Orchestration and Management of Adaptive IoT-centric Distributed Applications

Sehrish Amjad, Ahmed Akhtar, Muhammad Ali, Ayesha Afzal, Basit Shafiq, Jaideep Vaidya, Shafay Shamail and Omer Rana

Abstract—Current Internet of Things (IoT) devices provide a diverse range of functionalities, ranging from measurement and dissemination of sensory data observation, to computation services for real-time data stream processing. In extreme situations such as emergencies, a significant benefit of IoT devices is that they can help gain a more complete situational understanding of the environment. However, this requires the ability to utilize IoT resources while taking into account location, battery life, and other constraints of the underlying edge and IoT devices. A dynamic approach is proposed for orchestration and management of distributed workflow applications using services available in cloud data centers, deployed on servers, or IoT devices at the network edge. Our proposed approach is specifically designed for knowledge-driven business process workflows that are adaptive, interactive, evolvable and emergent. A comprehensive empirical evaluation shows that the proposed approach is effective and resilient to situational changes.

Index Terms—IoT-centric workflows, Edge Computing, Knowledge-driven business processes, Service orchestration.

1 Introduction

The emerging cloud and edge computing infrastructure provides new opportunities to develop next-generation Internet-centered distributed applications that are adaptive, evolvable and emergent [1], [2]. Such applications may include knowledge-driven distributed workflows that are dynamically orchestrated and managed by utilizing computation, data and storage resources available in a cloud data center, enterprise networks as well as Internet of Things (IoT) devices. IoT devices in such workflows provide a diverse range of functionalities, from measurement and dissemination of sensory data observation to computation services for real-time data stream processing. In workflows that are designed for extreme situations such as emergencies, a significant benefit of IoT devices is that they can help gain a more complete situational understanding of the environment. However, this requires the ability to effectively utilize resource constrained IoT devices.

The IoT applications that we consider in this paper are knowledge-driven workflows. A unique aspect that differentiates them from traditional workflows (business processes, scientific workflows, etc.) are that they are emergent and their execution evolves based on the knowledge of the execution status, environmental context, situation and case-specific parameters that are not known apriori and are subject to change at runtime. Moreover, in such workflows,

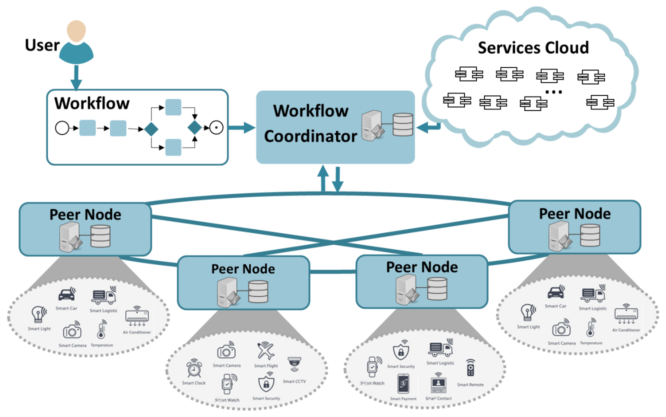

Fig. 1. Environment for Orchestration and Management of IoT-centric Distributed Workflow Applications

the binding of tasks to service / resource endpoints may not be known at design time [3]. The orchestration and management of such workflows therefore requires dynamic discovery, selection and binding of workflow tasks to available cloud or edge resources as well as establishing coordination between these resources based on the functional and non-functional requirements of the workflow.

We address this challenge and present an integrated framework for dynamic orchestration and management of IoT-centric and knowledge-driven workflow applications. In the cloud and edge computing environment we consider, users submit their workflow orchestration and management requests to a *workflow coordinator* by providing high-level workflow specifications. The different tasks in the workflow may run on computation/ data resources on the cloud, edge nodes and on IoT devices. These edge nodes and IoT devices may be geographically distributed. For binding the workflow tasks to the corresponding edge resource or IoT device, we consider a peer-to-peer overlay network of intercon-

- *S. Amjad, A. Akhtar, M. Ali, B. Shafiq and S. Shamail are with the Department of Computer Science, Lahore University of Management Sciences, Lahore, Pakistan. E-mail: 16030054@lums.edu.pk, 16030059@lums.edu.pk, 20030054@lums.edu.pk, basit@lums.edu.pk, sshamail@lums.edu.pk*
- *A. Afzal is with the Department of Computer Science, Air University, Pakistan. E-mail: ayesha@aumc.edu.pk*
- *J. Vaidya is with Rutgers University, Newark, NJ 07102. E-mail: jsvaidya@rutgers.edu*
- *O. Rana is with School of Computer Science & Informatics, Cardiff University, Cardiff, UK E-mail: ranaof@cardiff.ac.uk*

nected nodes consisting of geographically distributed edge devices. Peer nodes in this overlay network are partitioned into cells based on their locations. Each cell has one or more peer nodes which maintain information about other edge or IoT devices in the cell and their capabilities (in a local database). For workflow orchestration, the coordinator dynamically binds the workflow tasks to the services available in the cloud or edge/ IoT devices discovered through peer nodes based on the location and context requirements of tasks. The peer nodes may also act as local orchestrators. The binding may need to change in real time as the dynamics of the underlying environment change. To illustrate the requirements for dynamic orchestration and management of IoT-centric and knowledge-driven workflows an illustrative example is provided in section 1.1.

1.1 Illustrative Application Scenario

Consider a real-time incident management workflow application of a Supply Chain Monitoring Company that collects, integrates and analyzes real-time data from different data sources (e.g., emergency reporting systems and social media) and IoT devices (e.g., sensors – environmental, traffic, chemical, fire detection, surveillance cameras, etc.) This incident management workflow is depicted in Fig. 2. This workflow is instantiated dynamically in response to a highway accident in which a tractor-trailer carrying several drums of liquid acetone has overturned and exploded, generating a smoke plume that spreads over a large area. This workflow provides users with a real-time picture of their in-transit cargo and notifies them of risks and disruptions due to ongoing/ developing disasters/ accidents along the routes. It also reroutes cargo trucks to avoid delays, as well as risks due to ongoing/developing disasters.

For identifying the at-risk areas, the workflow deploys and runs a plume modeling service (e.g., ALOHA [4]) on an edge device. For dynamic generation of the plume model, this service requires a continuous stream of sensory data observations (including wind speed, wind direction, temperature, humidity and aerosol concentration) from multiple locations in the vicinity of the incident site. For example, the service may require receiving observations from a grid of location points that are separated by a distance of 50 meters from each other and are spread within a 500 meter radius from the incident site. The data observations can be obtained from different sensors installed on cars and buses that are near the incident site as well as from fixed environmental sensors installed along the highway. The sensor data originating from such diverse sources also requires pre-processing and integration in order to address the data granularity and format issues before it is fed to the plume modeling service. Such pre-processing and integration needs to be carried out on-the-fly on a nearby IoT/ edge device with computation and storage capability, for example on a smartphone of a user or on an onboard computer system of a vehicle. Similarly, other tasks (e.g., traffic congestion monitoring, rerouting cargo trucks, etc.) of this workflow also require collection, integration and preprocessing of data from different sensor data streams. Note that the location of these sensors/ IoT devices may not be known apriori and are determined after completion of prior

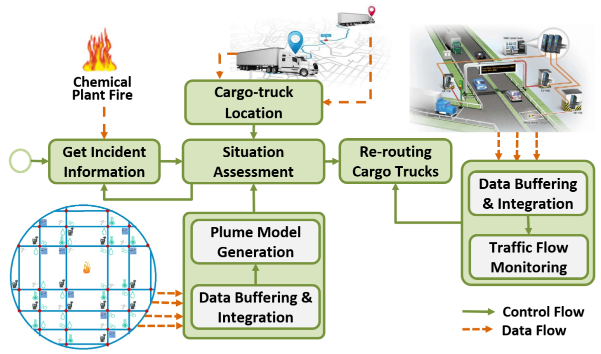

Fig. 2. IoT-centric Distributed Workflow for Incident Management

tasks of the workflow. For example, for traffic congestion monitoring and cargo truck rerouting, live traffic camera feeds are retrieved from different roads/ intersection points after identifying at-risk areas. Similarly, plume modeling related tasks cannot be bound to resources at design time due to their dependency on the incident location and unpredictable connectivity/ accessibility of the source services and devices in the vicinity of the incident.

The proposed framework enables orchestration and management of the above incident management workflow as well as other IoT-centric knowledge-driven workflows through location-aware and flexible service/ resource selection, iterative & incremental binding, and dynamic service deployment by taking into account resource availability status, spatio-temporal constraints, and resource constraints of the underlying service infrastructure as well as real-time processing constraints of the application.

