A Time-Sensitive IoT Data Analysis Framework

Harindu Korala, Dimitrios Georgakopoulos, Ali Yavari, Prem Prakash Jayaraman
Swinburne University of Technology, Australia
{hkorala, dgeorgakopoulos, ayavari, pjayaraman}@swin.edu.au

Abstract

This paper proposes a Time-Sensitive IoT Data Analysis (TIDA) framework that meets the time-bound requirements of time-sensitive IoT applications. The proposed framework includes a novel task sizing and dynamic distribution technique that performs the following: 1) measures the computing and network resources required by the data analysis tasks of a time-sensitive IoT application when executed on available IoT devices, edge computers and cloud, and 2) distributes the data analysis tasks in a way that it meets the time-bound requirement of the IoT application. The TIDA framework includes a TIDA platform that implements the above techniques using Microsoft's Orleans framework. The paper also presents an experimental evaluation that validates the TIDA framework's ability to meet the time-bound requirements of IoT applications in the smart cities domain. Evaluation results show that TIDA outperforms traditional cloud-based IoT data processing approaches in meeting IoT application time-bounds and reduces the total IoT data analysis execution time by 46.96%.

1. Introduction

Internet of Things (IoT) is a new evolution of the Internet that connects a variety of sensors, industrial machines, video cameras, and mobile phones (which we refer to all these as *IoT devices*) that can communicate with each other over the internet [1, 2]. In recent times, data produced from IoT devices (we refer this data as *IoT data*) have increased tremendously and a lot of attention has been given to extract valuable insights from this data [3]. To achieve this, *IoT applications* gather IoT data, analyze them and produce high value information.

In this paper we focus on IoT applications that require the results of their data analysis to be produced within a specific time bound, otherwise the produced information will not be useful. We refer such applications as *Time-Sensitive IoT (TS-IoT) applications* and the requirements of data analysis as *time-bound requirements*. For example, a vehicle accident prediction application must analyse IoT data collected from traffic and on-board cameras and sensors, predict a possible accident and prevent the accident by informing the corresponding driver in near real-time (e.g., within a 30ms time bound). If there is any extra time (i.e., more than the time bound) involved in completing the data analysis, the predicted accident information will not be useful to prevent the accident. To discuss further the problem of addressing time-bound requirements, consider that TS-IoT applications are comprised of a set of data analysis tasks. Each of these tasks may need to perform one of the following: consume IoT data from heterogeneous IoT devices, perform data processing ranging from basic stream processing to resource-intensive machine learning and statistics, manage the data queues required for stateful data analysis, and produce information that is used by other tasks in the same IoT application. Currently, TS-IoT applications, which are comprised of such data analysis tasks, are executed in distributed IoT environments.

Guaranteeing the time-bound requirements of TS-IoT applications heavily depend on the total application execution time. This can be measured as the summation of total data processing time and the total data communication time. The total data processing time is influenced by the resource where the data analysis is performed whilst the total data communication time is influenced by the relevant network delays involved in transferring IoT data to corresponding resources. Therefore, satisfying time-bound requirements heavily depends on the selection of appropriate resources from the IoT environment. However, the decision to select which cloud, edge [4], and/or IoT device resources to execute a TS-IoT application has its trade-offs. Processing IoT data on the IoT devices offers the lowest communication delays, but IoT devices have very limited computing resources. Edge computers have more computing resources than IoT devices, but they are subject to more communication delays than IoT devices. The cloud offers virtually unlimited resources [4] but suffers from significant communication delays when transferring IoT data to the cloud. Furthermore, each task has different resource requirements as well. Therefore, while it is often possible to meet the time-bound requirements of each TS-IoT application by distributing tasks for execution in the IoT devices, edge

URI: https://hdl.handle.net/10125/71486
978-0-9981331-4-0

and cloud resources, 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 any other application due to the volatile nature of the IoT environment. Because of this, it has become a major challenge to address this problem.

In this paper, we propose a novel Time-Sensitive IoT Data Analysis (TIDA) framework that utilises the computing resources available at the IoT devices, edge computers and cloud in meeting the time-bound requirements of each TS-IoT application when the entire pool of available computing resources is sufficient to collectively achieve this. The main contributions of the TIDA framework and this paper are:

1. A novel *dynamic distribution algorithm* for (possibly inter-dependent) IoT data analysis tasks that maintains distributions of such tasks across cloud, edge and IoT devices resources in a way the TS-IoT application meets its time bounds.
2. A TIDA platform that implements the above algorithms, as well as related task measuring, distribution, and migration techniques using Microsoft's Orleans Actor framework.
3. An experimental evaluation that shows that the TIDA platform outperforms existing cloud-based IoT data analysis solutions in a smart city application that requires maintaining a totally accurate count of all passengers that are currently being transported in all the buses of the public transport network of Sydney, Australia.

