each bus. We used these devices to count the passengers stepping in and out of each bus at

each bus stop in the transport network. The IoT data generated by these IoT devices included (1) video data, (2) depth sensor data, and (3) infrared data at 30 frames per second. In addition to generating a large volume and variety of IoT data from their sensors, the Orbbec Persee devices provide internal computing and storage resources.

2. Edge computers at bus stops and train stations. These edge computers act as gateways for IoT devices and connect to the cloud data center via the Internet. Furthermore, edge computers also include additional computing and storage resources that can be used for IoT data analysis.
3. A cloud data center with virtually unlimited computing resources.

In this IoT environment, the IoT devices, edge computers, and the cloud are connected via different networks (e.g., NB-IoT, 4G, and broadband). The Orbbec Persee IoT devices incorporate Wi-Fi cards and, as a result, they can connect to the edge computer at each bus stop. In addition, these IoT devices can also be directly connected to the cloud via 4G during the entire bus journey. However, IoT devices can connect to edge computers only when they are near bus stops or train stations. The edge computers and the cloud data centers are connected via broadband Internet. To compute the occupancy of each bus and the total occupancy, this TS-IoT application must perform the following: (1) capture passenger data while passengers are stepping in and out of each bus; (2) analyze the collected RGB/infrared/depth data and recognize individual passengers; and (3) compute the occupancy of each bus at each bus stop and the entire transport network. This task may involve the following sub-tasks: (1) pre-processing the collected RGB/infrared/depth data; (2) classifying passengers as entering or exiting by applying classification techniques such as the Haar-cascade classifier (please note that, in this paper, we consider the classifier to be an already trained classifier; hence training the classifier is not considered to be an IoT data analysis task and is not discussed further in this paper); (3) calculating the total occupancy of the bus; (4) computing the total occupancy of all the busses in the transport network.

The IoT passenger count application has a variable timebound that is hard to meet, i.e., it fails to meet its time-bound requirement when any bus reaches the next bus stop before its occupancy information from the previous bus stop is counted. Meeting the time-bound requirements in the IoT often depends on the appropriate selection of computing and networking resources for each TS-IoT application. In the passenger counting IoT application, although we perform the entire data analysis quickly in the cloud, this may involve a significant communication delay to collect all the passenger RGB/infrared/depth data. Offloading the collected passenger data to the edge computers and performing the data analysis in the edge computers is another option. However, we only have a limited time to transfer the passenger data to the edge computer and the computing resources in edge computers are more limited than in the cloud. Processing data in an IoT device itself is another option that is viable only if an IoT device has enough computing resources available for the tasks of the IoT application at hand.

Therefore, to meet the time-bound requirements of this and any other TS-IoT application, we must determine the best possible distribution of the data analysis tasks that comprise the TS-IoT application from the perspective of providing enough computing resources and communication capacity, and compute the assigned analysis tasks in a manner such that the entire TS-IoT application meets its time-bound(s).

However, as discussed in Section 2, unlike distributed real-time systems, the IoT environment introduces additional challenges. The IoT environment consists of heterogeneous computing resources connected by multiple networks, which have different data transfer rates, latencies, and bandwidth values. Moreover, the IoT environment tends to be volatile due to (1) unpredictable IoT data generation rates; (2) uncertain availability of resources caused by mobility, connection issues, etc.; and (3) overloaded resources attributed to multi-tenancy [3,8,25]. Therefore, TS-IoT applications require adaptation of their task distribution during the runtime to continue to satisfy the application time-bounds. For example, the number of passengers onboard at each bus stop varies and cannot be predicted; thus, the passenger onboarding data fluctuates throughout the bus journey. Moreover, some comput-

ing resources (e.g., the cloud virtual machines) that are processing the TS-IoT application tasks may disconnect from the IoT environment due to network issues. Additionally, some computing resources such as edge computers may become overloaded due to multiple applications competing for resources, in addition to simply not having enough available resources to handle the incoming variable IoT data. Hence, the computing and storage resources of the IoT are volatile. As a result of this, although we determine the best possible distribution of tasks for the TS-IoT application before deployment, it is a challenging task to meet the application time-bounds during the application runtime. Moreover, due to these reasons, existing optimization-based techniques devised for distributed real-time-systems are not suitable and are ineffective in supporting the time-bound requirements of the TS-IoT applications, because it is necessary to dynamically adapt the distribution of the tasks of TS-IoT applications to deal with possible time-bound violations caused by the volatile nature of the IoT environment.

4. System Model & Problem Formulation

Due to the trade-offs between IoT resources in the distributed IoT environment, it is necessary to generate a task execution plan (which meets the application's time-bounds requirement) by determining the relevant communication delays involved and the needed computing resource capacities for each task. A task execution plan is a mapping of the tasks of the TS-IoT application to the corresponding computing resources where the tasks are executed. To address this, first, we present a formal description of the resources in the IoT environment and the TS-IoT applications. Then, we formulate the task distribution problem as an optimization problem.

4.1. Resource Model

Computing resources (i.e., IoT devices, edge computers, and the cloud) and network resources in the distributed IoT environment form a graph $G_{Res} = (Comp\_Res\ ,\ Network\_Res)$, where $Comp\_Res$ represents the distributed computing resources and $Network\_Res$ represents the network links between computing resources. A single computing resource of G_{Res} can be denoted as cr_i, where $cr_i \in Comp\_Res$ and $i \in 1 \ldots m$, m is the total number of computing resources in G_{Res}. Each cr_i has an attribute called $Available\_Res_{cr_i}$, which is the amount of computing and storage resources available at cr_i. Further, $Available\_Res_{cr_i}$ can be represented as a tuple of $\langle cpu^i_{cr}\ ,\ ram^i_{cr} \rangle$. Figure 2 shows the IoT environment graph of the resource model.

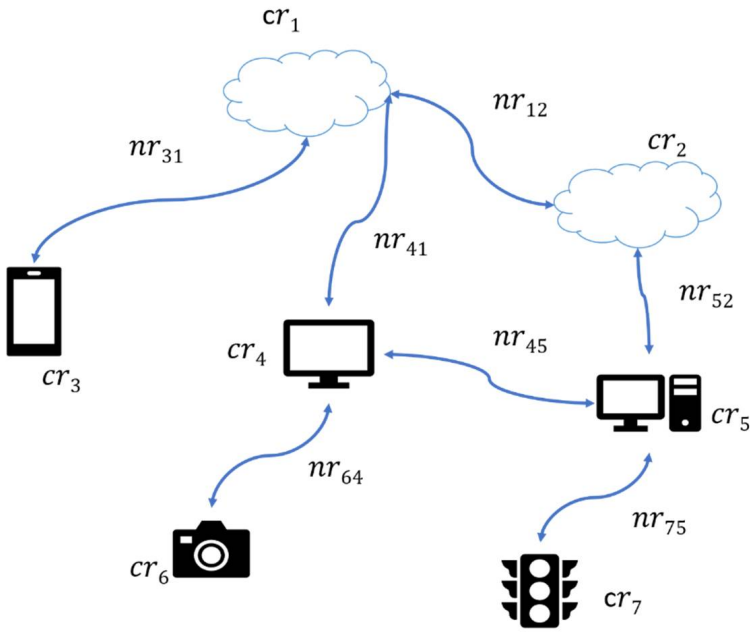

Figure 2. IoT environment graph.

A single network link of the G_{Res} represents the network resources of a network link between two computing resources, cr_i and cr_j. This can be denoted as $nr_{ij} \in Network\_Res$, where i and j denote the corresponding indexes of the two computing resources that are connected via network link nr_{ij}. Each nr_{ij} has the following attribute: $Available\_Band_{nr_{ij}}$ is the amount of available bandwidth of the network resource link nr_{ij}. Furthermore, $Available\_Band_{nr_{ij}}$ is captured by a tuple $\langle up^{ij}, down^{ij} \rangle$ where up^{ij} is the amount of available upload bandwidth and $down^{ij}$ is the amount of available download bandwidth in nr_{ij}.

As discussed in Section 3, due to the volatile nature of the IoT environment, $Available\_Res_{cr_i}$ of computing resources, the number of connected computing resources in the IoT environment, and $Available\_Band_{nr_{ij}}$ of the network links vary over time and the degree of change is hard to predict.

4.2. Application Model

A TS-IoT application is comprised of a set of (possibly inter-dependent) tasks that interact via data exchanges. A TS-IoT application can be represented as a directed acyclic graph (DAG), $G_{App} = (Tasks, Dataflows)$, where *Tasks* represents the tasks of the TS-IoT application and $Dataflows$ represents the data flows between Tasks. Each TS-IoT application has a time-bound requirement and we denote it as TB_{App}.

A single task of the G_{App} can be denoted as t_i, where $t_i \in Tasks$ and $i \in 1 \ldots n$, where n is the total number of tasks in G_{App}. Each task can be of two types: stateful tasks and stateless tasks. Stateful tasks require buffering a certain number of data items before processing them. We identify the number of data items required to buffer in a stateful task as the queue size and denote this as q_{t_i}. Stateless tasks do not require buffering of data items during their data processing; therefore, we consider q_{t_i} of stateless tasks to be 1. Furthermore, to identify whether a task is stateful, we denote the following binary attribute, $is\_stateful_{t_i}$: $is\_stateful_{t_i} = 1$ if the task t_i is a stateful task and $is\_stateful_{t_i} = 0$ otherwise. With the current proposed model, we assume that the tasks run continuously; hence, we do not consider any loop variables (i.e., control variables) for this model at this stage.

Each t_i, has the following attributes: $Task\_Res_{t_i}$ is the amount of computing resources required for the execution of t_i; $Proc\_time_{t_i}$ denotes the time taken to process the IoT data at a specific computing resource. This depends on the computing resource where the task is executed. A t_i is also associated with two delays. We denote these as $Start\_Delay_{t_i}$ and $Wait\_Delay_{t_i}$. The time taken to produce the first data item during IoT data processing is denoted by $Start\_Delay_{t_i}$ and the delay between producing data items is denoted as $Wait\_Delay_{t_i}$. We assume that the aforementioned attributes can be obtained by measurements.

A single dataflow of G_{App} represents the dataflow (i.e., data transfer) between the predecessor task t_i and successor task t_j, and this can be denoted as d_{ij}, where $d_{ij} \in Dataflows$. i and j denote the indexes of the corresponding tasks. In our model, we assume that data is transferred piece by piece. Each d_{ij} has the following attributes: $Data_{d_{ij}}$ is the size of a single data piece transferred through d_{ij}; the amount of time to send a single piece of data via a network link is denoted as $Com\_delay_{d_{ij}}$. Figure 3 shows the TS-IoT application graph.

The above model is based on the following assumptions:

- We assume that cloud data centers in the IoT environment have a large amount of computing and storage resources [8], whereas IoT devices and edge computers have limited computing and storage resources.
- We assume the $Task\_Res_{t_i}$ can be obtained by measurements via executing the corresponding task on a reference computing resource.
- We assume the $Proc\_time_{t_i}$ on a computing resource can be obtained by estimation based on previous measurements.
- We assume the G_{Res} is developed by considering the number of computing resources and their networks available in the IoT environment.

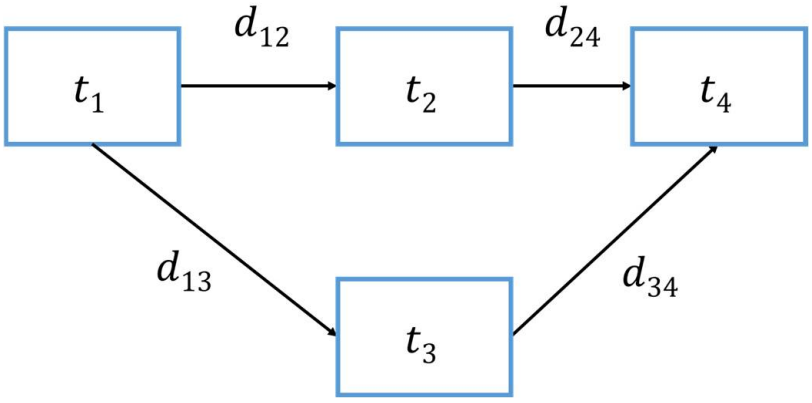

Figure 3. TS-IoT application graph.