The main contributions of this paper are to:

- Develop an integrated framework for orchestration and management of IoT-centric and knowledge-driven workflow applications that are dynamically composed and adapted.
- Provide a centralized as well as a distributed approach for orchestration and hierarchical resource management of IoT-centric distributed workflow applications.
- Perform an empirical evaluation of both centralized and distributed approaches by emulating two real-world streaming workflow examples. We also compare the performance of our approaches with an existing Edge Cluster Stream Processing (ECSP) approach.

The rest of the paper is organized as follows. Section 2 presents the background/ context and problem statement. Section 3 discusses the proposed approach and Section 4 provides implementation details. Section 5 presents experimental evaluation results. Section 6 discusses related work. Finally, Section 7 concludes the paper and discusses future work.

2 Preliminaries and Problem Statement

We now introduce the notations and formally define the key concepts discussed in the paper and then provide a formal statement of the problem under consideration.

Definition 1. *(Resource):* A resource $r \in \mathcal{R}$ is denoted by a tuple $r = \langle i, \rho, \pi, \mathcal{H}, \Sigma, q \rangle$, where i is the resource identifier, ρ denotes its type e.g., sensor, actuator, fog/edge device, or a cloud service, π denotes its location (URI), $\mathcal{H}$ and Σ denote

the software and hardware specification respectively, and q denotes quality-of-service (QoS) properties of r.
We represent the QoS properties of a resource as a list of attribute-value pairs $\{\langle q^{\alpha}, v^{\alpha}\rangle\}$.

Definition 2. *(Workflow):* An application workflow W is a two-parametric model (Γ, Λ) where:

- Γ is a set of tasks, each denoted by a tuple $\gamma = \langle i, l, d, \mathcal{H}, \Sigma, I, O, \delta, \rho, r\rangle$ where, i is the task identifier, l is the task's location requirement, d is the task duration, $\mathcal{H}$ and Σ are its software and hardware requirements respectively, I and O denote the input and output parameters for the task respectively, δ indicates whether γ is a deployable or non-deployable task, ρ denotes the type of resource the task requires for execution (e.g., sensor, actuator, edge device, cloud service), and r is the mapped resource.
- $\Lambda \subseteq \{\langle(\gamma,\ \acute{\gamma}),\ q,\ m\rangle\}$ denotes the interaction between a pair of tasks $(\gamma, \acute{\gamma}) \in \Gamma \times \Gamma$, q indicates QoS parameters defined in the interaction e.g., latency, bandwidth, sampling rate (samples per second - the rate at which data is read and displayed), and m is the message type (request/ response).

We assume that the workflow is specified at the finest granularity level with each task binded to only one resource.

Example 1. The wind measurement and plume computation tasks of the incident management workflow are modeled as follows:

- γ_1 = $\langle$ *wind_measurement, within 200 meter radius of incident location, 5 min,* $\{\langle$*Measuring range: 0.3 to 75 m/s*$\rangle$, $\langle$*Accuracy:+ 0.2 m/s or better*$\rangle, \dots\}$, $\{$*data processing software*$\}$, $\{$*lat-long, radius, measurement unit, precision,* $\dots\}$, $\{$*wind speed data*$\}$, *"non-deployable", "Wind Speed and Direction Sensor", windSensor1* $\rangle$
- γ_2 = $\langle$ *plume_computation, within 200 meter radius of incident location, 8 min,* $\{$*0.5 GB,* $\dots\}$, $\{$*Linux, Python,* $\dots\}$, $\{$*wind speed, wind direction, temperature, humidity, aerosol concentration,* $\dots\}$, *composed data, "deployable", "Edge device", edgeNode1*$\rangle$

Interaction between (γ_1, γ_2) is modeled as $\lambda \in \Lambda$ as:

- λ = $\langle(\gamma_1, \gamma_2)$, $\{\langle$*response time: 100ms*$\rangle$,$\langle$*bandwidth: 1 Mbps*$\rangle$,$\langle$*sampling rate: 10 samples per sec*$\rangle$,$\dots\}$, *(http://schemas.opengis.net/sos/2.0/sosGetObservation.xsd#request, http://schemas.opengis.net/sos/2.0/sosGetObservation.xsd#response)* $\rangle$

Note that resource binding for the tasks (parameter r) cannot be specified at design time as it can only be determined at runtime after resource discovery and selection. Similarly, the values of π_{γ_1} are π_{γ_2} in their λ are also determined at runtime after task-to-resource binding.

Definition 3. *(Binding):* Let $W = (\Gamma, \Lambda)$ be a workflow and $\mathcal{R}$ be a set of available resources, a task-to-resource binding $b : \Gamma \rightarrow \mathcal{R}$ is mapping from a resource request in Γ onto a specific resource $\in \mathcal{R}$.

The binding of the task γ_1 in Example 1 to an actual wind sensor is expressed as, $b(\gamma_1) = windSensor1$, where $windSensor1$=$\langle$*S123, "Wind Speed and Direction Sensor",* $\langle$*latLong:(29.331, 70.827)*$\rangle$,$\dots$, $\{\langle$*time constant: 62.3*$\rangle$, $\langle$*baud rate: 2400 to 38400*$\rangle$,$\dots\}\rangle$.

2.1 Problem Statement

Given,

- a distributed application workflow $W = (\Gamma, \Lambda)$,
- knowledge of the operational environment,
- services available in the cloud, and
- the database maintaining the information of different edge nodes, IoT devices and services which can be deployed

Develop an orchestration and management plan that:

- determines binding of the workflow tasks to available services based on workflow specification and available knowledge of the environment, and
- dynamically adapts (context-aware) execution of the workflow by taking into account the changes in the environmental context (e.g., unavailability of resources requiring re-binding of workflow tasks) and execution results of tasks.

In knowledge-driven distributed workflow applications, subsequent tasks are dependent upon the knowledge gained from the execution of previous tasks and binding of all the tasks cannot be performed once at design time. For example, in our incident management scenario, the binding of tasks related to plume modeling and finding congestion-free routes cannot be performed simultaneously at design time. Since the execution of the plume modeling-related task identifies the at-risk areas, and this knowledge is used in finding congestion-free routes. Therefore binding needs to be done in an iterative and incremental manner.

3 Proposed Approach

In this paper, we propose two approaches for orchestration and management of distributed application workflows in the cloud and edge computing environment. The first approach involves centralized orchestration and management of workflows. In the second approach, the orchestration and management activities are delegated to selected peers in the overlay network. Each selected peer of overlay network acts as a *Local Orchestration Manager* and performs service/resource discovery, selection, and deployment for the assigned task(s) in the distributed application workflow. Below we discuss these approaches in detail.

3.1 Global Orchestration and Management (GOM)

In this approach, orchestration and management of distributed workflow applications (which includes binding, coordination, management, and adaptation) are performed in a centralized manner by the workflow coordinator. The workflow coordinator determines at runtime, when and where to perform the different workflow tasks by taking into account the task specifications, location requirements, current execution status of tasks, and resource availability.

As illustrated in the incident management workflow example (Section 1.1), workflow tasks are dependent on each other for their inputs and binding to underlying resources. For example, traffic monitoring and re-routing task can only be initiated after the at-risk areas are identified by the plume modeling task. In such cases, the workflow coordinator must also determine when to execute tasks in addition to binding the tasks to appropriate resources (i.e.,

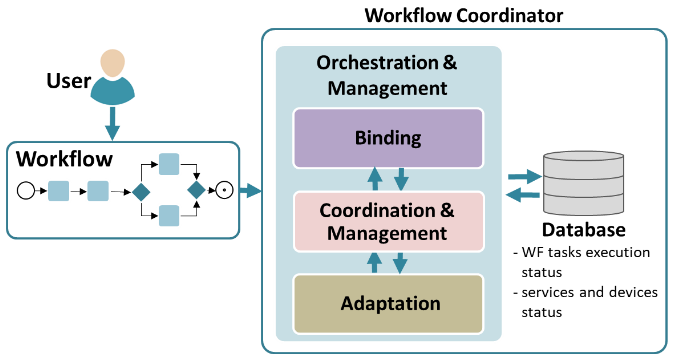
Workflow Coordinator
User
Workflow
Orchestration & Management
Binding
Coordination & Management
Adaptation
Database
- WF tasks execution status
- services and devices status

Fig. 3. Role of Workflow Coordinator in the GOM Approach

devices/services) for their execution. This is specifically true for computation services which need to be deployed on edge/ IoT resources.