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

2. 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 transport service to improve planning and scheduling of buses, allocate busses or trains to meet the actual demand, and to respond to unplanned incidents such as bus breakdowns and accidents. To count passengers in this smart city environment we utilized the following IoT devices, edge computers and cloud resources:

1. *Orbbec Persee*[1] *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 use 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 (i.e., RGB), 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 form their sensors, the Orbbec Persee devices provide internal computing and storage resources consisting of a Quad-core Cortex A17 processor (which has a processing speed of 1.8GHz), 2GB RAM and 8GB internal storage.
2. *Edge computers* at bus stops and train stations included cisco 807 industrial service routers[2]. These edge computers act as gateways for IoT devices and connect to the cloud data center via internet. Furthermore, the edge computers include additional computing and storage resources that can be used for IoT data analysis as well.
3. A *Cloud data center* with virtually unlimited computing resources.

In this IoT environment, the IoT devices, edge computers, and cloud are connected with each other via different networks (e.g., NB-IoT, 4G, broadband). The Orbbec Persee IoT devices incorporate Wi-Fi cards and via this 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, the IoT devices can connect to edge computers *only* when they are near bus stops or train stations. Edge computers and cloud data center 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 stepping in and out of each bus, 2) analyze the collected RGB/infrared/depth data and to 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 existing by applying classification techniques such as 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 it is not discussed further in this paper)

[1] https://orbbec3d.com/product-persee/

[2] https://www.cisco.com

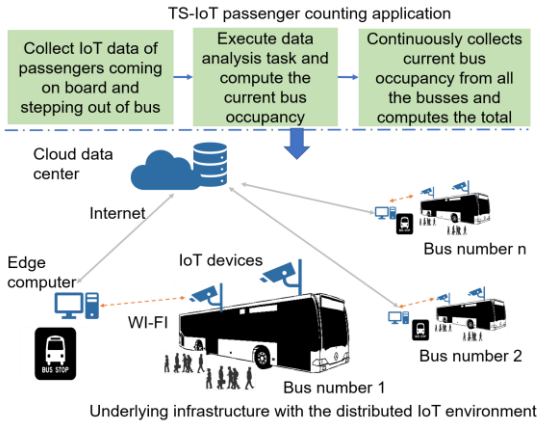

Figure 1. Illustration of motivating scenario

and 3) calculating the total occupancy of the bus, and 4) computing the total occupancy of all the busses in the transport network. Figure 1 illustrates the motivating scenario, computing resources, and IoT data analysis tasks in this TS-IoT application.

The IoT passenger count application has a variable timebound that is hard to meet, i.e., 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 time-bound requirements in IoT often depends on the selection of computing and networking resources for each TS-IoT application. In passenger counting IoT application, though we perform the entire data analysis quickly in the cloud, this may involve significant communication delay to collect all the passenger RGB/infrared/depth data. Offloading the collected passenger data to edge computers and performing the data analysis in edge computer is another option. However, the only limited time to transfer the passenger data to the edge computer, many buses may be near each bus stop, 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 way that the entire TS-IoT application meets its time bound(s).

3. System model & problem formulation

Due to the trade-offs between IoT resources in the distributed IoT environment, it is necessary to generate a task distribution plan (which meets the application's time-bounds requirement) by determining relevant communication delays involved and needed computing resources capacities for each task. 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 tasks distribution problem as an optimization problem.

Resource model. Computing resources (i.e., IoT devices, edge computers and cloud) and network resources in the distributed IoT environment form a graph $G_{Res} = (Comp\_Res, Network\_Res)$, where $Comp\_Res$ represent the distributed computing resources and $Network\_Res$ represent the network links between computing resources. A single computing resource of G_{Res} can be denoted cr_i, where $cr_i \in Comp\_Res$ and $i \in 1 \dots m$, 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 resources available at cr_i. Further, $Available\_Res_{cr_i}$ can be represented as a tuple of $\langle cpu^i_{cr}, ram^i_{cr} \rangle$.