4.3. Problem Formulation

Our objective is to generate an application-specific, time-bound satisfying task execution plan for the IoT environment within the available resources. To realize this, we need to generate a task execution plan in an IoT environment in a manner such that the end-to-end response time of the TS-IoT application is within the time-bound requirement of the application. Furthermore, in this model, we consider TS-IoT application graphs with multiple paths, and to capture this we consider the end-to-end response time of the critical path in the graph. We define this critical path of the application graph as a set of tasks and dataflows, forming a path, for which the end-to-end response time is maximal. We refer to this end-to-end response time of the application as *Total Application Execution* Time and denote it as $Total\_Exec_{App}$. Given this definition, we can formulate the following equation:

$$Total\_Exec_{App} = \max_{p \in 1...Paths} (Path\_Exec\_Time_p) \quad (1)$$

where $Path\_Exec\_Time_p$ is the end-to-end execution time along the path p and $Paths$ is the total number of paths in G_{App}. For any path p, we can calculate the $Path\_Exec\_Time_p$ as the summation of execution times (i.e., summation of data processing time at tasks and delays involved in bringing data to the task, buffering data at tasks, etc.) of each task that is in that path p. Given this definition, we obtain the following:

$$Path\_Exec\_Time_p = \sum_{j=1}^{Y} Task\_Exe_{t_j} \quad (2)$$

where Y is the total number of tasks in the path p and $Task\_Exe_{t_j}$ is the execution time of the j^{th} task in the path p of G_{App}. $Task\_Exe_{t_j}$ can be calculated from the following:

$$Task\_Exe_{t_j} = Proc\_time_{t_j} + Com\_delay_{d_{ij}} \cdot q_{t_j} + Wait\_delay_{t_i} \cdot q_{t_j} + Start\_delay_{t_i} \quad (3)$$

In Equation (3), $Proc\_time_{t_j}$ is the amount of time taken to process IoT data by t_j. $Com\_delay_{d_{ij}}$ is the amount of time taken to transfer a single data item from the predecessor task t_i to the task at hand t_j via d_{ij}. To capture the total $Com\_delay_{d_{ij}}$, we multiply this with the queue size of t_j, which we denote as q_{t_j}. Note we do not need to consider the maximum of $Com\_delay_{d_{ij}}$ because we apply this equation on a single path of the graph, and at the end, the critical path is chosen using Equation (1). We assume $Com\_delay_{d_{ij}}$ to be 0 if the two tasks (i.e., t_i and t_j) are executed in the same computing resource. $Start\_delay_{t_i}$ is the time taken to produce the first data item by the predecessor task t_i and $Wait\_delay_{t_i}$ is the delay between producing data items at the predecessor task t_i. For stateful tasks to capture the total $Wait\_delay_{t_i}$, this is multiplied by q_{t_j} (i.e., the queue size of the task t_j). $Com\_delay_{d_{ij}}$ can be calculated using the following:

$$Comm\_delay_{d_{ij}} = \frac{Data_{d_{ij}}}{Available\_Band_{nr_{ij}}} \quad (4)$$

where $Data_{d_{ij}}$ denotes the size of a single data piece that needs to be sent to t_j from predecessor task t_i via d_{ij} that is placed on network link nr_{ij} and $Available\_Band_{nr_{ij}}$ is the available bandwidth of the $nr_{ij.}$

Decision variables: We define the decision variables that form the task execution plan as follows: the first decision variable $\alpha_{t_j}^{cr_i}$ denotes whether a task t_j is distributed on a computing resource cr_i. The next decision variable $\gamma_{d_{ij}}^{nr_i}$ denotes whether a dataflow d_{ij} is placed on a network resource nr_i.

Constraints: First, the task distribution of computing resources and dataflow placement on network link resources must not exceed the available resources of those corresponding computing and network resources. A task t_j can be distributed in the computing resource cr_i if $Available\_Res_{cr_i}$ is at least equal to or more than $Task\_Res_{t_j}$ of t_j. We can formally denote it as follows:

$$\forall\, cr_i \in\ Comp\_Res,$$

$$\sum_{t_j}^{Tasks} Task\_Res_{tj} * \ \alpha_{t_j}^{cr_i} \ \leq Available\_Res_{cri} \tag{5}$$

Each network link can only transfer data that is within its available bandwidth and we can formally denote it as follows:

$$\forall\, nr_{ij} \in\ Network\_Res$$

$$\sum_{d_{ij}}^{Dataflows} Data_{d_{ij}} * \ \gamma_{d_{ij}}^{nr_{ij}} \leq\ Available\_Band_{nr_{ij}} \tag{6}$$

where $Data_{d_{ij}}$ denotes the amount of data transfer between task t_i and t_j via network link nr_{ij} and $\gamma_{d_{ij}}^{nr_{ij}}$ is the binary variable denoting whether a dataflow d_{ij} is placed on a network resource nr_{ij}.

Regarding the second constraint, TS-IoT applications must satisfy their time-bound requirements. We can formally denote it as follows:

$$Total\_Exe_{App} \ \leq\ TB_{App} \tag{7}$$

Objective function: The objective of the task distribution problem is to devise a task execution plan in an IoT environment that yields the minimum application execution time while satisfying the time-bound and resource constraints. We formally denote it as follows:

Minimize:

$$Total\_Exec_{App} = \max_{p\ \in\ 1 \ldots Paths} (Path\_Exec\_Time_p) \tag{8}$$

Subject to: Equations (5)–(7).

However, solving this problem via optimization techniques tends to be NP-hard, and to ensure the time-bound requirements of TS-IoT applications in a such volatile IoT environment we need to generate time-bound satisfying task execution plans at the rate of the change occurring in the IoT environment. However, existing optimization-based techniques devised for distributed systems are ineffective. Hence, we aim to solve this problem using a novel dynamic task distribution and adaptation technique described in the next section. Table 1 lists the definitions of all notations we used in the above system model and the problem formulation.

Table 1. Notations used in the system model and problem formulation.

Notation	Definition
G_{Res}	Gra ph representing the IoT environment
$Comp\_Res$	Distributed computing resources in the IoT environment
$Network\_Res$	Network links between computing resources
cr_i	i^{th} computing resource
m	Total number of computing resources in G_{Res}
$Available\_Res_{cr_i}$	The quantity of resources available at i^{th} computing resource
nr_{ij}	The network resources of a network link between two computing resources, cr_i and cr_j
$Available\_Band_{nr_{ij}}$	The amount of available bandwidth of the network resource link nr_{ij}
G_{App}	Graph representing the TS-IoT application
$Tasks$	Tasks of the TS-IoT application
$Dataflows$	Dataflows between tasks
TB_{App}	Time-bound requirement of the TS-IoT application
t_i	i^{th} task of the TS-IoT application
n	Total number of tasks in G_{App}
y	Total number of different paths throughout the application G_{App}
$is\_stateful_{t_i}$	Binary variable denoting whether a task is stateful or not
q_{t_i}	Queue size of the i^{th} task
$Task\_Res_{t_i}$	The amount of computing resources required for the execution of the task t_i
$Proc\_time_{t_i}$	Time taken to process the IoT data at a specific computing resource by task t_i
$Start\_Delay_{t_i}$	Time taken to produce the first data item during IoT data processing of task t_i
$Wait\_Delay_{t_i}$	The delay between producing data items of task t_i
d_{ij}	Dataflow between the predecessor tasks t_i and successor task t_j
$Data_{d_{ij}}$	Size of a single data piece transferred through d_{ij}
$Com\_delay_{d_{ij}}$	The amount of time to send a single piece of data via a network link d_{ij}
$Total\_Exec_{App}$	Total application execution time
$Path\_Exec\_Time_p$	End-to-end execution time along the path p of graph G_{App}

5. Task Management Technique That Includes Task Sizing, Distribution and Adaptation Techniques

In this section, we present three novel techniques we devised for meeting TS-IoT application time-bounds as part of the task management technique. The task sizing technique is presented in Section 5.1 and its results are used by the task distribution technique that is discussed in Section 5.2. The task adaptation technique adapts the task distribution by redistributing or reconfiguring individual tasks, whenever a potential violation of the TS-IoT applications time-bound is detected.

5.1. Task Sizing Technique

The task sizing technique is used for: (1) measuring the computing and network resources needed for the execution of TS-IoT application tasks in the available IoT devices, edge computers, and the cloud; and (2) the execution times of the TS-IoT application tasks. Unlike the existing task sizing techniques found in the literature, instead of estimating the computing and network resources needed to complete the IoT data analysis of the tasks of TS-IoT applications and the execution times of tasks, the proposed task sizing technique executes the TS-IoT application tasks in available computing resources in the IoT environment, and measures the computing and network resources needed for the execution of TS-IoT application tasks and execution times of tasks in the available IoT devices, edge computers, and the cloud. This enables more realistic resource needs and execution times of TS-IoT application tasks to be gathered compared to estimations. The task sizing technique is executed whenever the application developer submits a new TS-IoT application to the TIDA platform or whenever there is a change in the underlying IoT environment, such as resources becoming disconnected and connected. In this task sizing technique, each task of the TS-IoT application is executed on every available unique IoT device, edge computer, and the cloud using sample IoT data submitted by the TS-IoT application developer. The

computing and network resources needed for the execution of each TS-IoT application task in the IoT environment are measured and recorded. These resource measurements are then used by the task distribution technique discussed in Section 5.2, and for training the machine learning (ML) model that is used in the task adaptation technique, which is discussed in Section 5.3.

The task sizing measurement data mainly captures the resource utilization metrics (e.g., CPU utilization and memory utilization metrics) and execution times of the tasks when the tasks are executed on available computing resources. As shown in Table 2, the raw task sizing measurement data consisted of 12 features. Unlike earlier task sizing solutions, e.g., [3], this task sizing technique employs an unsupervised machine learning technique called Principal Component Analysis (PCA) [45] to extract the most important features from the 12 features of the task sizing measurement data and then store these extracted features in the database. Feature extraction is the process of constructing a new set of features that is more informative and non-redundant than the raw measured data. Feature extraction techniques such as PCA can find correlations among the features in the task sizing measurement data. For example, consider the two features, the average CPU usage and maximum CPU usage, from the task sizing measurement data. These two features could be closely correlated; hence, keeping both of these features will not yield any additional benefit. There may also be many other features that are highly correlated with each other. Therefore, using a feature extraction technique such as PCA, a summarized version (which is more informative and non-redundant) of the original features can be derived from a combination of the original features. After applying the PCA technique on the task sizing measurement data, we were able to derive five new features that better describe the task sizing measurement data.

Table 2. Features of task sizing measurement data.

Features	Average CPU Usage, Max CPU Usage, Min CPU Usage, Average Memory Usage, Max Memory Usage, Min Memory Usage, Average Memory Usage Percentage, Max Memory Usage Percentage, Min Memory Usage Percentage, ResourceCPU (No of CPU cores in the computing resource), ResourceRAM (Total RAM in the computing resource), Data Size

The extracted features are then used to train the machine learning model used in the task adaptation technique. In addition, during the execution of the application, as preformed in the task sizing, we periodically measure the resource utilization and execution times. This monitored data is then used to periodically update the task sizing measurements. This provides continuously updated task sizing measurements and allows this technique to deal with variations in the volume and velocity of IoT data that lead to unexpected demand for computing resources. Table 2 shows the features of task sizing measurement data.