Fig. 3 shows the architectural overview of the GOM approach. The workflow coordinator maintains a database to store the workflow execution status and current status information of various resources available within each cell of the peer node. For orchestration and management, workflow coordinator is responsible for resource discovery, task-to-resource binding, interaction between resources across cells for workflow coordination and management, and workflow adaptation in case of resource failure/un-availability.

Our proposed approach performs resource discovery and task binding iteratively at runtime. In each iteration, the workflow coordinator dynamically discovers resources for all those tasks whose inputs and deployment information are available and performs task binding. For example in the incident management workflow example, initially, input and deployment information is available for plume modeling related tasks only – location is provided by user at the start of the process. For plume modeling related tasks, the workflow coordinator first discovers the resources, including nearby sensors providing weather-related information, storage buffer for temporary storage and integration of sensor data, and computation resources for deployment of plume modeling service. Next, the workflow coordinator performs task-to-resource binding based on some optimality criteria. In case, resources binded to the plume modeling related tasks span multiple cells, the data exchange between these resources across cells is achieved through the workflow coordinator. Moreover, the binded resource may become unavailable (e.g., mobile device moved out of range or failed) during task execution. In such a situation, the workflow coordinator performs dynamic adaptation involving re-binding of task to a new resource.

Algorithm 1 presents the pseudo-code of the proposed GOM approach. Given, a user's workflow orchestration and management request expressed as $W = (\Gamma, \Lambda)$, Algorithm 1 works in an iterative and incremental manner to support dynamic discovery and binding of resources, coordination & management, and workflow adaptation as discussed in the following subsections.

3.1.1 Resource discovery and binding

Resource discovery involves discovering the devices and resources that can provide the functionality and data required by the unbinded tasks in the workflow for which the input

ALGORITHM 1: Global Orchestration and Management (to be executed by the central workflow coordinator)

```
Input: Workflow : $W = (\Gamma, \Lambda)$
 1: while (true) do
 2:    $\mathcal{R} \leftarrow$ Get resource status from each peer node
 3:    for each $\gamma \in \Gamma$ that are currently in execution do
 4:       $\kappa \leftarrow$ ReceiveExecutionOutput($\gamma$)
 5:    for each $\gamma \in \Gamma$ that needs to be invoked and requires
       user input do
 6:       $\mu \leftarrow$ ReceiveUserInput($\gamma$)
    {Perform resource binding for all tasks that needs to be
    invoked or for which the binded resource becomes
    unavailable}
 7:    Workflow : $(\acute{\Gamma}, \Lambda) \leftarrow$ Bind_Resources($W, \mu, \kappa, \mathcal{R}$)
 8:    for each interaction $\langle(\gamma, \acute{\gamma}), q, m\rangle \in \Lambda$ do
 9:       if (peer($\gamma$) $\neq$ peer($\acute{\gamma}$)) then
10:          ReceiveAndForward($\gamma$, peer($\gamma$), $\acute{\gamma}$, peer($\acute{\gamma}$), $m$)
```

ALGORITHM 2: Bind_Resources

```
Input: Workflow : $W = (\Gamma, \Lambda)$
Input: User input: $\mu$
Input: Execution status : $\kappa$
Input: Resource status: $\mathcal{R}$
Output: Updated Workflow $(\acute{\Gamma}, \Lambda)$
   {Perform resource binding for all tasks that need to be
   invoked or for which the binded resource becomes
   unavailable}
 1: $\acute{\Gamma} \leftarrow$ SelectResources($\Gamma, \Lambda, \mathcal{R}, \mu, \kappa$)
   {perform service deployment on selected resource if task
   requires deployment}
 2: for each $\gamma \in \acute{\Gamma}$: $\gamma$ requires service deployment do
 3:    PerformServiceDeployment($\gamma$)
 4: return $(\acute{\Gamma}, \Lambda)$
```

and deployment information are available. This essentially requires availability of up-to-date status information about the resources including, resource type (cloud service, computational device, sensor, etc.), its current location, available functions, metadata on input requirements and output, current resource utilization status (available memory, battery life, processor, etc.), and the QoS parameters (availability, cost, response time, etc.).

In the proposed GOM approach, peer nodes locally maintain the status information of resources in their respective cells by polling them at regular intervals. This information includes resource id, type, hardware and software specifications, URI, lat-long location, availability, and QoS parameters etc.

In each iteration of Algorithm 1, the workflow coordinator receives the current resource availability status from the connected peer nodes and updates its resource repository $\mathcal{R}$ (line 2). In addition, it collects the current status of all invoked tasks of the workflow (e.g., waiting for input, executing, interrupted or failed due to unavailability of binded resources, etc.) and their intermediate results (lines 3–4). This information guides the workflow coordinator to select which workflow tasks to invoke next and interacts with the user to receive any required inputs for such tasks (line 5–6). Resource binding is then performed for all tasks that need to be invoked (line 7).

Algorithm 2 outlines the steps in the resource binding procedure. Given, workflow specification (W), current execution status of workflow tasks (κ), user inputs (μ), and availability status of cloud services and edge/IoT resources ($\mathcal{R}$), the workflow coordinator first discovers the resources that meet the functional and QoS requirements of the unbinded tasks and their interaction and then performs resource selection for task-to-resource binding based on some optimality criteria (SelectResources() in line 1 of Algorithm 2). Several approaches exist in literature for optimal resource selection in a fog/edge computing environment considering different optimization criteria (including battery life, least number of disruption, longest availability duration, response time and other QoS parameters) [5], [6], [7], [8]. Any such approach can be utilized by the workflow coordinator for resource selection. In the selection process, the workflow coordinator considers only those resources which are compatible in terms of their data exchange requirements and communication protocols.

Binding may also involve deployment of data processing and/or integration services on edge/IoT resources for executing workflow tasks (e.g., data integration for plume generation task). To support service deployment, the workflow coordinator selects resources that support service deployment while considering the proximity and latency requirements of the underlying resources for task execution. In such cases, when a service needs to be deployed, URI of the task is updated after deployment. For deployment, the workflow coordinator dynamically deploys the executable code for such a service on the selected edge device through a container or virtual machine (VM) instance.

3.1.2 Coordination and management

The coordination and management component of the workflow coordinator is responsible for enabling communication/ interaction between data-dependent services that need to communicate for example, for sharing their execution results and data integration. For example, in our incident management workflow, for plume generation, wind speed, temperature, wind direction services need to communicate with computation service for their data sharing. The workflow coordinator enables this interaction by specifying the URI of each binded resource in the interaction. Ideally, resources should be able to directly communicate with each other. However, direct communication may not be possible between two interacting resources belonging to different cells since cross-cell devices may be out of range. In such cases, workflow coordinator acts as a cross-cell message relay entity – receives a message from one peer node and then forwards it to another peer node. This is depicted in lines 8–13 in Algorithm 1.

3.1.3 Adaptation

In a dynamic edge computing environment, the resources used for deployment of data integration, pre-processing, or other computation services, are often mobile and may move out of the connectivity range of interacting devices or fail during task execution. In such a situation, the computation service needs to be migrated to a new edge device that is within the connectivity range of the interacting data source (e.g., sensor). Alternatively, new data sources that are within the range of the moving host device need to be discovered and binded to the disrupted task.

The adaptation component of the proposed approach is responsible for monitoring the instantiated workflows for any exceptional conditions i.e., failure or unavailability of one or more binded resources during the workflow execution. Upon observing any exception or deviation from required QoS performance, a recovery action is triggered for workflow adaptation. This involves resource discovery, selection, and binding for each disrupted task in the workflow and re-configuring the resource end-point references for each affected interaction.

3.1.4 Discussion

In the GOM approach, the workflow coordinator has a global view of all the resources and services, from the edge infrastructure to the computation and storage places on the cloud. Therefore, after receiving the orchestration and management request of distributed application workflows (which includes multiple tasks), it takes advantage of global knowledge to perform task-to-resource binding.

The main advantage of this approach is simplicity – all the orchestration and management functions and decision-making are concentrated at a central component i.e., the workflow coordinator. Moreover, it eases the consistency concerns about resource availability and avoids conflicting decisions by providing central control over the entire system. Although the GOM approach supports efficient resource binding for tasks requiring resource selection from a large geographical area, due to its global view of resource availability, its centralized resource management approach suffers from the single point of failure and single point of congestion. Significant *communication overhead* is incurred as all connected peers report resource availability status to the workflow coordinator by sending messages.

In addition, there are *scalability concerns*. The workflow coordinator is responsible for orchestration and management of all tasks in each workflow. This may become a major performance bottleneck in case of high workloads when several workflows need to be managed simultaneously.