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 upload bandwidth available and $down^{ij}$ is the amount of download bandwidth available in nr_{ij}.

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 $Tasks$ represent the tasks of the TS-IoT application and $Dataflows$ represent 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 $i \in 1 \dots n$, 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 to buffer a certain number of data items before processing them. We identify the number of data items required to buffer in a stateful task as queue size and denote this as q_{t_i}. Stateless tasks do not require to buffer 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 or not, we denote the following binary attribute, $is\_stateful_{t_i}$: $is\_stateful_{t_i}$=1 if 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 don't 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 gets executed. A t_i, is associated with two delays as well. We denote them as $Start\_Delay_{t_i}$ and $Wait\_Delay_{t_i}$. 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 tasks 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 $Com\_delay_{d_{ij}}$.

The above model is based on the following assumptions:

1. We assume that cloud data centres in the IoT environment to have unlimited computing (CPU, memory, and storage) resources, whilst IoT devices and edge computers to have limited computing and storage resources.
2. We assume the bandwidth of all the network links to be limited in capacity and static.
3. We assume the $Task\_Res_{t_i}$ can be obtained by measurements via executing the corresponding task on a reference computing resource.
4. We assume the $Proc\_time_{t_i}$ on a computing resource can be obtained by estimating based on previous measurements.
5. We assume the G_{Res}, is developed by considering the amount of computing resources and their networks available in the IoT environment.

Problem formulation Our objective is to generate an application-specific, time-bound satisfying task distribution plan for the IoT environment within the available resources. To realise this, we need to generate a task distribution plan in an IoT environment in a way 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:

$$Total\_Exec_{App} = \max_{p \in 1 \ldots Paths} (Path\_Exec\_Time_p) \quad (01)$$

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 task, buffering data at tasks etc.) of each task that is in that path p. Given this definition, we obtain the following:

$$Path\_Exec\_Time_p = \sum_{j=1}^{Y} Task\_Exe_{t_j} \quad (02)$$

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:

$$Task\_Exe_{t_j} = Proc\_time_{t_j} + Com\_delay_{d_{ij}}.q_{t_j} + Wait\_delay_{t_i}.q_{t_j} + Start\_delay_{t_i} \quad (03)$$

In equation 03, $Proc\_time_{t_j}$ 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 denoted as q_{t_j}. Note in here we don't 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 01. 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 gets multiplied by q_{t_j} (i.e., the queue size of task t_j). $Com\_delay_{d_{ij}}$ can be calculated using the following:

$$Comm\_delay_{d_{ij}} = \frac{Data_{d_{ij}}}{Available\_Band_{nr_{ij}}} \quad (04)$$

where, $Data_{d_{ij}}$, denotes the size of a single data piece that need 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 available bandwidth of the nr_{ij}.

Decision variables: We define the decision variables that form the task distribution plan as follows: First decision variable $\alpha_{t_j}^{cr_i}$ denotes whether a task t_j is distributed on a computing resource cr_i or not. The next decision variable $\gamma_{d_{ij}}^{nr_i}$ denotes whether a dataflow d_{ij} is placed on a network resource nr_i or not.

Constraints: First, the task distribution on 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: $\forall\, cr_i \in Comp\_Res$,

$$\sum_{t_j}^{Tasks} Task\_Res_{t_j} * \alpha_{t_j}^{cr_i} \leq Available\_Res_{cr_i} \quad (05)$$

Each network link can only transfer data that is within its available bandwidth and we can formally denote it as follows:
$\forall\, nr_{ij} \in Network\_Res$,

$$\sum_{d_{ij}}^{Dataflows} Data_{d_{ij}} * \gamma_{d_{ij}}^{nr_{ij}} \leq Available\_Band_{nr_{ij}} \quad (06)$$

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} or not.

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

$$Total\_Exe_{App} \leq TB_{App} \quad (07)$$

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

Minimize:
$$Total\_Exec_{App} = \max_{p \in 1 \ldots Paths} (Path\_Exec\_Time_p) \quad (08)$$
Subject to: Eq (05), Eq (06) and Eq (07)

However, solving this problem tends to be NP hard, hence we aim to solve this problem using a novel task sizing technique and a greedy heuristic approach described in the next section.

4. Dynamic task distribution

Dynamic task distribution consists of two main components, the task sizing technique and greedy task distribution algorithm. Contrast to the traditional cloud-based IoT data processing approach, in here the proposed techniques explore how tasks can exploit the resources found at IoT devices as well as nearby edge computers to reduce the communication delay. Another possibility of the proposed techniques is that, we can execute this multiple time to produce different task distribution plans in instances where certain computing resources are disconnected from the IoT environment.