More in-depth details of this task sizing technique can be found in [3]. Most of the task sizing techniques in the literature include limited testbed implementations or rely on simulation tools to size the TS-IoT application tasks (i.e., to estimate the amount of resource required for the execution of tasks and the execution time of task on each resource). However, none of the existing task sizing techniques can effectively deal with (1) variations in the volume and velocity of IoT data that are common in IoT, and (2) the computing resource heterogeneity in the real IoT environment. Unlike other existing task sizing techniques, the technique proposed here measures the actual (i.e., not simulated) computing and network resources required to complete each task of the TS-IoT application at hand in the available computing resources in the IoT environment. Therefore, it provides more accurate estimations of the resource required for each TS-IoT application task and the execution time of each task per unique IoT device, edge computer, and cloud computing resource available. Section 5.2 discusses the task distribution technique that uses the

resource estimates (i.e., task sizing measurements) produced by the task sizing technique. The pseudocode of the task sizing technique is shown in Algorithm 1.

Algorithm 1: Task Sizing Technique

Input: TaskList, ResourceList
Output: MeasuredData
function SizeTask (TaskList, ResourceList)
1: **foreach** resource in ResourceList **do**
2: **foreach** task in TaskList **do**
3: Execute task and measure the computing and network resource usage and execution time
4: Return the measured data
5: **end foreach**
6: **end foreach**
7: **end function**

5.2. Task Distribution Technique

In this section, we propose a novel task distribution technique that follows a greedy heuristic approach to incrementally solve the task distribution problem and generate a time-bound satisfying task execution plan. As mentioned in Section 5.1, the greedy task distribution technique utilizes the task sizing measurements produced by the task sizing technique to construct a time-bound satisfying task execution plan for executing each TS-IoT application. The greedy task distribution technique can be re-executed as needed to construct different task execution plans in situations where certain computing resources are disconnected from or connected to the IoT environment after the greedy task distribution technique was previously executed for any TS-IoT application. The pseudocode of the proposed greedy task distribution technique is shown in Algorithm 2.

The technique takes the TaskList, ResourceList, MeasuredData, and TB_{App} as inputs. Then, for each task in the TaskList, the technique finds an eligible (i.e., has enough capacity to fulfill the resources required by the task) computing resource, which yields the lowest execution time for that task from a sorted resources map. To construct the sorted resources map for the first task in the TaskList, the technique uses only the computing resources that are closer to the IoT data source. To find such resources the algorithm uses the GetResourcesCloserToDataSource () function. Therefore, the first task of the application is always assigned to a computing resource that is closer to the data source, provided it has enough resource capacity (lines 4–5 in Algorithm 2). Moreover, to construct the sorted resources map for tasks that have predecessor tasks, the algorithm retrieves the tuples of the corresponding task from the measurement table and constructs a sorted resources map using the data in the tuples. The map consists of the resources and the corresponding execution time measured for that task. Furthermore, the map is sorted based on the measured execution times and we consider that one computing resource can host multiple tasks if it has enough resource capacity (lines 6–8 in Algorithm 2).

Once the sorted resources map is created, the technique iterates through each item in the sorted resources map until it finds an eligible computing resource. When the technique identifies an eligible computing resource, it first assigns that resource to the corresponding task via updating the task distribution map, then updates the available resources of the selected resource, updates the $Total\_Exec_{App}$ based on the estimated execution time, exits the while loop, and moves to the next task in the task list (lines10–19 in Algorithm 2). The technique iteratively determines eligible computing resources in a greedy manner (i.e., picks the resource that would yield the lowest execution time) for each task in the task list.

It should be noted that IoT can be volatile, because in the real world the IoT data volume/velocity and the available computing resources often vary. Although the task distribution technique can be executed several times to produce different task execution plans whenever computing resources are disconnected or connected, it is not capable of changing or adapting its task execution plan at hand to deal with possible time-bound violations

caused by varying IoT data volumes and available computing resources. Therefore, the task execution plans constructed by the task distribution technique may not achieve the time-bound requirements of the TS-IoT application due to such volatility. To overcome this drawback, we propose a novel task adaptation technique, which is discussed in Section 5.3.

Algorithm 2: Greedy Task Distribution Technique

Input: TaskList, ResourceList, MeasuredData, TB_{App}
Output: TaskExecutionPlan <task, resource>
function GreedyTaskDistribution (TaskList, ResourceList, MeasuredData, TB_{App})
01: *Initialize* TaskExecutionPlan <task, resource $\geq$ null, CanDistribute = true, SortedResourceMap <resource, execution_time>, $Total\_Exe_{App}$ = null;
02: **while** TaskList is not empty AND CanDistribute is true **do**
03: task = TaskList. GetItem ();
04: **if** IsFirstTask(task)
05: SortedResourceMap = GetResourcesCloserToDataSource ();
06: **else**
07: SortedResourceMap = MeasuredData.GetData (task);
08: **end if**
09: *Initialize* TaskPlaced = false, i = 0;
10: **while** i < SortedResourceMap. Count **AND** TaskPlaced is false **do**
11: resource = SortedResourceMap. GetItem(i);
12: **if** CheckResourceCapacity (task, resource)
13: TaskExecutionPlan. Add (task, resource);
14: UpdateAvailableResourceCapacities (resource, ResourceList);
15: $Total\_Exe_{App} \pm$ SortedResourceMap. GetValue (resource);
16: TaskPlaced = true;
17: **end if**
18: i++;
19: **end while**
20: **if** TaskPlaced is false **OR** $Total\_Exe_{App} > TB_{App}$
21: CanDistribute = false;
22: **end if**
23: **end while**
24: **if** $Total\_Exe_{App} \leq TB_{App}$ **AND** CanDistribute
25: return TaskExecutionPlan;
26: **else**
27: TaskExecutionPlan = AllocateAllTasksToCloud ();
28: return TaskExecutionPlan;
29: **end if**
end function

5.3. Task Adaptation Technique

The objective of the task adaptation technique is to dynamically adapt the task execution plan of the TS-IoT application to mitigate potential time-bound violations caused by unexpected changes in the IoT device observation volume and velocity, and computing resources becoming unavailable due to the volatility of the IoT environment. To achieve this, the task adaptation technique employs a variation of the XGBoost regression tree model [46] to periodically predict the application's total execution time assuming that the application continues to follow its current task execution plan. Next, the predicted total application execution is compared with the time-bound requirement of the TS-IoT application to assess whether the current task execution plan at hand can meet its time-bound requirement. If the predicted total application execution time is earlier than or meets the time-bound requirement of the TS-IoT application, then the current task execution plan remains unchanged. If the predicted total application execution time is later than the time-bound of the TS-IoT application, then an alternative task execution plan that can guarantee the time-bound requirements is selected from a set of alternative task execution plans (the creation of alternative execution plans is discussed further in Section 5.4). More specifically, when selecting an alternative task execution plan, the XGBoost model is used

to predict the total application execution times for each of the alternative task execution plans and pick the task execution plan that yields the lowest total application execution time. After selecting an alternative task execution plan, the tasks that were running will be stopped. Then, the TS-IoT application tasks will be redistributed according to this alternative task execution plan and start executing again.

The task adaptation technique continuously trains the XGBoost model in an online manner using the features extracted from the task sizing measurement data (as shown in Table 2 and discussed in Section 5.1). XGBoost is a tree-based ensemble machine learning algorithm, which was selected as the basis of the proposed task adaptation technique due to its fast convergence speed. Compared to the other performance prediction models, such as neural networks [47], decision tree-based models require less training data, less training time, and less parameter tuning. Moreover, the ensemble tree-based algorithms, which combine several decision trees, performed better when compared to single decision tree-based models [44].

5.4. Combining the Task Sizing, Distribution, and Adaptation Techniques to Meet TS-IoT Application Time-Bounds

In this section, we outline how the techniques we discussed in Sections 5.1–5.3 are combined in an integrated task management technique called the Dynamic Task Distribution and Adaptation (DTDA) technique. In summary, DTDA ensures that TS-IoT application meets their time-bounds as follows:

Step 1: Size the TS-IoT application's tasks in the available resources (i.e., all available IoT devices, edge computers, and cloud virtual machines) using the task sizing technique (line 4 in Algorithm 3).

Step 2: Train the XGBoost model according to the task adaptation technique using the features extracted from the task sizing measurement data (lines 5–8 in Algorithm 3).

Step 3: Construct all the possible task execution plans for the submitted TS-IoT application. For this purpose, we need to create all the different possibilities in distributing the tasks of the TS-IoT application to the available computing resources. Therefore, we generate all the combinatorial possibilities for distributing tasks of a given TS-IoT application to the available computing resources in the IoT environment. For example, consider an instance where a given TS-IoT application is comprised of three tasks (e.g., t_1, t_2, and t_3) and an IoT environment that has two available computing resources (e.g., cr_1 and cr_2). In this step, all the different possible distributions of the tasks t_1, t_2, and t_3 in the two computing resources cr_1 and cr_2 will be generated. As we discussed in Section 4, each different task distribution possibility in the IoT environment is considered as a task execution plan; thus, we identify all the possible task execution plans for the submitted TS-IoT application. Pre-computing all the possible task execution plans permits the task adaptation technique to quickly pick an alternative task execution plan from the list of all possible plans that have been pre-computed. This enables the dynamic adaptation technique to quickly pick an alternative plan that will meet the time-bound of the application (line 7 in Algorithm 3).

Step 4: Generate a time-bound satisfying task execution plan (from this point on we refer to this as the current task execution plan) using the greedy task distribution technique, which was discussed in Section 5.2 (line 9 in Algorithm 3).

Step 5: Distribute the tasks into the IoT devices, edge computers, and/or cloud-based virtual machines according to the task execution plan and start executing the application tasks (line 10 in Algorithm 3).

Step 6: Periodically measure the resource usage and execution times of the tasks to update the task sizing measurement data by (1) extracting the features from them, and (2) using the extracted features to retrain the ML model (lines 13–18 in Algorithm 3).

Step 7: Periodically predict the total application execution time of the task execution plan at hand using the XGBoost model (lines 20–22 in Algorithm 3).

Step 8: Assess whether the task execution plan at hand can guarantee the time-bound requirement of the TS-IoT application using the predicted total application execution time (line 23 in Algorithm 3).

Step 9: If a possible time-bound violation is detected, select an alternative task execution plan from the set of pre-computed task execution plans (lines 24–34 in Algorithm 3).

Step 10: Redistribute the TS-IoT application tasks according to the alternative task execution plan and restart the application execution (line 35 in Algorithm 3).

The pseudocode of the proposed DTDA technique is shown in Algorithm 3.

Algorithm 3: Dynamic Task Distribution and Adaptation Technique (DTDA)

Input: G_{App} , G_{Res}, TB_{App}

```
01:   Initialize TaskList = null, ResourceList = null; TaskExecutionPlanAtHand = null;
02:   TaskList = CreateTaskListFromAppGraph (G_App);
03:   ResourceList = CreateResourceListFromResourceGraph (G_Res);
04:   MeasuredData = SizeTask (TaskList, ResourceList, TB_App);//Use Task Sizing Technique
05:   ExtractedFeatures = ExtractFeatures (MeasuredData);//Extract features from measured data
06:   SaveExtractedFeatures (ExtractedFeatures);
07:   TaskExecutionPlanList = ComputeAllPossiblePlans (TaskList, ResourceList);
08:   TrainMLModel(ExtractedFeatures);//Train ML model using extracted features
09:   TaskExecutionPlanAtHand = GreedyTaskDistribution (TaskList, ResourceList, MeasuredData, TB_App);
10:   DistributeTasks (TaskExecutionPlanAtHand);
11:   NoChangeInIoTEnviornment = True;//This variable will change to false if any resource disconnects or connects to the IoT
12:   while NoChangeInIoTEnviornment is True do
13:       if PeriodicUpdate is true AND PeriodicPredict is false
14:           MeasuredData = MonitorAndMeasure ();
15:           ExtractedFeatures = ExtractFeatures (MeasuredData);
16:           UpdateValuesOfFeatureData (ExtractedFeatures);
17:           TrainMLModel (ExtractedFeatures);
18:       end if
19:       if PeriodicUpdate is false AND PeriodicPredict is true
20:           MeasuredData = MonitorAndMeasure ();
21:           ExtractedFeatures = ExtractFeatures (MeasuredData);
22:           Total_Exe_App = Predict (ExtractedFeatures, TaskExecutionPlanAtHand);
23:           IsTimeBoundViolation = AssessTimeBoundViolation (Total_Exe_App, TB_App);
24:           if IsTimeBoundViolation is true
25:               Initialize FoundNewPlan = False; i = 0;
26:               while TaskExecutionPlanList is not empty AND FoundNewPlan is false do
27:                   TaskExecutionPlan = TaskExecutionPlanList. GetItem(i);
28:                   PredictedTime = Predict (ExtractedFeatures, TaskExecutionPlan);
29:                   if PredictedTime =< TB_App
30:                       TaskExecutionPlanAtHand = TaskExecutionPlan;
31:                       FoundNewPlan = True;
32:                   end if
33:               end while
34:           end if
35:           DistributeTasks (TaskExecutionPlanAtHand);
36:       end if
37:   end while
```

6. TIDA Platform

To meet the time-bound requirements of TS-IoT applications we developed a novel time-sensitive IoT data analysis (TIDA) platform [48]. TIDA utilizes the combination of the task sizing, distribution, and adaptation techniques as described in Section 5 to distribute and execute the tasks of each TS-IoT application in the cloud, edge computers, and IoT devices. In this section, we discuss the design and implementation of the TIDA platform.