To address these problems, we propose the *Local Orchestration and Management* approach that distributes the overall workload among edge devices (peer nodes) in a context-aware manner as discussed in the following section.

3.2 Local Orchestration and Management (LOM)

In this approach, orchestration and management of distributed workflow application is performed in a distributed manner by different collaborating peer nodes. The central workflow coordinator in the LOM approach is only responsible for coordination and management and assignment of workflow tasks to the peer nodes. Unlike the GOM approach, the coordinator is only aware of the peer nodes and has no knowledge about the resource availability within the cells. Each peer node has a local orchestration and management component that supports binding, coordination within cell, and adaptation of the tasks at cell level.

The workflow coordinator maintains information about the different peer nodes including their availability status, task assignment, and the execution status of different tasks in the received workflow orchestration and management requests. When the user submits a workflow orchestration and management request, the workflow coordinator assigns the different tasks in the given workflow to suitable peer nodes considering the location and contextual requirements of the tasks. Peer nodes are then responsible for task-to-resource binding. Peer nodes in LOM approach, continuously interact with workflow coordinator for handling orchestration and management requests, sharing their status and execution results of assigned tasks.

ALGORITHM 3: Coordination and Management in Local Orchestration and Management (to be executed by the central workflow coordinator)

```
Input: Workflow : $W = (\Gamma, \Lambda)$
 1: $A_w \leftarrow \varnothing$
 2: while (true) do
 3:     $\mathcal{P} \leftarrow$ Get status information of peer nodes
 4:     for each $\gamma \in \Gamma$ that are currently in execution do
 5:         $\kappa \leftarrow$ ReceiveExecutionOutput($\gamma, peer(\gamma)$)
 6:     for each $\gamma \in \Gamma$ that needs to be invoked do
 7:         if $\gamma$ requires user input then
 8:             $\mu \leftarrow$ ReceiveUserInput($\gamma$)
 9:         if $\gamma$ requires binding to a cloud service then
10:             $\gamma.r \leftarrow$ SelectCloudService($\gamma$)
11:             $A_w$.updateAssignment($\gamma, c$)
12:         else
13:             $(\gamma, p) \leftarrow$ SelectPeerNode($\gamma, W, \mathcal{P}, A_w$)
14:             p.assignTask($\gamma, W, \mu, \kappa$)
15:             $A_w$.updateTaskAssignment($\gamma, p$)
```

ALGORITHM 4: Local Orchestration and Management (to be execute by peer nodes)

```
Input: Workflow : $W = (\Gamma, \Lambda)$
Input: Assigned Tasks : $A_\Gamma \subseteq \Gamma$
Input: User input: $\mu$
Input: Execution status : $\kappa$
Output: Updated Workflow $(\acute{\Gamma}, \Lambda)$
 1: while (true) do
 2:     $\mathcal{R} \leftarrow$ Receive status of each resource in the cell
 3:     $(\acute{\Gamma}, \Lambda) \leftarrow$ BindResources($W, \mathcal{R}, \mu, \kappa$)
 4:     $\acute{\Lambda} \leftarrow \{\langle(\gamma,\ \acute{\gamma}),\ q,\ m\rangle |$ either $\gamma$ or $\acute{\gamma} \in A_\Gamma\} \subseteq \Lambda$
 5:     for each interaction $\langle(\gamma, \acute{\gamma}), q, m\rangle \in \acute{\Lambda}$ do
 6:         if (peer($\gamma$) $\neq$ peer($\acute{\gamma}$)) then
 7:             SendMessage($m$)
 8: return $(\acute{\Gamma}, \Lambda)$
```

Algorithm 3 outlines the role of the workflow coordinator in the LOM approach when it receives a new workflow orchestration and management request. Given a user workflow request, W, the workflow coordinator interacts with peer nodes in the overlay network to get their current status information including their current location and QoS parameters (line 3). This information is used for selection of appropriate peers for workflow task assignment. As in GOM, workflow coordinator keeps track of the current execution status of each task in the workflow to decide which tasks should be scheduled next and where they need to be executed. For tasks that require binding to cloud services, the workflow coordinator performs selection and binding itself (lines 9—11). However, for the tasks which require binding to edge/ IoT resources for their execution, workflow coordinator selects peer nodes for orchestration and management of the task (lines 13). For selection, workflow coordinator considers the location information given in the workflow specification or infers it from the execution results of preceding tasks. If multiple peer nodes satisfy the location requirement, the workflow coordinator may perform selection of a node satisfying closest proximity criterion and/or the QoS characteristics of the nodes. The task is then assigned to the selected peer node for binding and instantiation (line 14).

As discussed, task orchestration and management is managed by peer. When a task is assigned to a peer node, it performs task-to-resource binding (Algorithm 2) based on the current availability status of resources in its cell.

3.2.1 Adaptation

After resource binding, local orchestration manager initiates monitoring of the assigned task(s). If it observes any exceptions such as binded resource for a task becomes unavailable or a task deadline is missed, the local orchestration manager re-selects another resource, performs binding, and notifies any other peers involved for data sharing.

Algorithm 4 outlines the functionality of the Local Orchestration Manager running at peer nodes. The local orchestration manager continuously monitors the current status of devices in its cell (line 2) and performs resource binding for all assigned tasks which are unbinded or their previously binded resource is no more available (line 3). To enable data sharing among tasks spanning multiple cells, peer nodes directly communicate with each other by sending messages (lines 5–7). Note that in the LOM approach orchestration and management of multiple tasks are performed simultaneously by different peer nodes. If various tasks have the same spatial requirements, then the workflow coordinator assigns all such tasks to a single peer node as long as the QoS conditions are satisfied.

3.2.2 Discussion

LOM approach brings additional complexity to the system in terms of error recovery and fault handling. It also introduces huge storage overhead for Local Orchestration Managers. The main advantages of the LOM approach are load balancing and scalability. The responsibilities of the central workflow coordinator are shared between multiple Local Orchestration Managers running at peer nodes.

LOM approach also incurs significant *communication overhead* when a given workflow task requires resources from multiple cells. For example, plume modeling task in the incident management workflow may require sensory data at different points of observation within a large area spanning multiple cells. For such tasks, multiple peers need to communicate with each other and lookup resources in different cells). However, a major messaging overhead incurred by the GOM approach for resource status sharing with the central workflow coordinator by the peers is avoided in the LOM approach.

Peer nodes in LOM approach have knowledge of available resources within their own cells, therefore locally optimal resource selection by the peers does not guarantee that the selection of resources for the entire application workflow is also globally optimal.

4 Implementation

The implementation in Java J2EE for both GOM and LOM approaches includes the following four components:

1) *Workflow Authoring Interface* is a web-based component developed in HTML5 and JavaScript which includes a graphical user interface that allows the user to specify workflow requirements in terms of tasks and their interactions. This component also includes an interface for registering cloud services, edge nodes, IoT devices, and services that can be deployed.
2) *Workflow Coordinator Component* implements both global and local workflow orchestration and management algorithms for resource selection, binding, and messaging with peer nodes and end devices.
3) *Peer Node Component* implements the proposed algorithms for global and local orchestration and management at peer devices. In addition, it includes the mechanism for managing the overlay network of peer nodes and messaging between the different devices using Apache Paho and Moquette.
4) *End Device Component* is installed on IoT devices to communicate with the workflow coordinator and peer nodes.

All these components are provided as executable JAR files for deployment on devices equipped with Java Runtime Environment (JRE) and MySQL server. Moquette MQTT broker and Apache Paho clients are used for managing the overlay network of peer nodes and messaging between the different devices. Moquette is a Java-based broker of the Message Queue Telemetry Transport (MQTT) light-weight messaging protocol. Apache Paho is an open-source client implementation of MQTT and MQTT-SN messaging protocols for IoT.

The source code of all components has been made available as Eclipse Java Projects in a single GitHub repository along with the instructions for downloading and setting up the proposed solution[1].

For experimental evaluation (discussed in Section 5), we hosted the workflow coordinator on an Intel Core i7 machine (3.60 GHz processor; 16GB RAM) equipped with JRE, and MySQL server. For peer nodes, we used Raspberry Pi 3 devices with Raspbian operating system, JRE, and MySQL DBMS. The peer nodes are connected with several sensors/IoT devices (e.g., smartphones, tablets, temperature, and other environmental sensors) for providing sensor, computation and data services.

5 Experimental Evaluation

We evaluate the performance of the proposed approaches by considering an incident management workflow, specifically focusing on plume modeling and traffic monitoring activities. We also compare the proposed approaches with the existing state-of-the-art ECStream Processing (ECSP) baseline approach [9] on a license plate recognition workflow, which though not a knowledge-driven workflow, is a standard benchmark used in the literature.