4.1 Task sizing technique

Task sizing technique is used for measuring the computing and network resources required by the tasks when they are executed in the available IoT devices, edge computers and cloud. This gets executed whenever the underlying IoT environment changes, thus allows us to obtain IoT environment specific measurements for each task in the TS-IoT application. This technique takes G_{App}, G_{Res} as inputs. As the first main step, the algorithm creates a *TaskList*, by traversing through the task graph G_{App} in breadth first search (BFS) manner. Then it creates a *ResourceList*, from the resource graph G_{Res}. Then for each resource in the *ResourceList*, every task is executed. Then during the execution, the computing and network resources required by each task and the execution time for each task is measured and recorded in the measurement table. This process is repeatedly done until the end of resources in the resource list. The output of the task sizing technique is a measurement table, which is comprised of computing and network resources required for each task on each resource. Figure 2 illustrates the pseudocode for the task sizing technique.

Algorithm 1 Task Sizing Technique
Input: G_{App}, G_{Res}, TB_{App}
Output: MetricsTable
1: TaskList = CreateTaskListFromAppGraph(G_{App})
2: ResourceList = CreateResourceListFromResGraph(G_{Res})
3: **for each** resource in ResourceList **do**
4: **for each** task in TaskList **do**
5: MetricsTable = ExecuteAndMeasure(task, resource)
6: **end for**
7: **end for**

Figure 2. Pseudocode for the task sizing techniques

4.2 Greedy task distribution algorithm

In this section, we discuss the proposed greedy task distribution algorithm to solve the problem formulated in section 3.2. In here, we follow a greedy heuristic approach that aims to incrementally solve the task distribution problem and finally generate a task distribution plan. Figure 3 illustrates the pseudocode of the proposed greedy algorithm.

Algorithm 2 Greedy heuristic algorithm for constructing task distribution map

Input: TaskList, ResourceList, MetricsTable, TB_{App}
Output: TaskDistributionMap <task, resource>

```
1:  TaskDistributionMap <task, resource> = null
2:  SortedResourceMap <resource, executionTime> = null
3:  Total_Exe_App = null
4:  CanDistribute = null
5:  while TaskList is not empty and CanDistribute is true do
6:     task = TaskList.GetItem()
7:     if CheckFirstTask(task) then
8:        SortedResourceMap = GetResourcesCloserToDataSource()
9:     else
10:       SortedResourceMap = GetResources(task, MetricsTable)
11:    end if
12:    TaskPlaced = false , i = 0
13:    while i < SortedResourceMap.Count() and TaskPlaced is false do
14:       resource = SortedResourceMap.GetItem(i)
15:       if CheckResourceCapacity(task, resource) then
16:          TaskDistributionMap.Add(task, resource)
17:          UpdateAvailableResourceCapacities(resource, ResourceList)
18:          TotalExe_App += SortedResourceMap.GetValue(resource)
19:          TaskPlaced = true
20:       end if
21:       i++
22:    end while
23:    if TaskPlaced is false or Total_Exe_App > TB_App then
24:       CanDistribute = false
25:    end if
26: end while
27: if Total_Exe_App =< TB_App and CanDistribute is true then
28:    return TaskDistributionMap
29: else
30:    return false
31: end if
```

Figure 3. Pseudocode for the dynamic task distribution algorithm

The algorithm takes the task list, resource list, measurement records and TB_A as inputs. Then for each task in the task list, the algorithm finds an eligible (i.e., has enough capacity to fulfil the resources required by the task) computing resource, that 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 algorithm 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 will always get assigned to a computing resource that is closer to the data source, provided it has enough resource capacity (lines 7-8). On the other hand, 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 9 - 10).

Once the sorted resources map is created, the algorithm iterates through each item in sorted resources map until it finds an eligible computing resource. When the algorithm identifies an eligible computing resource, it first assigns that resource to the corresponding task via updating task distribution map, then update the available resources of the selected resource, update the $Total\_Exe_{App}$ based on the estimated execution time, exists the while loop and move to the next task in the task list (lines13-23). The algorithm 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. If a task couldn't find any eligible computing resource or $Total\_Exec_{App}$ exceeds TB_{App}, the algorithm stops executing and indicates that the TS-IoT application cannot meet its TB_{App} with the current available resources or else the tasks will be distributed according to the task distribution map (steps 27-30).