6.1. Architecture of the TIDA Platform

In this section, we discuss each component of the architecture. The architecture for the TIDA platform is illustrated in Figure 4.

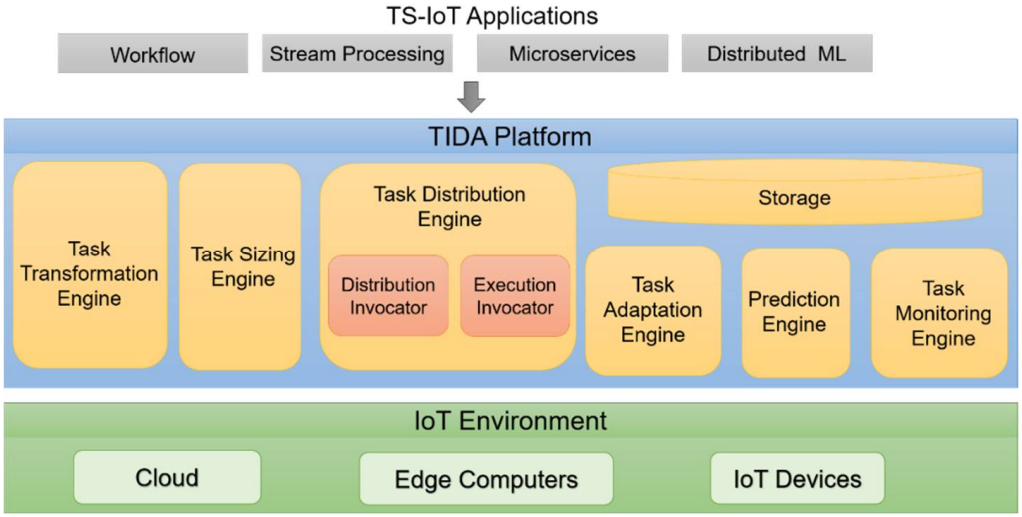

Figure 4. The architecture of the TIDA platform.

Task Transformation Engine: To execute any TS-IoT application irrespective of its underlying application model, we propose a transformation technique that transforms data analysis tasks of any TS-IoT application into a set of common executable units of the TIDA platform. We refer to these executable units as "actors". Each actor has the following characteristics: (1) represents a data analysis task of the TS-IoT application, (2) functionally equivalent to its corresponding data analysis task, and (3) independent of any application model. The transformation engine is responsible for this function of the platform, and it takes any TS-IoT application specification as an input and transforms its data analysis tasks into a set of functionally equivalent actors that can be executed by the platform.

Task Sizing Engine: This engine is responsible for implementing the task sizing technique that involves measuring computing and network resources for the execution of tasks of the TS-IoT application.

Task Distribution Engine: This implements the task distribution technique and is responsible for efficiently managing the distribution of tasks. The task distribution engine is comprised of two modules. These are the distribution planner module and distribution invocator module. Distribution planners can accommodate different task distribution techniques. Task distribution techniques (such as the dynamic task distribution and adaptation technique discussed in Section 5) generate task execution plans. Then, these task execution plans are sent to the distribution invocator, which then distributes the tasks to the corresponding resources according to the plan and invokes their executions.

Monitoring Engine: This engine periodically monitors the execution landscape in terms of resource utilization (CPU usage and RAM usage) and execution progress of tasks. It is comprised of a measuring module and a metric module. The measuring module is responsible for periodically collecting the resource utilization and task execution progress and forwarding this information to the metrics module. The metrics module then uses this information to compute numerous metrics, which is then forwarded to the reasoner engine. The metrics [33] currently supported by the TIDA platform are as follows:

- Total Application execution time;
- Total data communication time during the application execution;
- Total data processing time of the application. (i.e., time taken to analyze the IoT data);
- Data processing time of each data analysis task;
- Time-bound violation ratio of the TS-IoT application.

Prediction Engine: This engine is responsible for periodically predicting the application's total execution using monitored data. To realize this, the prediction engine employs an XGBoost-based machine learning model to periodically predict the application's total execution time using features extracted from the monitored data. Using the predicted

application's total execution time, the prediction engine can assess whether the tasks at hand can meet the time-bound requirements with the task execution plan at hand, as discussed in Section 5.3. If it cannot meet the time-bound requirements, the task adaptation engine is triggered to make the necessary adaptations to the task execution plans to deal with predicted future time-bound violations.

Task Adaptation Engine: This engine implements the task adaptation technique and is responsible for making the necessary dynamic adaptations to the task execution plans whenever the prediction engine predicts a possible future time-bound violation. To achieve this, the task adaptation engine selects an alternative task execution plan that can guarantee the time-bound requirements from a set of pre-computed task execution plans. Finally, it triggers the task distribution engine to redistribute the tasks according to the alternative task execution plan.

6.2. Implementation of the TIDA Platform

A proof of concept implementation of the TIDA platform [48] was implemented using Microsoft's Orleans Actor framework [49]. Orleans actors are developed to scale elastically, and they can run on any operating system that has .NET core installed. Therefore, we decided to implement the TIDA platform's underlying executable units as Orleans actors. This enabled us to develop a highly scalable and efficient task management system that led to the development of a proof-of-concept task distribution engine. Furthermore, we implemented the discussed DTDA technique as part of the task distribution engine. The transformation engine was implemented as a .NET CORE class library. For the proof of concept implementation of this research, we developed a wrapper that can be used to read a workflow specification file modeled using the camunda [50] workflow modeler. The monitoring engine was implemented as an Orleans start-up service, which is activated when TIDA is up and running. The monitoring engine periodically monitors the execution landscape in terms of resource utilization and execution progress of tasks. Moreover, its metric module then computes the metrics that are mentioned in Section 6.1. The reasoning engine was implemented as a Flask Python service. The machine learning model was implemented using the Python XGBoost library. Other engines interact with the reasoner engine via REST API calls. The task adaptation engine was implemented as another .NET CORE class library. A PostgreSQL [51] relational database was used as our storage provider. This store measured data, feature data, application-specific data, and information of the resources such as the health of each resource.

7. Experimental Evaluation of TIDA

7.1. Time-Sensitive IoT Application, Dataset and Experimental Setup

To evaluate the TIDA platform and its techniques, we implemented the passenger counting TS-IoT application discussed in Section 3. The application was modeled as a workflow application using the camunda workflow modeler. As we discussed in Section 3, the passenger counting TS-IoT application is comprised of three main data analysis tasks: (1) pre-processing task, (2) classification task, and (3) counting task. The pre-processing task is mainly responsible for pre-processing the passenger onboarding video data by converting them to gray-scale from RGB. The classification task uses a Haar-cascade machine learning classifier to classify passengers as entering or exiting, and to maintain counts for entering/exiting. The counting task simply aggregates the exit and enter counts received from the classification task to compute the total occupancy in the bus. We used the OpenCV library for pre-processing and classification tasks, and implemented the application using C#. Figure 5 shows the application graph of the passenger counting IoT application.

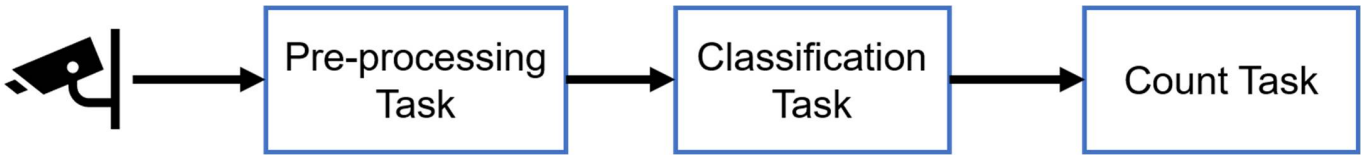

Figure 5. Passenger counting TS-IoT application tasks.

The dataset used for experimental evaluation was obtained during a project carried out in Sydney, Australia [52], using a Orbbec Persee device, which provides internal computing and storage resources consisting of a Quad-core Cortex A17 processor (which has a processing speed of 1.8 GHz), 2 GB RAM, and 8 GB internal storage. We used three passenger onboarding video files with three different sizes (12.5, 25, and 37.5 MB) for the evaluations. Each of these video files has a resolution of 640 × 480 and 30 frames per second (FPS). Figure 6 shows a sample of the passenger onboarding data.

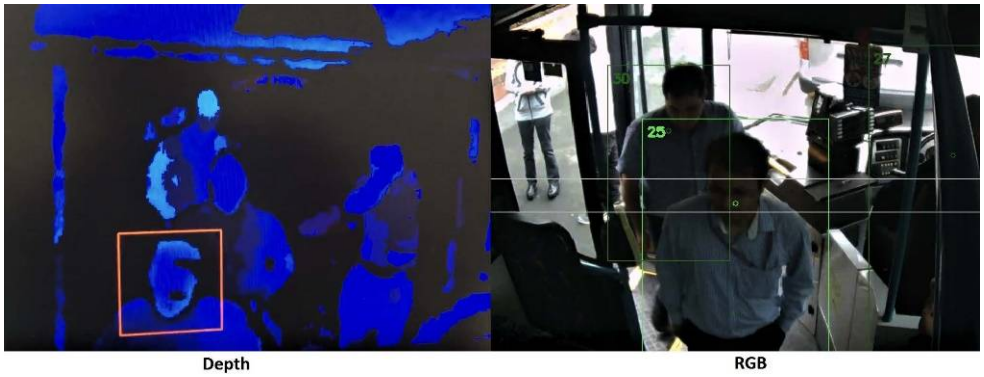

Figure 6. Passenger onboarding video.

The experiments were conducted in a cloud-based environment. For this purpose, we created a testbed in the cloud using the NECTAR research cloud [53]. The computing resources of the experimental testbed were categorized into four clusters of cloud virtual machines and each cluster consisted of three cloud virtual machines. To emulate edge computers and IoT devices, we created eight cloud virtual machines (four virtual machines that emulate edge computers and four virtual machines that emulate IoT devices) with similar system configurations of real-world edge computers and IoT devices. For this purpose, we considered the system configurations of the Cisco 807 industrial services router for edge computers and Orbbec Persee camera's system configurations for the IoT device. In summary, the experimental setup comprised the following computing resources: Cluster 1 consisted of three virtual machines that emulated IoT devices; Cluster 2 consisted of three virtual machines that emulated edge computers; Cluster 3 consisted of three cloud virtual machines; and Cluster 4 consisted of three virtual machines that emulated an IoT device, edge computer, and a cloud virtual machine. In this evaluation, we simulated the IoT data communication through the network because all the computing resources were created in the cloud. To compute the total data communication time, the sample video files and the pre-processed video files (i.e., the data output of the pre-processing task) were transferred to the cloud virtual machines over the Internet (bandwidth 4 Mbps) from a local computer and the amount of time taken to transmit was measured. Figure 7 shows the four computing resources clusters and the computing resources of each cluster with their corresponding system configurations.