The performance of the proposed approach is measured using the following metrics:

1. https://github.com/aakhtar4/IoT-Centric-Workflows-Orchestration

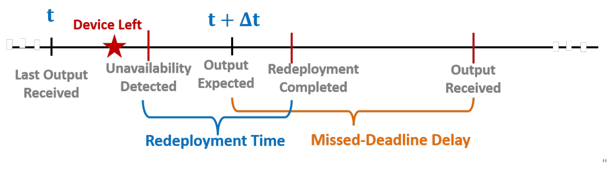

Fig. 4. Delays dues to deadline misses and redeployment of tasks.

- **Deadline Misses:** If the output of a given task is not generated by its associated deadline we refer it to as a deadline miss. Any subsequent dependent tasks are also delayed in case of a deadline miss.
- **Redeployment Time:** When a coordinator discovers that a resource with an assigned task has become unavailable, it performs re-selection, re-binding, and re-deployment to execute the disrupted task at a new resource. This re-selection, re-binding, re-deployment, and restarting of the task at the new device constitutes the *redeployment time*. Figure 4 elaborates on the delays caused due to deadline misses and task redeployment. Here Δt denotes task duration i.e., the expected time for the given task to generate its output.
- **Communication Overhead:** Communication overhead is determined in terms of messages exchanged between the workflow coordinator, peer nodes, and end devices involved in workflow execution. These messages are related to task assignments, task execution status, and resource status updates.

5.1 Evaluation on Incident Management Workflow

The incident management workflow involves five key activities including, plume modeling, plume visualization, vehicle tracking, traffic flow monitoring, and vehicle re-routing as shown in Figure 5. The plume modeling activity involves two key tasks i.e., *data buffering & integration* task and a *plume model generation* task. We are getting sensory data observations from twenty-five different sensors (wind speed, wind direction, temperature, humidity, and aerosol concentration). The sensors are located in three different cells near the incident site. All these sensors are sending their data with different granularity to *data buffering & integration service* deployed in the proximity of incident location. The integrated data is fed to a cloud service for *plume model computation*. Plume modeling service receives integrated data at fixed intervals and generates the plume. Workflow coordinator imposes the resulting plume model on Google Map for visualization and identifying at-risk areas. *Vehicle tracking cloud service* is then invoked by the workflow coordinator to track and re-route any vehicles currently within the identified at-risk region or heading towards it. For re-routing, the workflow coordinator selects the alternate routes through *Google Map Direction Service*. *Traffic flow monitoring* activity is executed to monitor traffic flow and possibility of congestion on the selected alternate routes. Live camera feed of 12 cameras at 3 intersection points (3 cells) is obtained (10 second video clips) and buffered by data buffering and integration service. Each video clip is processed by traffic flow monitoring services deployed on three peer devices near the intersection points. If an alternate route is found congested, the workflow coordinator selects another route and re-routes the vehicles to the less congested route. This workflow continuously executes. Whenever the plume model indicates a change in at-risk region, subsequent tasks in the workflow are performed as per the new region identified.

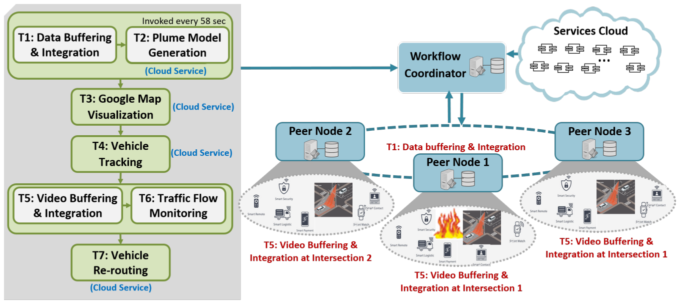
Invoked every 58 sec
T1: Data Buffering & Integration
T2: Plume Model Generation
(Cloud Service)
T3: Google Map Visualization
(Cloud Service)
T4: Vehicle Tracking
(Cloud Service)
T5: Video Buffering & Integration
T6: Traffic Flow Monitoring
T7: Vehicle Re-routing
(Cloud Service)
Workflow Coordinator
Services Cloud
Peer Node 2
Peer Node 3
Peer Node 1
T1: Data buffering & Integration
T5: Video Buffering & Integration at Intersection 2
T5: Video Buffering & Integration at Intersection 1
T5: Video Buffering & Integration at Intersection 1

Fig. 5. Incident Management Workflow Implementation

The environment we considered for Incident Management Workflow comprises 20 peer nodes directly connected to the Workflow Coordinator. In GOM configuration, the Workflow coordinator is responsible for maintaining the status of all peer nodes, as well as each of the 20 devices connected to each peer nodes i.e., 400 end devices. Workflow coordinator is notified about the updated status of devices after every 2 seconds. In LOM configuration, workflow coordinator is only maintaining the status of 20 peer nodes. Each peer is responsible for maintaining the status of its connected 20 end devices. The availability of devices in the cells is modeled using exponential distribution where the average residence time of each end device in the cell is assumed to be 5 minutes. As discussed above, in the incident management workflow, the data buffering service that feeds to the plume modeling service is deployed on the peer operating in the cell of the incident site however, the sensory observations are collected from sensors in three cells. Instances of video buffering and integration service feeding the traffic flow monitoring service are deployed on 3 peer nodes operating in the cells of 3 intersection points for traffic monitoring.

This streaming workflow is executed for 60 minutes. Selection, binding, and service deployment are performed iteratively whenever any of the devices executing a workflow task becomes unavailable. The availability of devices in the cells is modeled using exponential distribution considering an average residence time of devices in a cell to be 5 minutes.

5.1.1 Evaluation Results

Table 1 presents the experimental evaluation results of plume modeling and traffic flow monitoring activities for incident management for both GOM and LOM approaches. We executed this workflow for 60 minutes.

Plume Modeling Activity. We considered a 58 seconds deadline for this workflow i.e., the plume model is expected to be updated every 58 seconds. The data buffering & integration task feeding plume modeling cloud service was hosted and executed on an end device in one cell however, it required sensory observations from three different cells. Rebinding (selection, mapping, and service deployment) of a new device is performed for the data buffering & integration service if its host device becomes unavailable. Similarly, if

TABLE 1
Experimental Evaluation of Incident Management Workflow

Average Residence Time of devices in a cell: 5 min.				
Activity	**Plume Modeling**		**Traffic Monitoring**	
Parameters	**GOM**	**LOM**	**GOM**	**LOM**
No. of Deadline Misses	22 (out of 62)	23 (out of 62)	45 (out of 270)	43 (out of 270)
Total Delay due to Deadline Misses	169.83 sec	207.44 sec	126 sec	111 sec
Total Redeployment Time	90.66 sec	105.57 sec	145.35 sec	132.65 sec
Total No. of Messages Exchanged for Redeployment	176	276	360	258

a source sensor becomes unavailable, re-selection and remapping of a new sensor are performed. For the 60-minute long workflow, there are 62 deadlines given that an output needs to be generated after every 58 seconds.

In the GOM setting, 22 deadline misses were observed i.e., 35%. Average delay for the activity was 7.72 seconds and the average redeployment time was 4.12 seconds. Redeployment of the data buffering & integration task or replacement of a sensor in GOM configuration required an exchange of 8 messages. A total of 176 messages were exchanged in this case for 22 device failures/ unavailability (host device/ sensor).

In the LOM setting, we observed 23 deadline misses i.e., 37%. Average delay due to deadline misses was 9.02 seconds and the average redeployment time was 4.59 seconds. Each redeployment required an exchange of 12 messages. This includes messaging between peer nodes to look up sensory devices in three cells involved in environmental data collection. A total of 276 messages were exchanged to handle 23 device failures/ unavailability.

Traffic flow monitoring activity. We considered a 40 seconds deadline for this activity. A video buffering & integration task was hosted and executed in parallel on three devices in different cells. For the 60-minute long workflow, there is a total of 90 deadlines per device considering that an output needs to be generated after every 40 seconds.

In the GOM setting, 45 deadlines were missed i.e., 24%. Average delay for the task was 2.79 seconds and average redeployment time was 3.23 seconds. Each redeployment in this case required exchange of 8 messages. A total of 360 messages were exchanged for 45 redeployments.

In the LOM setting, we observed 43 deadline misses for this task i.e., 23%. Average delay due to deadline misses was 2.58 seconds and average redeployment time was 3.08 seconds. Redeployment of the traffic flow monitoring-related tasks required an exchange of 6 messages. A total of 258 messages were exchanged in this case for 43 redeployments.