5. TIDA framework

To overcome the challenges of meeting time-bound requirements of TS-IoT applications, we introduce a novel time-sensitive IoT data analysis (TIDA) framework that utilizes cloud, edge and IoT devices resources. In this section, we discuss design and implementation of the framework via scalable and efficient distributed task management.

5.1. Architecture of the TIDA framework

We propose the following architecture for the framework, which is illustrated in figure 4. In this section, we discuss each component of the architecture.

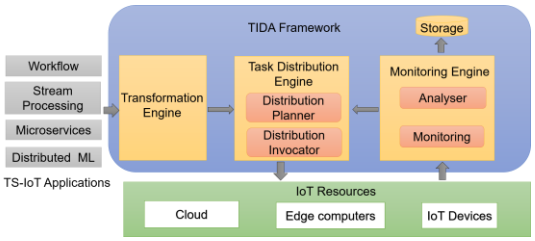

Figure 4. Architecture of the framework

Transformation Engine: To execute any TS-IoT application irrespective to its underlying application model, we propose a transformation technique, which transforms data analysis tasks of any TS-IoT application into a set of common executable units of the framework. 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 functionality of the framework 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 framework.

Task Distribution Engine: This is responsible for efficiently managing the distribution of tasks. The task distribution engine is comprised of two modules. They are distribution planner module and distribution invocator module. Distribution planner can accommodate different task distribution algorithms. Task distribution algorithms (such as dynamic task distribution algorithm discussed in section 04) generate task distribution plans. Then these task distribution plans are sent to the distribution invocator, which then distributes the tasks to the corresponding resources according to the plan and invoke their executions.

Monitoring Engine: This engine continuously monitors the execution landscape in terms of resource utilization (CPU and RAM) and execution progress of tasks. It is comprised of two modules: Monitoring and Analyzer. Monitoring module monitors and collects resource usage and execution information and forward them to analyzer module. Analyzer analyses the monitored data and identifies whether the time-bounds can be met with the current execution or not.

5.2. Implementation of the TIDA platform

A proof of concept implementation of TIDA platform [5] was implemented using Microsoft's Orleans Actor framework [6]. Orleans actors are developed to scale in an elastic way and they can run on any operating system that has .NET core installed. Therefore, we decided to implement TIDA platform's underlying executable units as Orleans actors. This facilitated us to develop a highly scalable and efficient task management system that led us to develop a proof of concept task distribution engine. Furthermore, we implemented the discussed greedy dynamic task distribution algorithm as part of the task distribution engine. In addition to the greedy algorithm, we implemented a random task distribution algorithm that generates random task distribution plans. 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 modelled using camunda [7] workflow modeler. The monitoring engine was implemented as an Orleans start-up service, which gets activated when TIDA is up and running. The monitoring engine periodically (every second) collects metrics such as CPU utilization percentage, RAM utilization percentage and the execution progress of tasks. The collected metrics are stored in a database via Orleans's persistent capabilities. PostgreSQL [8] relational database was used as our storage provider. This stores performance metrics, application specific data and information of the resources such as health of each resource etc.

6. Evaluation

In this section, we discuss how the TIDA was evaluated and present the results.

6.1. Methodology for Experiments

In this evaluation, we considered the IoT environment to be static throughout the evaluation. Therefore, the proposed task sizing technique is executed only once before the start of the task distribution, thus the evaluation is solely focused on the greedy distribution algorithm of TIDA.

Testbed configurations: We created a testbed in the cloud using NECTAR research cloud [9].The testbed consists of a cluster of four cloud instances. To emulate edge and IoT devices, we created two cloud instances with similar system configurations of real world edge and IoT devices. For this purpose, we considered the system configurations of cisco 807 industrial services router for the edge device and Orbbec Persee camera's system configurations for the IoT device. We created a PostgreSQL database server in another cloud instance that is responsible for storage and cluster membership. Before we ran our experiments, we installed our platform's runtime on each instance of the testbed. Table 1 illustrates the system configurations of the computing resources used in the testbed.