Table 3 illustrates the overall system configurations of the computing resources in the experimental setup.

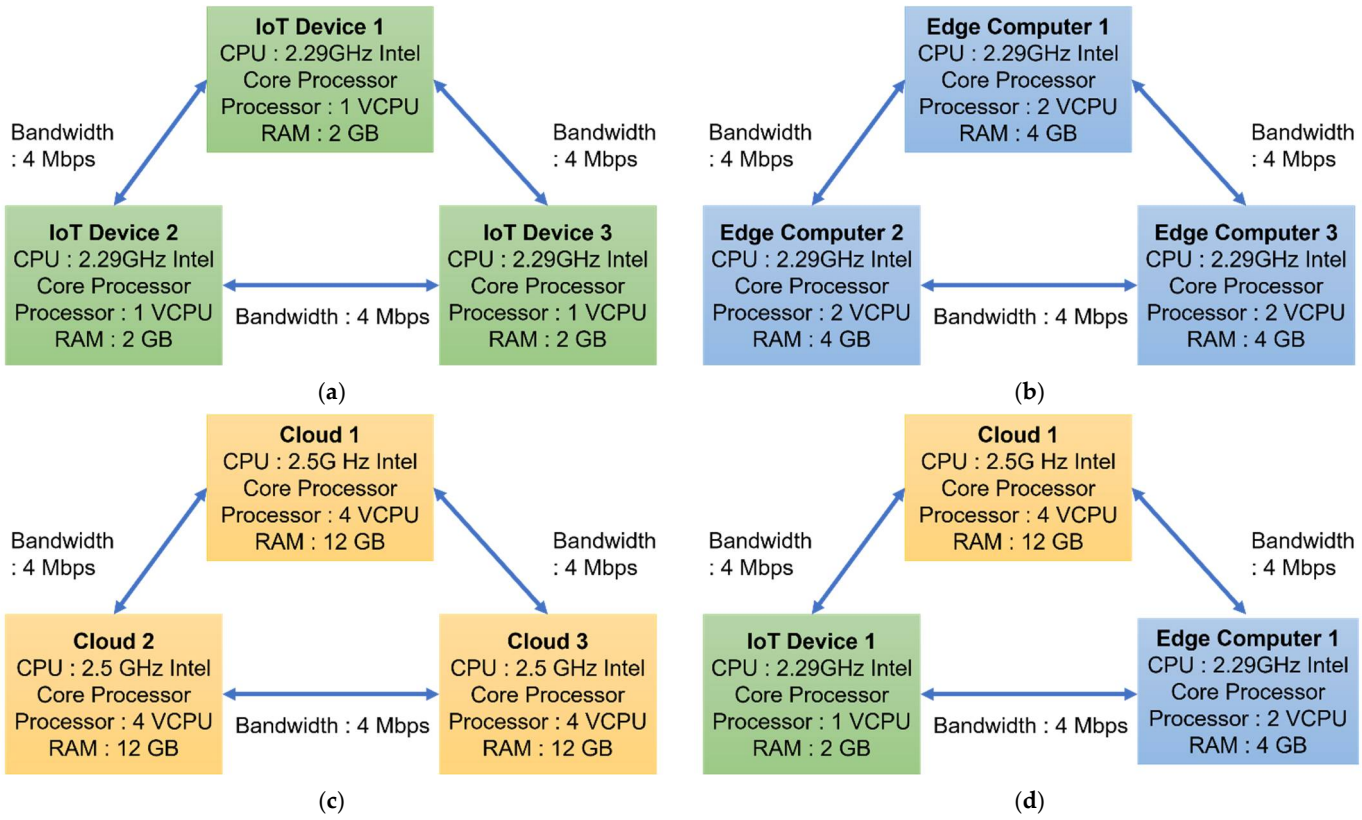

Figure 7. Experimental setup with corresponding system configurations. (**a**) Cluster 1 (IoT devices). (**b**) Cluster 2 (edge computers). (**c**) Cluster 3 (cloud servers). (**d**) Cluster 4 (IoT devices, edge computer, and cloud server).

Table 3. System configurations of the computing resources.

Computing Resources Type	CPU	RAM	Number of Computing Resources
Cloud virtual machine	2.5 GHz Intel Core Processor 4 VCPUs	12 GB	4
Edge computer	2.29 GHz Intel Core Processor 2 VCPUs	4 GB	4
IoT device	2.29 GHz Intel Core Processor 1 VCPUs	2 GB	4

7.2. Experimental Evaluation Methodology and Evaluation Metrics

We compared the performance of the proposed task management technique (DTDA) with three state-of-the-art task management techniques, namely binpack [54], random [55], and greedy [3]. We implemented the abovementioned task management techniques in the TIDA platform to fairly compare the performance of the aforementioned techniques under different computing resource clusters and different quantities of data (passenger onboarding video size). Table 4 shows the task management techniques we used for this experimental evaluation.

Table 4. Task management techniques used for experimental evaluation.

Task Management Technique	Description	Capabilities of Task Management Technique		
		Task Sizing	Task Distribution	Task Adaptation
Binpack technique [54]	Binpack task management technique manages the TS-IoT application tasks by distributing them to computing resources that have the least available amount of CPU and memory. This minimizes the number of resources in use and is more cost-efficient. However, this technique does not size the tasks before the distribution and it is not capable of adapting its task distribution during the runtime.	No	Yes	No
Random technique [55]	Random task management technique manages the TS-IoT application tasks by distributing tasks to computing resources randomly and executing them. However, this technique does not size the tasks before distribution and it is not capable of adapting its task distribution.	No	Yes	No
Greedy technique [3]	Greedy technique sizes the tasks and then uses that task sizing information to generate a time-bound satisfying task execution plan and distribute the tasks according to that plan. However, it cannot adapt its task execution plans during runtime.	Yes	Yes	No
DTDA technique	DTDA technique is the task management technique proposed in this paper and described in Section 5.	Yes	Yes	Yes

We continuously executed the passenger counting IoT application tasks for 1 h in each computing resources cluster (as discussed in Section 7.1) using the task management techniques presented in Table 4. During the execution, at regular intervals (i.e., every 2 min), we generated IoT data processing requests (i.e., a request sent to the TS-IoT application to process the IoT data generated by IoT devices along with the generated data) with varying passenger onboarding video data sizes to be processed by the passenger counting IoT application tasks. To vary the data sizes, we used the following passenger onboarding video files—12.5, 25.0, and 37.5 MB. Generating IoT data processing requests at regular intervals with varying data sizes enabled us to replicate how a real passenger counting IoT application deployed on buses processes different quantities of passenger onboarding data collected at bus stops, as discussed in Section 3. In this experimental evaluation, we considered the following two evaluation scenarios:

1. Evaluate the impact of task management techniques on the total application execution time of the passenger counting IoT application.

The main objective of this evaluation scenario was to evaluate the impact of the task management techniques on the total application execution time (i.e., the end-to-end time taken to complete a single IoT data processing request) of the passenger counting IoT application when it is executed in different computing resources clusters with varying data sizes. To realize this, during the experimentation, for each IoT data processing request, we measured the data processing time and data communication delay incurred by the passenger counting IoT application for completing the IoT data processing request. We denote the data processing time as $Data\_Processing\_Time_{App}$ and data communication

delay as $Data\_Comm\_Delay_{App}$. Then, we used these two metrics to compute the total application execution (which is denoted as $Total\_Exe_{App}$) for completing an IoT data processing request as follows:

$$Total\_Exe_{App} = Data\_Processing\_Time_{App} + Data\_Comm\_Delay_{App} \quad (9)$$

It should be noted that, in general, there can be more attributes that contribute to the $Total\_Exe_{App}$ of a TS-IoT application. However, in this experiment, we considered $Data\_Processing\_Time_{App}$ and $Data\_Comm\_Delay_{App}$ as the two main attributes that make up the $Total\_Exe_{App}$. The results of this experimental evaluation scenario are presented in Section 7.3.1.

2. Evaluate the impact of task management techniques on the time-bound violation of the passenger counting IoT application.

The main objective of this evaluation scenario was to assess how well the task management techniques supported the time-bound requirements of the passenger counting IoT application when it is executed in different computing resources clusters with varying data quantities. To realize this, we first set the time-bound requirement of the passenger counting IoT application to 60 s, which is the average time to travel between two bus stops. For each IoT data processing request, we then observed whether the passenger counting IoT application's $Total\_Exe_{App}$ exceeds the time-bound requirement (i.e., 60 s). A time-bound violation happens when $Total\_Exe_{App}$ of the passenger counting IoT application exceeds the time-bound requirement when processing an IoT data processing request. We used an evaluation metric called time-bound violation counts to capture the total number of time-bound violations that occur during the experimentation. We denote the time-bound violation count as V_Count. Moreover, to compare the impact of task management techniques on time-bound violations, we computed an evaluation metric called time-bound violation ratio, which we denote as V_Ratio. Time-bound violation ratio (V_Ratio) is the time-bound violation count over the total number of IoT data processing requests completed during the experimentation. We denote the total number of IoT data processing requests completed as $Total\_Requests$ and V_Ratio is computed as follows:

$$V\_Ratio = \frac{V\_Count}{Total\_Requests} \quad (10)$$

The range for the V_Ratio is from 0 to 1, where 0 is the best-case scenario when each IoT data processing request is completed within the time-bound requirement of the TS-IoT application, whereas 1 is the worst-case scenario where the passenger counting IoT application has failed to meet the time-bound requirement for any IoT data processing request. The results of this experimental evaluation scenario are presented in Sections 7.3.2 and 7.3.3. Table 5 provides a summary of the evaluation metrics discussed in this section.

Table 5. Evaluation metrics.

Evaluation Metric Notation	Definition of the Metric
$Total\_Exe_{App}$	Application execution time
$Data\_Processing\_Time_{App}$	Data processing time of the application
$Data\_Comm\_Delay_{App}$	Data communication delay of the application
V_Count	Time-bound violation count
V_Ratio	Time-bound violation ratio

7.3. Experimental Evaluation Results and Discussion

7.3.1. Impact of Task Management Techniques on the Total Application Execution Time under Different Computing Resources and Data Sizes

As discussed earlier, for this experimental evaluation scenario, we considered the total application execution time as the summation of data processing time and the data communication delay of the passenger counting IoT application. For ease of comparison, we took the average values of the total application execution time, data processing time,

and data communication delay because these metrics were measured/computed per IoT data processing request during the experimentations. Therefore, Figure 8 illustrates the comparison of the average total application time, average data processing time, and average data communication delay of the passenger counting IoT application under each task management technique when executed in four different computing resource clusters with varying data sizes.