The results in Table 1 depicts that for the plume modeling activity, GOM performed better than LOM in terms of deadline misses, total delay, redeployment time, and communication overhead. However, for the traffic monitoring activity, LOM performed better than GOM on all four metrics. The reason is that in the plume modeling activity, there is a higher degree of cross-cell data dependency. Specifically,

TABLE 2
Effect of cross-cell data dependence on performance of GOM and LOM for the Plume Modeling Activity

Average residence time of devices in a cell: 5 min.								
Cross-cell data dependence ratio	10:1		8:1		4:1		0:1	
	GOM	LOM	GOM	LOM	GOM	LOM	GOM	LOM
No. of deadline misses (out of 62)	22	23	20	20	18	19	16	16
Total Delay due to Deadline Misses	169.83 sec	207.44 sec	133.76 sec	180.76 sec	101.42 sec	138.82 sec	41.97 sec	34.47 sec
Total redeployment time	90.66 sec	105.57 sec	71.72 sec	84.83 sec	45.114 sec	72.32 sec	24.08 sec	20.28 sec
Total No. of Messages Exchanged for Redeployment	176	276	160	240	144	228	128	96

the data buffering and integration (DBI) service deployed in one cell required sensory data observations from multiple sensors in three different cells (5 sensors from each cell). Whereas, there was no cross-cell data dependence in case of the traffic monitoring activity. When there is a higher degree of cross-cell data dependence, the central coordinator having knowledge of the availability status of all devices incurs lesser overhead in terms of message exchanges and redeployment time when devices become unavailable. Thus, resulting in fewer deadline misses and lesser delay. In the case of LOM, peers need to communicate with each other to share device status of their respective cells when there is cross-cell data dependency as discussed in Section 3.2.2.

In order to confirm this hypothesis, we carried out additional experiments to evaluate the effect of cross-cell data dependence. Table 2 shows the performance of both GOM and LOM for the plume modeling activity for a number of cross-cell data dependence settings. Specifically, we considered three cells for the plume modeling activity, each having an equal number of sensors feeding data to the DBI service (which was hosted on a device in one of these three cells). The degree of cross-cell data dependence is measured based on the number of sensors that provide data to the DBI service in a different cell. For example, if 5 sensors in each of the 3 cells provide data to the DBI service then the degree of cross-cell dependence is 10:1 because there are 10 sensors in 2 cells (Cell-1 and Cell-2) that provide data to the DBI service hosted in Cell-3. The remaining 5 sensors are in the same cell of the DBI service. Similarly, the cross-cell data dependence of 0:1 implies no data sharing between cells for task execution, though there may be data dependence within a cell. In Table 2, the results for 0:1 data dependence are computed considering 2 sensors within the same cell providing data to DBI service.

The results depicted in Table 2 confirm that the degree of cross-cell dependence significantly affects the performance, and the gap between the two approaches increases as the degree of cross-cell data dependence increases. Thus, while the LOM approach outperforms GOM when there is no or very low cross-cell data dependence, for even medium cross-cell data dependence (4:1), GOM starts outperforming LOM. This also highlights the need for a hybrid approach that allows workflow orchestration to switch from GOM to LOM and vice versa as the degree of cross-cell data dependence changes during workflow execution. Note that such cross-cell data dependence may not be predicted in advance for knowledge-driven workflows as discussed in the introduction. While this is not considered further in this work, we plan to examine this in the future.

5.2 Evaluation on License Plate Recognition Workflow

The License plate recognition workflow comprises of Video Decoding, Object Detection, and License Plate Recognition tasks as described below and depicted in Fig. 6.

Video Decoding. This task takes a video as input, partitions it into 10-second *clips*, and then extracts frames (FFmpeg images) from each clip. Each clip is divided into 30 frames which are fed to the object detection task.

Object Detection. This task takes frames as input and provides detected car images as output. In our implementation, we have utilized an object detection service that employs a deep neural network model, YOLOv3. It detects objects from frames, extracts the images with recognized vehicles, and stores them. These object images are then transferred to the device running the license plate recognition task.

License Plate Recognition. This task takes vehicle images as input and performs detection and recognition of the license plate by invoking the OpenALPR library functions. It returns the list of license plates with a confidence value.

The configuration we considered, in this case, consists of the workflow coordinator, 3 Raspberry Pi peer nodes (1 peer device per cell), and 2 end devices connected per peer device. Note that the workflow coordinator and all the devices are located on the same WLAN. Average residence times of devices in a cell are considered to be 5, 7.5 minutes, and 10 minutes.

Each workflow task is deployed on a separate peer device such that video processing is performed in a clip-by-clip manner. Peer node 1 performs video decoding and sends extracted frames to peer node 2. Peer node 2 performs object detection from the received frames and then shares the images with peer node 3. Peer node 3 performs a license plate recognition task.

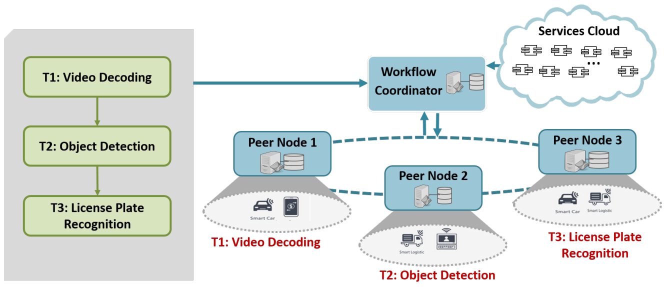

Fig. 6. License Plate Recognition Workflow Implementation

For both GOM and LOM approaches, the coordinator (global/local) is updated after every 5 seconds with the current state of the connected devices. If the coordinator

TABLE 3
Experimental Evaluation of License Plate Recognition Workflow (considering task execution in three cells)

Parameters	**Avg. Residence Time: 5 min.**		**Avg. Residence Time: 7.5 min.**		**Avg. Residence Time: 10 min.**	
	GOM	**LOM**	**GOM**	**LOM**	**GOM**	**LOM**
No. of Deadline Misses (out of 30)	29	26	28	25	26	23
Total Delay due to Deadline Misses	181 sec	153 sec	171 sec	144 sec	158 sec	131 sec
Total Redeployment Time	223.01 sec	162.76 sec	220.64 sec	153.51 sec	200.31 sec	145.13 sec
Total No. of Messages Exchanged for Redeployment	174	104	168	100	156	92

TABLE 4
Comparative Evaluation Results for ECSP, GOM, and LOM Approaches for License Plate Recognition Workflow (Single cell configuration).

Parameters	**Avg. Residence Time: 5 min.**			**Avg. Residence Time: 7.5 min.**			**Avg. Residence Time: 10 min.**		
	GOM	**LOM**	**ECSP**	**GOM**	**LOM**	**ECSP**	**GOM**	**LOM**	**ECSP**
No. of Deadline Misses (out of 30)	29	26	24	28	25	23	26	23	22
Total Delay due to Deadline Misses	145 sec	112 sec	100 sec	137 sec	106 sec	95 sec	126 sec	97 sec	90 sec

does not receive a heartbeat message after 5 seconds, it assumes that the device has left.

5.2.1 Evaluation Results

The experiments for license plate recognition workflow were conducted using a Dashcam stream captured during a ride through London downtown in HD quality of 1,920×1,080 pixels. The workflow had a total of 30 deadlines for processing the video at 3 peer devices. Table 3 presents the evaluation results. First, we note that there is very little cross-cell dependence in this workflow (1:1), and therefore as supported by our prior analysis, the LOM approach outperforms the GOM approach on all 4 metrics. We also wanted to examine the impact of the average residence time of devices within each cell on the different metrics. As expected, the number of deadline misses and average delay decreased as we increased the average residence time. Note, however, that the degree to which LOM outperforms GOM does not change with respect to the average residence time. Redeployment of a task in this workflow had an associated communication overhead of 6 message exchanges in the GOM configuration and 4 message exchanges in the LOM configuration. It can be observed that in this case, the total number of messages exchanged for handling redeployments was higher for the GOM approach as compared to LOM. This is because of the additional messaging between the central workflow coordinator and local orchestration managers for resource redeployment in GOM. Note that each task in this workflow was performed within one cell therefore, cross-cell communication for this workflow only involves data sharing between peers.

5.3 Comparison with ECStream Processing Approach

ECSP [9] is a decentralized approach that extends the functionality of Apache NiFi stream processing middleware to provide support for dynamic clustering of edge devices to enable dynamic deployment of workflow tasks on the clustered edge devices. The idea is to provide a shared pool of resources via clustered edge devices for parallel processing of workflow tasks. These clustered devices act as both cluster coordinators and worker nodes.