Table 1. System configurations of computing resources

Computing Resources	CPU	RAM
Cloud server	2.5GHz Intel Core Processor 4 VCPUs	12 GB
Edge device	2.29GHz Intel Core Processor 2 VCPUs	4 GB
IoT device	2.29GHz Intel Core Processor 1 VCPUs	2 GB

IoT application, Dataset and Task Distribution Plans: We developed the IoT application as a workflow application. To model the application, we used camunda workflow modeler. The application consists of three tasks 1) pre-processing 2) classification and 3) counting. We developed each of these three tasks as a C# program and we utilized OpenCV library for the preprocessing

task and classification task. For the dataset, we used real video data collected using an Orbbec Persee camera during a trial project carried out in Sydney, Australia [10]. For this experiment, we used a RGB video file, which is 20 seconds long and that has a resolution of 640 x 480 and 30 FPS (frames per second). We executed the IoT application multiple times under different task distribution plans provided by five task distribution algorithms including the greedy dynamic task distribution algorithm, which was discussed in section 4. Table 2 illustrates the five task distribution algorithms and how tasks were distributed in the computing resources.

Experimental evaluation metrics: We measured the following performance metrics during the execution of the application.

- 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.

Table 2. Task distribution algorithms

Task distribution algorithm	Description
Cloud only	All tasks to the cloud server
Edge only	All tasks to the edge device
IoT device only	All tasks to the IoT device
Random distribution	Randomly generate a task distribution plan
Greedy dynamic task distribution	Use greedy dynamic task distribution algorithm to task distribution plan

6.2. Experimental evaluation results

Figure 5 compares the total data processing time, total data communication time and total application execution time of the passenger counting IoT application under each task distribution algorithm. (Note in here, we have taken the average values for the comparison.)

Although, executing all the tasks in IoT devices resulted in zero data communication time, this has recorded the highest total data processing time, due to the limited computing resources in IoT devices. On the other hand, executing all the tasks in the cloud or edge devices have notably improved the total data processing time compared to that of IoT devices. However, the total application execution time hasn't improved much in both occasions (i.e., all tasks at cloud and edge), due to the data communication time involved in sending data to the edge device and the cloud server. Random task distribution algorithm generates different tasks distribution plans for the application randomly without

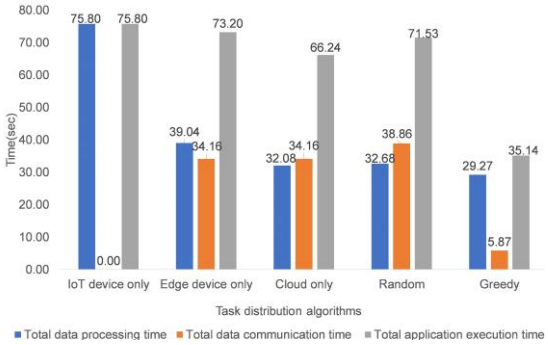

Figure 5. Comparison of total data processing time, total data communication time and total application execution time under each task distribution algorithm

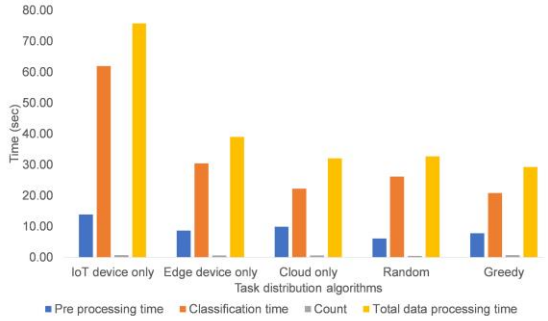

Figure 6. Comparison of data processing times of each task under each task distribution algorithm

considering the IoT environment or IoT application requirements such as time-bound requirements, resource requirements for tasks etc. Therefore, by looking at the results, we can see that the task distribution plans generated by random task distribution algorithm shows mediocre results. The greedy dynamic task distribution algorithm aims to generate time-bound satisfying, application and IoT environment specific task distribution plans. Therefore, we can notice that compared to the other four task distribution algorithms, the greedy dynamic task distribution algorithm has significantly improved the total data processing time, total data communication time and total application execution time. Furthermore, if we make a comparison between executing all of the tasks in the cloud, which is the traditional way of IoT application execution, and executing tasks based on the task distribution plans generated by greedy dynamic task distribution algorithm, we can observe that, greedy algorithm has improved the total data processing time by 8.59%, the total data communication time by 82.81% and the total application execution time by 46.96%.

Figure 6 illustrates the data processing times for each task. This figure shows that the classification tasks'

data processing time and the computing resource where it takes place significantly influences the total data processing time of the IoT application compared to the other two data analysis tasks (i.e., pre-processing task and count task). Furthermore, we can notice that the lowest data processing time for the classification task results when the greedy dynamic task distribution algorithm is used.