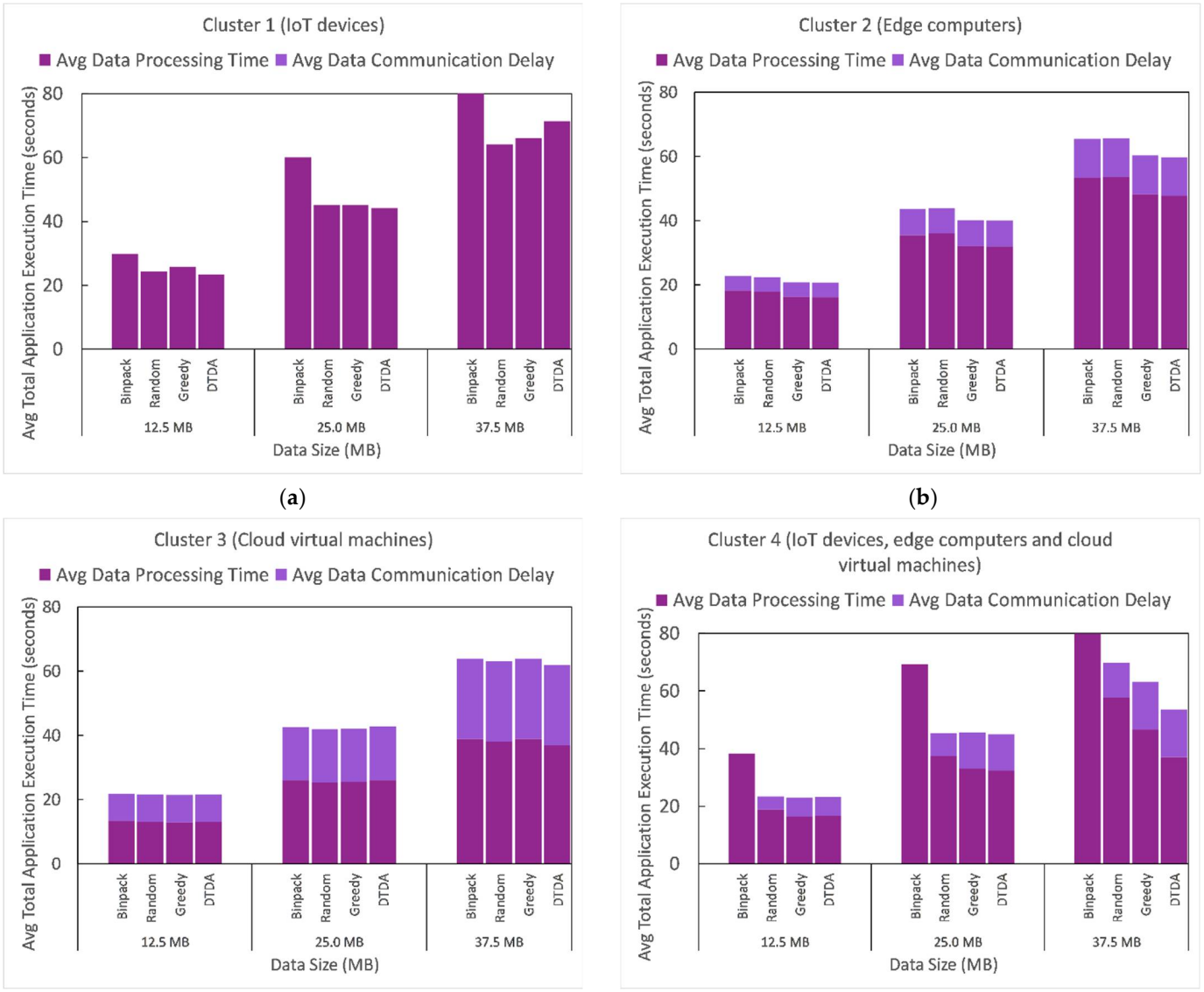

Figure 8. Comparison of the average data processing time, communication delay, and total application execution time of the passenger counting IoT application under each task management technique when executed in four different computing resources clusters with varying data sizes. (**a**) Cluster 1 (IoT devices). (**b**) Cluster 2 (edge computers). (**c**) Cluster 3 (cloud virtual machines). (**d**) Cluster 4 (IoT devices, edge computer and cloud virtual machine).

By analyzing the results presented in Figure 8, we can see that as the size of the data increases the average total application execution time also increases. Results presented in Figure 8a show that executing the passenger counting IoT application in cluster 1 did not incur any data communication delay, but this led to higher average data processing time for each task management technique compared to that of other clusters. This is mainly because cluster 1 consisted entirely of IoT devices, which (1) have limited computing resources and (2) do not involve any communication delay. Furthermore, we note that the impact of the

task management techniques on average total application execution time, average data processing time, and average data processing delay in cluster 1 varied inconsistently with the size of data. More specifically, the binpack technique yielded the highest average data processing time in cluster 1 for all the three data sizes, whereas the random, greedy, and DTDA techniques resulted in similar average total data processing time when the size of the data was 12.5 and 25 MB. However, when the size of the data increased to 37.5 MB, the DTDA technique yielded the second-highest average data processing time (71.41 s).

By comparison, results presented in cluster 3 show that the average total application execution time, average data processing time, and average data communication delay of the passenger counting IoT application were not significantly impacted by the task management technique in use. In addition, we can observe that cluster 3 recorded the highest average data communication delay and lowest data processing time for each of the three data sizes compared to the other clusters. This is mainly because cluster 3 consisted of three cloud virtual machines, which (1) have higher computing resources and (2) involve high network delays.

Results presented in cluster 2 show that both greedy and DTDA techniques performed equally well for each of the three data sizes and reduced the average data processing time compared to the binpack and random techniques as follows: by 11.11% when data size was 12.5 and 25 MB, and by 9.43% data size was 37.5 MB.

Results presented in Figure 8d illustrate that the binpack technique yielded the highest average data processing time for each of the three data sizes in cluster 4. Noticeably, the binpack technique did not involve any data communication delay. As we discussed in Section 7.2, it manages the TS-IoT application tasks by distributing them to computing resources that have the least available CPU and memory. Therefore, it is evident that, in this case, the binpack technique distributed all the tasks of the passenger counting IoT application to the IoT device; thus, no communication delay is involved. By comparison, the random, greedy, and DTDA techniques recorded similar average total application execution times when the size of the data was 12.5 and 25.0 MB. However, when the size of the data increased to 35.0 MB, the DTDA technique improved the average data processing time compared to the greedy technique by 20%, the random technique by 36%, and the binpack technique by 53%.

7.3.2. Time-Bound Violation Ratio of Task Management Techniques under Different Computing Resources

The comparison of time-bound violation ratio results in Figure 9 shows that the time-bound violation ratios for cluster 1 and cluster 3 are similar (i.e., 0.34) for each of the four task management techniques, with the exception of cluster 1, where binpack resulted in a 0.5 time-bound violation ratio. This is mainly due to the computing resources in cluster 1 and cluster 3. Cluster 1 utilizes three IoT devices, which have limited resources, whereas cluster 3 utilizes three cloud virtual machines, which have more resources compared to cluster 1. When a TS-IoT application is entirely executed in a resource-constrained environment, its tasks may overload the most constrained resources, causing a time-bound violation. In addition, when a TS-IoT application is executed entirely on cloud virtual machines, the significant communication delays involved in sending IoT data to the cloud contribute to higher time-bound violation ratios. In these cases, the task management technique employed cannot prevent the time-bound violation of the passenger counting application.

However, we can note that the time-bound violation ratios significantly varied when the passenger counting application was executed in cluster 2 and cluster 4. The binpack technique manages TS-IoT applications by distributing the tasks of the application to computing resources that have the least available amount of CPU and memory. Although this minimizes the number of computing resources in use and is more cost-efficient, this leads to overloading of the computing resources that are processing the tasks. Thus, this can increase the data processing time of the tasks, which will lead to time-bound violations. As we can see from the results in clusters 2 and 4, the binpack technique

recorded the highest time-bound violation ratios. More specifically, it recorded time-bound violations ratio of 0.34 and 0.67 in clusters 2 and 4, respectively. By comparison, the random technique manages the TS-IoT applications by randomly distributing the tasks of the application to computing resources. Therefore, the results obtained by executing the application under the random task management technique in clusters 2 and 4 show average results. In particular, this approach recorded a time-bound violation ratio of 0.34 in both clusters 2 and 4. The greedy technique generates time-bound satisfying task execution plans by considering the IoT environment, the time-bound requirements of the application, and the computing resource demands of the application tasks (obtained by the task sizing technique). Therefore, compared to the binpack and random task management techniques, the greedy technique recorded lower time-bound violations by distributing and executing the tasks according to a time-bound satisfying task execution plan that keeps the passenger counting application's total execution time within the application time-bound on most occasions. However, the greedy technique is not capable of adapting its task execution plans to cope with the dynamic changes occurring in the IoT environment, such as varying IoT data sizes and available computing resources; thus, when the application is executed in clusters 2 and 4 under the greedy technique, we still observe time-bound violation ratios of 0.25 and 0.23, respectively. On the contrary, the proposed DTDA technique of TIDA is capable of adapting the task execution plans of the running application to deal with possible time-bound violations; thus, we can observe that the DTDA technique recorded the lowest time-bound violation ratios for the passenger counting application when it was executed in clusters 2 and 4. More precisely, DTDA recorded a time-bound violation ratio of 0.08 in cluster 2 and 0.13 in cluster 4. Compared to the greedy technique [3], the novel DTDA technique presented in this paper improves the time-bound violation by 68% in cluster 2 and 42.8% in cluster 4.

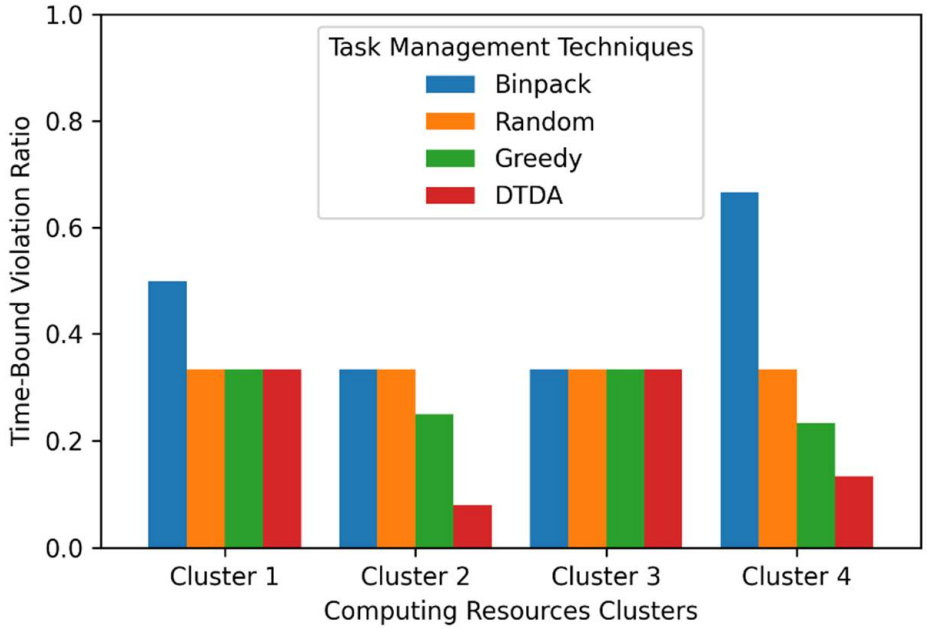

Figure 9. Comparison of time-bound requirement violation ratios under different computing resources clusters.

7.3.3. Time-Bound Violation Ratio of Task Management Techniques under Different Passenger Onboarding Video Data Sizes

The comparison of time-bound violation ratio results in Figure 10 shows that when the size of the data increases, the time-bound violation ratio also increases. More specifically, we can observe that, in all clusters, when the size of the passenger onboarding video was 12.5 and 25.0 MB, random, greedy and DTDA techniques recorded zero time-bound violations.