ECSP approach comprises of the following steps: (i) First, *Clusterization of edge devices* is performed. Clusterization is initiated by an edge node that needs to offload a computational task to peers by broadcasting offloading requests to edge devices in its range (cell). Upon receiving a request, eligible devices in the cell decide whether to participate; (ii) Device *selection* is then performed by the initiator/ coordinator node by taking into account the QoS constraints and mobility patterns; (iii) *Placement and Configuration* is then performed by assigning tasks to selected nodes by sending configuration parameters. A customized ZooKeeper Placement & Orchestration module is configured on each device for cluster establishment and configuration. The cluster then executes the deployed tasks in parallel on its nodes.

Prototype configuration. For the prototype setup of the ECSP approach, Apache Nifi and Zookeeper were configured on 5 Raspberry Pi devices. Executable instances of ECSP Cluster middleware with customized Nifi processors were also deployed on all devices. All these devices were located on the same WLAN to trigger the ECSP clusterization process. For these experiments, we considered the License Plate Recognition workflow which was executed by ECSP with task-level parallelism on an edge cluster created by five devices. In the given setup, Zookeeper allocated the tasks to all available devices since all devices are capable of task offloading. Note that ECSP does not support cross-cluster communication therefore for comparison we only considered all three tasks executing in a single cell.

For comparison with ECSP, the setup we considered for GOM and LOM approaches comprised a server machine acting as the central workflow coordinator, and 6 Raspberry Pi devices including, one peer node and five end devices.

The evaluation metrics we have considered for this comparison are the number of deadline misses and the associated delay. Since we are unable to instrument the ECSP approach, we could not compute the redeployment times and the total number of messages exchanged.

5.3.1 Results & Discussion.

It can be observed from Table 4 that ECSP outperforms both GOM and LOM in terms of the average delay due to deadline misses for all three residence times. However, we stress that the License Plate Recognition Workflow utilizes a single-cell configuration since ECSP does not support cross-cell coordination. Specifically, any tasks that span across multiple cells cannot be accomplished by ECSP due to the absence of central coordination and lack of interaction among the edge clusters. Whereas in GOM, peers communicate with each other via the central coordinator, and in LOM via direct messaging. This is exactly why the ECSP approach cannot be utilized at all for the incident management workflow where the plume modeling activity requires cross-cell interaction. Finally, we note that even when there is no cross-cell data dependence both LOM and GOM require roughly comparable (though more) time, while having the ability to deal with cross-cell data dependence, if needed.

6 RELATED WORK

We discuss the related works in the context of distributed workflow composition, distributed application deployment & container migration, and serverless computing.

6.1 Distributed Workflow Composition

Most works on distributed workflow composition and adaptation are based on AI planning approaches. Given a composition goal, these approaches generate a plan [10], [11]. Chen et al. proposed GoCoMo [10], a goal-driven service composition framework in a mobile computing environment. GoCoMo leverages a decentralized planning algorithm based on backward chaining to support goal-driven service discovery with opportunistic service execution and QoS-based heuristic discovery checking. Moeini et al. [11] proposed a decentralized planning technique based on Graphplan that given a goal, achieves composition of distributed services by building a logic-based overlay network that models the logical relationships of IoT services. While some of the planning-based approaches e.g., GoCoMo [10] support dynamic adaptation due to mobility/ unavailability of binded devices or changes in QoS parameters, these approaches are not designed to support data analytics and streaming workflows that require edge devices to pre-process and integrate data from multiple sensors or IoTs.

Other related works on service selection and task assignment involve worklfow graph partitioning and optimization-based approaches [12], [13], [14], [15], [16]. Ko et al. proposed the SoIoT framework [17] to address the problem of selection, composition, and delivery of user-centric services in IoT environment. They used a graph coloring-based heuristic for optimal service assignment to IoT devices. Lera et al. in their work on availability-aware service placement [16] proposed a two-phase partitioning-based optimization algorithm. The idea is to use a combination of complex network communities for device partitioning and service transitive closures for application graph partitioning with the objective to optimize service availability and QoS. Renart et al. proposed a programming framework for dynamic data stream processing leveraging edge computing resources [18]. This programming framework builds on a stream-processing overlay network that coordinates the execution of the distributed streaming application workflows using a publish/subscribe messaging model to facilitate content-based interactions among workflow participants. A location-aware protocol has been used for efficient allocation of streaming computation at the edges of the infrastructure along the data path from data source to consumer. Our proposed approach uses a similar idea of an overlay network for orchestration and management of location-aware knowledge-driven workflows which are both emergent and evolvable as opposed to the streaming workflows considered in above mentioned works.

6.2 Distributed Application Deployment and Container Migration

Distributed application deployment is a challenging problem in a cloud and edge computing environment where devices are resource-constrained, geographically distributed, and highly dynamic. Several works have been carried out in the past few years to propose optimal service placement considering different placement characteristics, deployment constraints (resource capabilities, network constraints, and application requirements such as delay sensitivity and locality), and strategies [8], [19], [20], [21], [22], [23], [24]. It is worth noting that optimal service placement approaches rely on a central entity having complete knowledge of the infrastructure and all application components for decision-making on task offloading/ service placement. Our proposed GOM approach also relies on global knowledge and can employ any of these existing approaches for optimal task assignment, such centralized approaches have scalability-related concerns. While decentralized placement strategies improve scalability, in general, such strategies cannot achieve a global optimum [25].

Decentralized approaches to handle the scalability concern consider a limited geographic region (cell) for application deployment and cluster the resources within that region to achieve scalability and load balancing [9], [25], [26]. Existing resource clustering-based approaches consider a restricted application topology, where applications are monolithic or all tasks are executed in a single cluster / geographic cell [9], [16], [25], [27], [28]. Furthermore, most of the approaches consider the mobility of end-user devices [29], not the edge devices and IoT resources, and are not designed to support cross-cluster coordination and data exchange among workflow tasks [19], [25]. In the distributed workflows we are considering, tasks may need to be executed in multiple geographic regions due to the location requirements of the workflow and changes in the operational environment due to intermittent availability/ mobility of edge and IoT devices.

6.3 Serverless Computing

Serverless computing paradigm enables development of applications decomposed into microservices and deploying these microservices to containers in the cloud edge continuum. A microservice essentially implements certain functions of the application thus, serverless computing can be viewed as Funtion-as-a-Service (FaaS). Recently, a lot of attention has been given to using FaaS model to develop advanced IoT applications with strict QoS requirements [30], [31]

Garbugli et al. proposed an IoT-oriented middleware [30] for management and execution of application functions on the virtualized resources in the edge cloud continuum based on the application's QoS requirements. Puliafito et al. proposed a framework for composition and execution of stateful and location-aware applications with realtime processing and latency constraints in the FaaS model [31]. Specifically, the proposed framework enables functions deployed on edge nodes to dynamically migrate between remote state storage and local state storage depending on the latency and bandwidth requirement of the application which may keep on changing during execution. Barcelona-Pons et al. proposed Crucial [32], a system to program highly-parallel stateful serverless applications such as machine learning and scientific computing workflows. Crucial allows to port a multi-threaded code base to FaaS platform for improving scalability and reducing infrastructure cost.

These middleware solutions discussed above are primarily designed for allocation and provisioning of virtualized resources/ containers in the edge cloud continuum and do not consider the application semantics and their contextual dependencies. Our proposed framework which is designed for knowledge-driven workflow applications can leverage these middleware solutions for management of QoS and application latency constraints.

7 Conclusions and Future Work

In this paper, we presented a centralized as well as a distributed approach for dynamic orchestration and management of IoT-centric distributed workflow applications. Both approaches are designed for knowledge-driven business process workflows that are adaptive, interactive, evolvable, and emergent. These approaches provide support for location-aware and flexible service/resource selection, iterative/incremental binding, and dynamic service deployment by taking into account resource availability status, spatio-temporal constraints, and resource constraints of the underlying service infrastructure as well as real-time processing constraints of the application. A comprehensive experimental evaluation performed by emulating two real-time data streaming workflow examples shows the effectiveness of our proposed approaches.

In the future, we plan to work on several underlying technical challenges including, building capabilities for fault tolerance and resilience in orchestration and management of knowledge-driven workflows, and the security & privacy concerns of collaborating parties.

Acknowledgments

Research reported in this publication was supported by the Pakistan Higher Education Commission (HEC) NRPU Grant $P\#14601$, the National Science Foundation under awards CNS-1624503 and CNS-1747728, and the National Institutes of Health under award R35GM134927. The content is solely the responsibility of the authors and does not necessarily represent the official views of the agencies funding the research.