In summary, the evaluation demonstrated TIDA's capability in distributing tasks of a TS-IoT application in IoT devices, edge devices and cloud resources. The evaluation results showed that the task distribution plans generated by the greedy dynamic task distribution algorithm of the TIDA has improved the total application execution time of the passenger counting IoT application by 46.96% and reduced the IoT data communication overhead by 82.81%, compared to the traditional cloud-based approach in executing IoT applications. Moreover, we noticed that only the task distribution plans generated by the greedy dynamic task distribution algorithm met the time-bound requirement of the passenger counting IoT application, whilst the others failed to guarantee the time-bound requirement.

7. Related Work

Meeting the time-bound requirements of TS-IoT applications is challenging due to the heterogenous and volatile nature of the IoT environment and the time-bound requirements of such applications [4]. In [5] 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 a way that they collectively meet the application's time-bound requirements. To enable such task distribution, we proposed the use of task sizing techniques 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 [11] and [12] that 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 doing this via the Cloudsim simulator. [13] proposed a technique for measuring the performance of computing resources when different IoT application tasks are executed there, while [14] introduced a platform to experimentally evaluate performance of TS-IoT applications.

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 that. For example, [15] proposed a technique for efficient distribution of application tasks across cloud, edge resources in a resource-aware manner. [16] proposed an optimization technique that generates task distribution plan for IoT applications. [17] introduces a technique for optimizing the scheduling IoT application tasks in edge devices. [18] formulates IoT application distribution as an Integer Non-Linear Problem (INLP). It then used INLP to minimize the cost of resource usage while satisfying QoS requirements of the applications. The optimization techniques proposed by [19, 20] determines appropriate computing resource selection for meeting the QoS requirements of IoT applications. Related computing frameworks and tools, such as [21], [22] and [23], have employed similar techniques to manage the distribution of TS-IoT applications while a QoS simulation-based tool for IoT applications. [24] proposed a recommender system for dealing with the heterogeneity of cloud computing resources

In summary, task sizing techniques found in the literature have relied on simulation tools [11, 12] or include limited testbeds [13] for sizing tasks. Such techniques cannot effectively estimate the resources needed by TS-IoT application tasks because they do deal with the heterogeneity and dynamic nature of the IoT environment. Most of the task distribution techniques in the literature employ complex optimization techniques [17,18,19,20] to device task distribution plans and most of them do not consider task sizing and they are expensive to compute. Due to these reasons, these techniques are not suitable for TS-IoT applications that have demanding time-bound requirements. On contrary, TIDA presents a novel dynamic task distribution technique that includes 1) task sizing that measures the computing and network resources required by the tasks when they are executed in the IoT environment, and 2) a greedy algorithm that uses the task sizing information to generate time-bound satisfying task distribution plans to distribute tasks in IoT environment. Furthermore, TIDA has been implemented by extending Microsoft Orleans and the greedy algorithm has been evaluated using a real world smart city application.

8. Conclusion and future work

In this work, we proposed a novel time-sensitive IoT data analysis (TIDA) framework for meeting time-bound requirements of TS-IoT applications. We first defined a formal system model for TS-IoT applications and IoT environment. Next, we formulated the task distribution problem as an optimization problem and proposed a novel task sizing technique and a dynamic task distribution algorithm to solve the task distribution

problem. We implemented TIDA platform that implements the above algorithms and Microsoft's Orleans framework. We evaluated the TIDA by developing a passenger counting IoT application, executing the application in a cloud-based testbed under different task distribution plans provided by five task distribution algorithms and assessing how well each of these task distribution plans enable the application to meet its time-bound requirements. The results showed that the TIDA on average improves the total application execution time by 46.96% and total data communication time by 82.81%, compared to traditional cloud-based processing of the passenger counting IoT application. Moreover, the dynamic task distribution algorithm of TIDA successfully met the time-bound requirement of the passenger counting IoT application in each execution iteration as well. In our future work, we plan to develop cost effective dynamic task adaptation techniques to deal with possible time-bound violations and to compare TIDA platform's ability to meet time-bound requirements with existing solutions.

9. References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on Enabling Technologies, Protocols, and Applications," in *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347-2376, 2015.