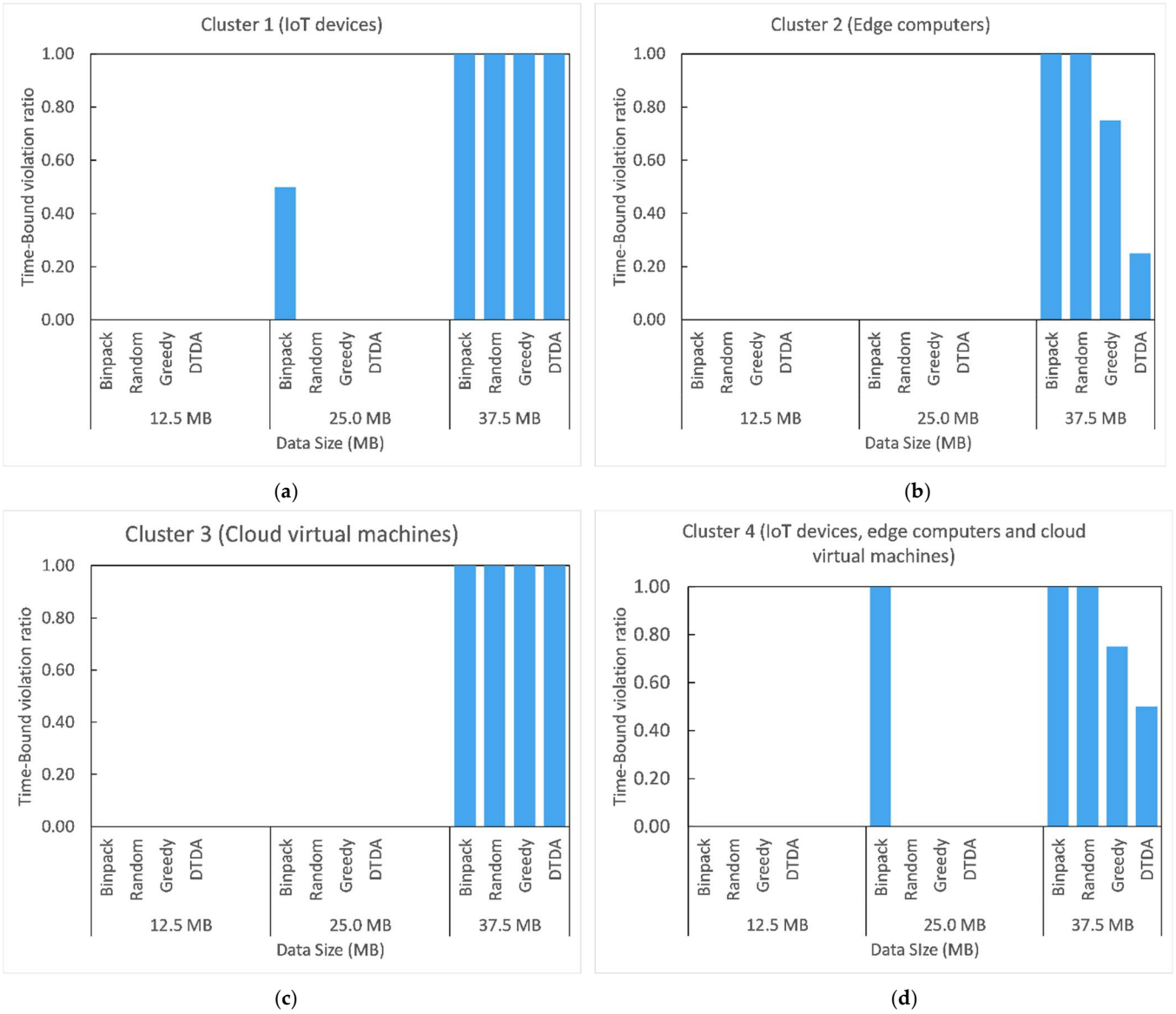

Figure 10. Comparison of the time-bound violation ratio of task management techniques under different computing resource clusters and varying data sizes. (**a**) Cluster 1 (IoT devices). (**b**) Cluster 2 (edge computers). (**c**) Cluster 3 (cloud virtual machines). (**d**) Cluster 4 (IoT devices, edge computer and cloud virtual machines).

However, we can observe that, in clusters 1 and 4, the binpack technique recorded time-bound violation ratios of 0.50 and 1.00 when the size of the passenger onboarding video reached 25.0 MB. By comparison, we note that when the size of the passenger onboarding video increased to 37.5 MB, all the task management techniques failed to fulfil the time-bound requirements of the passenger counting IoT application in clusters 1 and 3. As discussed in the previous section, this is mainly due to the limited resources found in IoT devices and the high communication delays involved in sending the passenger onboarding data to the cloud. Figure 10b–d indicates that both binpack and random techniques were unable to support the time-bound requirements of the passenger counting IoT application when the size of the passenger onboarding data reached 37.5 MB. However, for the same size of the passenger onboarding data, the DTDA technique maintained the time-bound violation ratio within 0.25 in cluster 2 and 0.50 in cluster 4. These results demonstrate that, in cluster 2, the DTDA technique improved the time-bound violation ratio compared to the greedy technique by 67%, and compared to the random and binpack techniques by 75%. In cluster 4, the DTDA technique reduced the time-bound violation compared to

the greedy technique by 33.33%, and compared to the random and binpack techniques by 50%. Finally, these results validate that the DTDA technique dynamically adapts the task execution plans to deal with the increased size of the passenger onboarding data, and this enables it to reduce the time-bound violations of the passenger counting IoT application compared to other task management techniques when the size of the data increases.

In summary, the evaluation results showed that when the passenger counting application is executed in cluster 1 (the cluster that consists of only IoT devices) and cluster 3 (the cluster that consists of only cloud virtual machines), regardless of the task management technique in use, on most occasions the passenger counting application failed to meet its time-bound requirement. The communication delay involved with sending data to the cloud and the resource-constrained nature of IoT devices may be the main reasons behind this behavior. Furthermore, the results showed that DTDA, which is our proposed task management technique of the TIDA platform, successfully fulfilled the time-bound requirements of the passenger counting IoT application with varying sizes of passenger onboarding data when the passenger counting IoT application was executed in cluster 3 (the cluster that consists of only edge computers) and cluster 4 (the cluster that consists of IoT devices, edge computers, and cloud virtual machines), whereas the three other task management techniques failed to meet the time-bound requirements. Moreover, the evaluation results showed that the DTDA technique improved the time-bound violation ratios by 42.84% and the data processing time by 20.42% compared to the greedy technique presented in [3] when the application was executed in cluster 4. Finally, the evaluation results demonstrated that the TIDA platform that includes the DTDA technique periodically predicts possible future time-bound violations and adapts its task execution plans to mitigate possible time-bound violations. This enabled the platform to meet the time-bound requirements of the passenger counting IoT application more often than the other task management techniques.

8. Conclusions and Future Work

In this paper, we extended our previous research presented in [3] and proposed a complete solution, called TIDA, for meeting the time-bound requirements of TS-IoT applications. The proposed TIDA platform uses a novel task management technique, called DTDA, which combines three novel techniques: task sizing, task distribution, and task adaptation. The task sizing technique measures the computing and network resources required to complete each TS-IoT application task in each available computing resource in the IoT environment, whereas the task distribution technique utilizes the measurements of task sizing to create time-bound satisfying task execution plans to distribute and execute TS-IoT application tasks in the IoT environment. Finally, the task adaptation technique utilizes a machine learning model to accurately and periodically predict possible time-bound violations and, in the case of a time-bound violation, dynamically adapts the task distribution by redistributing tasks according to an alternative task execution plan.

We described a proof-of-concept implementation of the TIDA platform that implements the above techniques using Microsoft's Orleans framework. We evaluated the TIDA by developing a passenger counting IoT application, executing the application in a cloud-based testbed under four TS-IoT application management techniques, and assessing how well each of these techniques enables the application to meet its time-bound requirements. The results showed that the dynamic task distribution and adaptation (DTDA) technique of TIDA (discussed in Section 5), on average, improves the time-bound violation ratio by 43.34%, compared to the greedy technique, which is the base-line TS-task management technique of the TIDA platform. Moreover, the evaluation demonstrated the TIDA's ability to adapt to the varying volume of IoT data by dynamically adapting the task execution plans to deal with possible time-bound violations.

Although the current implementation of the DTDA technique is capable of dynamically adapting its task execution plans, during the adaptation step, the technique iterates through a set of pre-computed task execution plans to select an alternative time-bound

satisfying plan, and then redistributes all the tasks into the computing resources according to the new plan. This step is not efficient and scalable because it may cause a significant delay when the number of tasks and the computing resources increase. Moreover, in this research, for the evaluation, we considered only one TS-IoT application. However, in the real world, there can be multiple IoT applications competing for the computing and networking resources in the IoT environment, and resources often connect and disconnect from the IoT environment. Therefore, in our future work, we aim to: (1) improve the DTDA technique to be more efficient and scalable; (2) conduct more experiments in a volatile IoT environment with multiple IoT applications; (3) compare the TIDA platform's ability to meet time-bound requirements with existing solutions; and (4) improve the existing machine learning technique by incorporating federated machine learning techniques [56]. Moreover, we plan to explore and incorporate (1) IoT services search and discovery techniques [57], (2) digital twin capabilities [58], and (3) contextualization [59] and approximation approaches [60] into the TIDA platform to make it a comprehensive IoT solution.

Author Contributions: Conceptualization, H.K., D.G. and P.P.J.; Formal analysis, H.K.; Methodology, H.K.; Software, A.Y.; Supervision, D.G., P.P.J. and A.Y.; Writing—original draft, H.K.; Writing—review & editing, D.G. and P.P.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2347–2376. [CrossRef]
2. Georgakopoulos, D.; Jayaraman, P.P. Internet of things: From internet scale sensing to smart services. *Computing* **2016**, *98*, 1041–1058. [CrossRef]
3. Korala, H.; Georgakopoulos, D.; Jayaraman, P.P.; Yavari, A. A Time-Sensitive IoT Data Analysis Framework. In Proceedings of the 54th Hawaii International Conference on System Sciences, Koloa, HI, USA, 5 January 2021; pp. 7185–7194.
4. Chen, J.; Chen, Y.; Du, X.; Li, C.; Lu, J.; Zhao, S.; Zhou, X. Big data challenge: A data management perspective. *Front. Comput. Sci.* **2013**, *7*, 157–164. [CrossRef]
5. Koga, Y.; Miyazaki, H.; Shibasaki, R. A CNN-Based Method of Vehicle Detection from Aerial Images Using Hard Example Mining. *Remote Sens.* **2018**, *10*, 124. [CrossRef]
6. Zhao, Q.; Zhang, B.; Lyu, S.; Zhang, H.; Sun, D.; Li, G.; Feng, W. A CNN-SIFT Hybrid Pedestrian Navigation Method Based on First-Person Vision. *Remote Sens.* **2018**, *10*, 1229. [CrossRef]
7. Zhou, H.; Taal, A.; Koulouzis, S.; Wang, J.; Hu, Y.; Suciu, G.; Poenaru, V.; De Laat, C.; Zhao, Z. Dynamic Real-Time Infrastructure Planning and Deployment for Disaster Early Warning Systems. In *Lecture Notes in Computer Science, Proceedings of the International Conference on Computational Science, Wuxi, China, 11–13 June 2018*; Springer: Berlin, Germany, 2018; pp. 644–654.
8. Naha, R.K.; Garg, S.; Georgakopoulos, D.; Jayaraman, P.P.; Gao, L.; Xiang, Y.; Ranjan, R. Fog Computing: Survey of Trends, Architectures, Requirements, and Research Directions. *IEEE Access* **2018**, *6*, 47980–48009. [CrossRef]
9. Georgakopoulos, D.; Jayaraman, P.P.; Fazia, M.; Villari, M.; Ranjan, R. Internet of Things and Edge Cloud Computing Roadmap for Manufacturing. *IEEE Cloud Comput.* **2016**, *3*, 66–73. [CrossRef]
10. Garg, S.; Forbes-Smith, N.; Hilton, J.; Prakash, M. SparkCloud: A Cloud-Based Elastic Bushfire Simulation Service. *Remote Sens.* **2018**, *10*, 74. [CrossRef]
11. Jayaraman, P.P.; Perera, C.; Georgakopoulos, D.; Dustdar, S.; Thakker, D.; Ranjan, R. Analytics-as-a-service in a multi-cloud environment through semantically-enabled hierarchical data processing. *Softw. Pract. Exp.* **2016**, *47*, 1139–1156. [CrossRef]
12. Naha, R.K.; Garg, S.; Chan, A.; Battula, S.K. Deadline-based dynamic resource allocation and provisioning algorithms in Fog-Cloud environment. *Futur. Gener. Comput. Syst.* **2020**, *104*, 131–141. [CrossRef]
13. Aazam, M.; Huh, E.-N. Dynamic resource provisioning through fog micro datacenter. In Proceedings of the 2015 IEEE International Conference on Pervasive Computing and Communication Workshops, St. Louis, MO, USA, 23–27 March 2015; pp. 105–110.