References

[1] H. Mei, G. Huang, and T. Xie, "Internetware: A Software Paradigm for Internet Computing," *Computer*, vol. 45, no. 6, pp. 26–31, 2012.

[2] O. Almurshed, O. Rana, Y. Li, R. Ranjan, D. N. Jha, P. Patel, P. P. Jayaraman, and S. Dustdar, "A fault-tolerant workflow composition and deployment automation iot framework in a multicloud edge environment," *IEEE Internet Computing*, vol. 26, no. 4, pp. 45–52, 2021.

[3] A. Elahraf, A. Afzal, A. Akhtar, B. Shafiq, J. Vaidya, S. Shamail, and N. R. Adam, "A framework for dynamic composition and management of emergency response processes," *IEEE Transactions on Services Computing*, vol. 15, no. 4, pp. 2018–2031, 2022.

[4] NOAA, "ALOHA: Areal Locations of Hazardous Atmospheres 5.4.4," Available at https://response.restoration.noaa.gov/sites/default/files/ALOHA_Tech_Doc.pdf, National Ocean Service, Office of Response and Restoration, Seattle, Washington, Tech. Rep., 2013 (Accessed: 2020/06/06).

[5] A. Brogi and S. Forti, "Qos-aware deployment of iot applications through the fog," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1185–1192, 2017.

[6] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[7] S. Wang, M. Zafer, and K. K. Leung, "Online placement of multi-component applications in edge computing environments," *IEEE Access*, vol. 5, pp. 2514–2533, 2017.

[8] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments," *ACM Transactions on Internet Technology (TOIT)*, vol. 19, no. 1, p. 9, 2018.

[9] R. Dautov and S. Distefano, "Stream processing on clustered edge devices," *IEEE Transactions on Cloud Computing*, vol. 10, no. 2, pp. 885–898, 2022.

[10] N. Chen, N. Cardozo, and S. Clarke, "Goal-driven service composition in mobile and pervasive computing," *IEEE Transactions on Services Computing*, vol. 11, no. 1, pp. 49–62, 2016.

[11] H. Moeini, I.-L. Yen, and F. Bastani, "Decentralized service discovery and composition in dynamic iot systems," in *2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2021, pp. 130–137.

[12] D. Zhao, Z. Zhou, P. C. K. Hung, S. Deng, X. Xue, and W. Gaaloul, "CTL-Based Adaptive Service Composition in Edge Networks," *IEEE Transactions on Services Computing*, pp. 1–14, 2022.

[13] S. Berrani, A. Yachir, B. Djamaa, S. Mahmoudi, and M. Aissani, "Towards a new semantic middleware for service description, discovery, selection, and composition in the internet of things," *Transactions on Emerging Telecommunications Technologies*, p. e4544, 2022.

[14] A. G. Neiat, A. Bouguettaya, and M. Bahutair, "A deep reinforcement learning approach for composing moving iot services," *IEEE Transactions on Services Computing*, vol. 15, no. 5, pp. 2538–2550, 2021.

[15] Z. Wang, B. Cheng, W. Zhang, and J. Chen, "Many-objective automatic service composition based on temporal goal decomposition," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3813–3828, 2021.

[16] I. Lera, C. Guerrero, and C. Juiz, "Availability-aware service placement policy in fog computing based on graph partitions," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3641–3651, 2018.

[17] I.-Y. Ko, H.-G. Ko, A. J. Molina, and J.-H. Kwon, "SoIoT: Toward a user-centric IoT-based service framework," *ACM Transactions on Internet Technology (TOIT)*, vol. 16, no. 2, p. 8, 2016.

[18] E. G. Renart, J. Diaz-Montes, and M. Parashar, "Data-driven stream processing at the edge," in *2017 IEEE 1st International Conference on Fog and Edge Computing*. IEEE, 2017, pp. 31–40.

[19] F. A. Salaht, F. Desprez, and A. Lebre, "An overview of service placement problem in fog and edge computing," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–35, 2020.

[20] M. Goudarzi, M. Palaniswami, and R. Buyya, “Scheduling iot applications in edge and fog computing environments: a taxonomy and future directions,” *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–41, 2022.
[21] J. Liu, G. Li, Q. Huang, M. Bilal, X. Xu, and H. Song, “Cooperative resource allocation for computation-intensive iiot applications in aerial computing,” *IEEE Internet of Things Journal*, 2022.
[22] V. Farhadi, F. Mehmeti, T. He, T. F. La Porta, H. Khamfroush, S. Wang, K. S. Chan, and K. Poularakis, “Service placement and request scheduling for data-intensive applications in edge clouds,” *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 779–792, 2021.
[23] C. Pham, D. T. Nguyen, Y. Njah, N. H. Tran, K. K. Nguyen, and M. Cheriet, “Share-to-run iot services in edge cloud computing,” *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 497–509, 2021.
[24] A. Orive, A. Agirre, H.-L. Truong, I. Sarachaga, and M. Marcos, “Quality of service aware orchestration for cloud–edge continuum applications,” *Sensors*, vol. 22, no. 5, p. 1755, 2022.
[25] Z. A. Mann, “Decentralized application placement in fog computing,” *IEEE Transactions on Parallel and Distributed Systems*, 2022.
[26] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, “Optimized iot service placement in the fog,” *Service Oriented Computing and Applications*, vol. 11, no. 4, pp. 427–443, 2017.
[27] L. Baresi, D. F. Mendonça, and G. Quattrocchi, “Paps: A framework for decentralized self-management at the edge,” in *Service-Oriented Computing: 17th International Conference, ICSOC 2019, Toulouse, France, October 28–31, 2019*. Springer, 2019, pp. 508–522.
[28] R. Mahmud, S. Pallewatta, M. Goudarzi, and R. Buyya, “ifogsim2: An extended ifogsim simulator for mobility, clustering, and microservice management in edge and fog computing environments,” *Journal of Systems and Software*, vol. 190, p. 111351, 2022.
[29] J. Deng, B. Li, J. Wang, and Y. Zhao, “Microservice pre-deployment based on mobility prediction and service composition in edge,” in *2021 IEEE International Conference on Web Services (ICWS)*. IEEE, 2021, pp. 569–578.
[30] A. Garbugli, A. Sabbioni, A. Corradi, and P. Bellavista, “Tempos: Qos management middleware for edge cloud computing faas in the internet of things,” *IEEE Access*, vol. 10, pp. 49 114–49 127, 2022.
[31] C. Puliafito, C. Cicconetti, M. Conti, E. Mingozzi, and A. Passarella, “Stateful function as a service at the edge,” *Computer*, vol. 55, no. 9, pp. 54–64, 2022.
[32] D. Barcelona-Pons, P. Sutra, M. Sánchez-Artigas, G. París, and P. García-López, “Stateful serverless computing with crucial,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 3, pp. 1–38, 2022.

Sehrish Amjad is a PhD candidate in the School of Science and Engineering, Lahore University of Management Sciences, Pakistan. Her research interests are in the areas of internet of things and edge computing.

Ahmed Akhtar is a PhD candidate in the School of Science and Engineering, Lahore University of Management Sciences, Pakistan. His research interests are in the areas of distributed systems and service-oriented computing.

Muhammad Ali is a PhD student in the School of Science and Engineering, Lahore University of Management Sciences, Pakistan. His research interests are in the areas of distributed computing and IoT systems.

Ayesha Afzal is an Assistant Professor at Air University, Multan, Pakistan. Her research interests are in the areas of distributed systems, business process management, and big data analytics.

Basit Shafiq is an Associate Professor of Computer Science in the School of Science and Engineering, Lahore University of Management Sciences, Pakistan. He has published over 75 research papers in international conferences and journals. His research interests are in the areas of distributed systems, security and privacy, semantic Web, and Web services.

Jaideep Vaidya is a Distinguished Professor of Computer Information Wystems at Rutgers University. He has published over 200 papers in international conferences and journals. His research interests are in privacy, security, data management, and data analytics. He is an IEEE and AAAS Fellow as well as an ACM Distinguished Scientist.

Shafay Shamail is a Professor of Computer Science in the School of Science and Engineering, Lahore University of Management Sciences, Pakistan. He has worked both in the software industry and academia. His research interests include software quality, cloud computing, and e-government architectures.

Omer Rana is the Dean of International for the Physical Sciences and Engineering College, Cardiff, UK. He obtained his BEng in Information Systems Engineering from Imperial College of Science, Technology Medicine (London University) and MSc in Microelectronics from the University of Southampton. He gained his PhD from Imperial College of Science, Technology Medicine (London University) in 1998. He is a member of IEEE, ACM and a Fellow of HEA UK.