[2] D. Georgakopoulos and P. P. Jayaraman, "Internet of things: from internet scale sensing to smart services," *Computing*, vol. 98, no. 10, pp. 1041-1058, 2016.

[3] J. Chen *et al.*, "Big data challenge: a data management perspective," *Frontiers of Computer Science*, vol. 7, no. 2, pp. 157-164, 2013.

[4] R. K. Naha *et al.*, "Fog Computing: Survey of trends, architectures, requirements, and research directions," *IEEE Access*, vol. 6, pp. 47980-48009, 2018.

[5] H. Korala, A. Yavari, D. Georgakopoulos, and P. P. Jayaraman, "Design and Implementation of a Platform for Managing Time-Sensitive IoT Applications," in *2020 IEEE 6th International Conference on Collaboration and Internet Computing (CIC)*, December 2020.

[6] S. Bykov, A. Geller, G. Kliot, J. R. Larus, R. Pandya, and J. Thelin, "Orleans: cloud computing for everyone," in *2nd ACM Symposium on Cloud Computing*, pp. 1-14, New York, Oct 2011.

[7] Camunda.org. , "Camunda.", https://camunda.com/

[8] The PostgreSQL Global Development Group, "PostgreSQL", https://www.postgresql.org/

[9] National Research Infrastructure for Australia" Nectar Cloud." https://nectar.org.au/research-cloud/

[10] I. Moser *et al.*, "A Methodology for Empirically Evaluating Passenger Counting Technologies in Public Transport," in *41st Australasian Transport Research Forum (ATRF)*, Canberra, Oct 2019.

[11] M. Aazam and E. Huh, "Dynamic resource provisioning through Fog micro datacenter," in *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom workshops)*, pp. 105-110, St.Louis, Mar 2015.

[12] M. Aazam, M. St-Hilaire, C. Lung, and I. Lambadaris, " MeFoRE: QoE based resource estimation at Fog to enhance QoS in IoT," in *2016 23rd International Conference on Telecommunications (ICT)*, pp. 1-5, Thessaloniki, May 2016.

[13] M. Arlitt, M. Marwah, G. Bellala, A. Shah, J. Healey, and B. Vandiver, "IoTAbench: an Internet of Things Analytics Benchmark," in *6th ACM/SPEC International Conference on Performance Engineering*, pp. 133-144, New York, Jan 2015.

[14] H. Korala, P. P. Jayaraman, A. Yavari, and D. Georgakopoulos, "APOLLO: A Platform for Experimental Analysis of Time Sensitive Multimedia IoT Applications," in *18th International Conference on Advances in Mobile Computingand Multimedia (MoMM '20)*, Chiang Mai, December 2020.

[15] M. Taneja and A. Davy, "Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 1222-1228, Lisbon, May 2017.

[16] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, "Resource Provisioning for IoT Services in the Fog,"in *2016 IEEE 9th international conference on service-oriented computing and applications (SOCA)*, pp. 32-39, Macau, Nov 2016.

[17] H. Hong, P. Tsai, and C. Hsu, "Dynamic module deployment in a fog computing platform," in *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 1-6, Kanazawa, Oct 2016.

[18] A. Yousefpour *et al.*, " FOGPLAN: A Lightweight QoS-Aware Dynamic Fog Service Provisioning Framework," in *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5080-5096, June 2019.

[19] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards QoS-Aware Fog Service Placement," in *2017 IEEE 1st international conference on Fog and Edge Computing (ICFEC)*, pp. 89-96, Madrid, May 2017.

[20] L. Li, S. Li, and S. Zhao, "QoS-Aware Scheduling of Services-Oriented internet of Things," in *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1497-1505, May 2014.

[21] E. Yigitoglu, M. Mohamed, L. Liu, and H. Ludwig, "Foggy: A Framework for Continuous Automated IoT Application Deployment in Fog Computing," in *2017 IEEE International Conference on AI & Mobile Services (AIMS)*, pp. 38-45, Honolulu, June 2017.

[22] P. Michalák and P. Watson, "PATH2iot: A Holistic, Distributed Stream Processing System," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 25-32, Hong Kong, Dec 2017.

[23] A. Brogi and S. Forti, "QoS-Aware Deployment of IoT Applications Through the Fog," in *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1185-1192, Oct 2017.

[24] M. Zhang, R. Ranjan, A. Haller, D. Georgakopoulos, and P. Strazdins, "Investigating decision support techniques for automating Cloud service selection," in *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pp. 759-764, Taipei, Dec 2012.