14. Aazam, M.; St-Hilaire, M.; Lung, C.-H.; Lambadaris, I. MeFoRE: QoE Based Resource Estimation at Fog to Enhance QoS in IoT. In Proceedings of the 23rd International Conference on Telecommunications (ICT), Thessaloniki, Greece, 16–18 May 2016; pp. 1–5. [CrossRef]
15. Zeng, X.; Garg, S.K.; Strazdins, P.; Jayaraman, P.P.; Georgakopoulos, D.; Ranjan, R. IOTSim: A simulator for analysing IoT applications. *J. Syst. Arch.* **2017**, *72*, 93–107. [CrossRef]
16. Arlitt, M.; Marwah, M.; Bellala, G.; Shah, A.; Healey, J.; Vandiver, B. IoTAbench: An Internet of Things Analytics Benchmark. In Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, Austin, TX, USA, 31 January 2015; pp. 133–144. [CrossRef]
17. Hong, H.-J.; Tsai, P.-H.; Hsu, C.-H. Dynamic module deployment in a fog computing platform. In Proceedings of the 2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS), Kanazawa, Japan, 5–7 October 2016; pp. 1–6.
18. Yousefpour, A.; Patil, A.; Ishigaki, G.; Kim, I.; Wang, X.; Cankaya, H.C.; Zhang, Q.; Xie, W.; Jue, J.P. FOGPLAN: A Lightweight QoS-Aware Dynamic Fog Service Provisioning Framework. *IEEE Internet Things J.* **2019**, *6*, 5080–5096. [CrossRef]
19. Skarlat, O.; Nardelli, M.; Schulte, S.; Dustdar, S. Towards QoS-Aware Fog Service Placement. In Proceedings of the 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC), Madrid, Spain, 14–15 May 2017; pp. 89–96.
20. Li, L.; Li, S.; Zhao, S. QoS-Aware Scheduling of Services-Oriented Internet of Things. *IEEE Trans. Ind. Inform.* **2014**, *10*, 1497–1505. [CrossRef]
21. Skarlat, O.; Karagiannis, V.; Rausch, T.; Bachmann, K.; Schulte, S. A Framework for Optimization, Service Placement, and Runtime Operation in the Fog. In Proceedings of the 2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC 2018), Zurich, Switzerland, 17–20 December 2018; pp. 164–173.
22. Rivas, J.M.; Gutiérrez, J.J.; Palencia, J.C.; Harbour, M.G. Schedulability Analysis and Optimization of Heterogeneous EDF and FP Distributed Real-Time Systems. In Proceedings of the 2011 23rd Euromicro Conference on Real-Time Systems, Porto, Portugal, 5–8 July 2011; pp. 195–204.
23. Eles, P. Distributed Real-Time Systems. Available online: http://www.it.uom.gr/teaching/distrubutedSite/dsIdaLiu/lecture/lect11-12.frm.pdf (accessed on 22 June 2021).
24. CAR. Distributed Systems and Internet of Things. Available online: https://www.icar.cnr.it/en/sistemi-distribuiti-e-internet-delle-cose/ (accessed on 10 September 2021).
25. Ranjan, R.; Rana, O.; Nepal, S.; Yousif, M.; James, P.; Wen, Z.; Barr, S.; Watson, P.; Jayaraman, P.P.; Georgakopoulos, D.; et al. The Next Grand Challenges: Integrating the Internet of Things and Data Science. *IEEE Cloud Comput.* **2018**, *5*, 12–26. [CrossRef]
26. Tămaş-Selicean, D.; Pop, P.; Steiner, W. Design optimization of TTEthernet-based distributed real-time systems. *Real Time Syst.* **2014**, *51*, 1–35. [CrossRef]
27. Deng, P.; Zhu, Q.; Davare, A.; Mourikis, A.; Liu, X.; Di Natale, M. An efficient control-driven period optimization algorithm for distributed real-time systems. *IEEE Trans. Comput.* **2016**, *65*, 3552–3566. [CrossRef]
28. Mishra, R.; Rastogi, N.; Zhu, D.; Mossé, D.; Melhem, R. Energy aware scheduling for distributed real-time systems. In Proceedings of the International Parallel and Distributed Processing Symposium, Nice, France, 22–26 April 2003; p. 9.
29. Kopetz, H. *Real-Time Systems: Design Principles for Distributed Embedded Applications*; Springer: Berlin, Germany, 2011.
30. Zhao, Y.; Gala, V.; Zeng, H. A Unified Framework for Period and Priority Optimization in Distributed Hard Real-Time Systems. *IEEE Trans. Comput. Des. Integr. Circuits Syst.* **2018**, *37*, 2188–2199. [CrossRef]
31. Xie, G.; Zeng, G.; Li, Z.; Li, R.; Li, K. Adaptive Dynamic Scheduling on Multifunctional Mixed-Criticality Automotive Cyber-Physical Systems. *IEEE Trans. Veh. Technol.* **2017**, *66*, 6676–6692. [CrossRef]
32. Ranjan, R.; Hsu, C.-H.; Chen, L.Y.; Georgakopoulos, D. Holistic Technologies for Managing Internet of Things Services. *IEEE Trans. Serv. Comput.* **2020**, *13*, 597–601. [CrossRef]
33. Korala, H.; Jayaraman, P.P.; Yavari, A.; Georgakopoulos, D. APOLLO: A Platform for Experimental Analysis of Time Sensitive Multimedia IoT Applications. In Proceedings of the 18th International Conference on Advances in Mobile Computing and Multimedia, Chiang Mai, Thailand, 30 November–2 December 2020; pp. 104–113. [CrossRef]
34. Alhamazani, K.; Ranjan, R.; Jayaraman, P.P.; Mitra, K.; Liu, C.; Rabhi, F.; Georgakopoulos, D.; Wang, L. Cross-Layer Multi-Cloud Real-Time Application QoS Monitoring and Benchmarking As-a-Service Framework. *IEEE Trans. Cloud Comput.* **2019**, *7*, 48–61. [CrossRef]
35. Souza, A.; Cacho, N.; Noor, A.; Jayaraman, P.P.; Romanovsky, A.; Ranjan, R. Osmotic Monitoring of Microservices between the Edge and Cloud. In Proceedings of the 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Exeter, UK, 28–30 June 2018; pp. 758–765.
36. 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; pp. 1222–1228.
37. Skarlat, O.; Schulte, S.; Borkowski, M.; Leitner, P. Resource Provisioning for IoT Services in the Fog. In Proceedings of the IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), Macau, China, 4–6 November 2016. [CrossRef]

38. Yigitoglu, E.; Mohamed, M.; Liu, L.; Ludwig, H. Foggy: A Framework for Continuous Automated IoT Application Deployment in Fog Computing. In Proceedings of the 2017 IEEE International Conference on AI & Mobile Services (AIMS), Honolulu, HI, USA, 25–30 June 2017; pp. 38–45.
39. Brogi, A.; Forti, S. QoS-Aware Deployment of IoT Applications through the Fog. *IEEE Internet Things J.* **2017**, *4*, 1185–1192. [CrossRef]
40. Khan, M.S.H.; Roy, P.; Khanam, F.; Hera, F.H.; Das, A.K. An Efficient Resource Allocation Mechanism for Time-Sensitive Data in Dew Computing. In Proceedings of the 2019 International Conference of Artificial Intelligence and Information Technology (ICAIIT), Yogyakarta, Indonesia, 13–15 March 2019; pp. 506–510.
41. Meng, J.; Tan, H.; Li, X.-Y.; Han, Z.; Li, B. Online Deadline-Aware Task Dispatching and Scheduling in Edge Computing. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *31*, 1270–1286. [CrossRef]
42. Štefanič, P.; Cigale, M.; Jones, A.C.; Knight, L.; Taylor, I.; Istrate, C.; Suciu, G.; Ulisses, A.; Stankovski, V.; Taherizadeh, S.; et al. SWITCH workbench: A novel approach for the development and deployment of time-critical microservice-based cloud-native applications. *Futur. Gener. Comput. Syst.* **2019**, *99*, 197–212. [CrossRef]
43. Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Strazdins, P. Investigating decision support techniques for automating Cloud service selection. In Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, Taipei, Taiwan, 3–6 December 2012; pp. 759–764.
44. Xu, Y.; Li, J.; Lu, Z.; Wu, J.; Hung, P.C.; Alelaiwi, A. ARVMEC: Adaptive Recommendation of Virtual Machines for IoT in Edge–Cloud Environment. *J. Parallel Distrib. Comput.* **2020**, *141*, 23–34. [CrossRef]
45. Shlens, J. A tutorial on principal component analysis. *arXiv* **2014**, arXiv:1404.1100.
46. Chen, T.; Guestrin, C. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 785–794.
47. Kousiouris, G.; Cucinotta, T.; Varvarigou, T. The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks. *J. Syst. Softw.* **2011**, *84*, 1270–1291. [CrossRef]
48. Korala, H.; Yavari, A.; Georgakopoulos, D.; Jayaraman, P.P. Design and Implementation of a Platform for Managing Time-Sensitive IoT Applications. In Proceedings of the 2020 IEEE 6th International Conference on Collaboration and Internet Computing (CIC), Atlanta, GA, USA, 1–3 December 2020; pp. 44–53.
49. Bykov, S.; Geller, A.; Kliot, G.; Larus, J.R.; Pandya, R.; Thelin, J. Orleans: Cloud Computing for Everyone. In Proceedings of the 2nd ACM Symposium on Cloud Computing, ACM, New York, NY, USA, 26–28 October 2011; pp. 1–14.
50. Camunda.org. 2019. Available online: https://camunda.com/ (accessed on 30 March 2021).
51. NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy, F5 Inc. Available online: https://www.postgresql.org (accessed on 22 July 2020).
52. Moser, I. A Methodology for Empirically Evaluating Passenger Counting Technologies in Public Transport. In Proceedings of the 41st Australasian Transport Research Forum (ATRF), Canberra, Australia, 30 September–2 October 2019.
53. Nectar. 2018. Available online: https://nectar.org.au/research-cloud/ (accessed on 21 March 2019).
54. Soppelsa, F.; Kaewkasi, C. *Native Docker Clustering with Swarm*; Packt Publishing Ltd.: Birmingham, UK, 2016.
55. Breitbach, M.; Schafer, D.; Edinger, J.; Becker, C. Context-Aware Data and Task Placement in Edge Computing Environments. In Proceedings of the 2019 IEEE International Conference on Pervasive Computing and Communications (PerCom), Kyoto, Japan, 11–15 March 2019; pp. 1–10.
56. Khan, L.U.; Saad, W.; Han, Z.; Hossain, E.; Hong, C.S. Federated Learning for Internet of Things: Recent Advances, Taxonomy, and Open Challenges. *IEEE Commun. Surv. Tutorials* **2021**, *23*, 1759–1799. [CrossRef]
57. Dawod, A.; Georgakopoulos, D.; Jayaraman, P.P.; Nirmalathas, A. An IoT-owned Service for Global IoT Device Discovery, Integration and (Re)use. In Proceedings of the 2020 IEEE International Conference on Services Computing (SCC), Beijing, China, 7–11 November 2020; pp. 312–320.
58. Bamunuarachchi, D.; Banerjee, A.; Jayaraman, P.P.; Georgakopoulos, D. Cyber twins supporting industry 4.0 application development. In Proceedings of the 18th International Conference on Advances in Mobile Computing & Multimedia, Chiang Mai, Thailand, 30 November–2 December 2020; pp. 6–73. [CrossRef]
59. Yavari, A. *Internet of Things Data Contextualisation for Scalable Information Processing, Security, and Privacy*; RMIT University: Melbourne, Australia, 2019.
60. Katsipoulakis, N.R.; Labrinidis, A.; Chrysanthis, P.K. Spear: Expediting stream processing with accuracy guarantees. In Proceedings of the 2020 IEEE 36th International Conference on Data Engineering (ICDE), Dallas, TX, USA, 20–24 April 2020; pp. 1105–1116. [CrossRef]