Adaptive Edge-Cloud Environments for Rural AI

Osama Almurshed*, Panos Patros†, Victoria Huang†, Michael Mayo†, Melanie Ooi†,
Ryan Chard‡, Kyle Chard‡, Omer Rana*, Harshaan Nagra†, Matt Baughman‡, Ian Foster‡

*Cardiff University, Cardiff, UK; †University of Waikato, Waikato, NZ;
‡University of Chicago, Chicago, USA §Argonne National Laboratory, Lemont, USA

***Abstract*—Cloud computing provides on-demand access to computational resources while outsourcing infrastructure and service maintenance. Edge computing could extend cloud computing capability to areas with limited computing resources, such as rural areas, by utilizing low-cost hardware, such as single-board computers. Cloud data centre hosted machine learning algorithms may violate user privacy and data confidentiality requirements. Federated learning (FL) trains models without sending data to a central server and ensures data privacy. Using FL, multiple actors can collaborate on a single machine learning model without sharing data. However, rural network outages can happen at any time, and the quality of a wireless network varies depending on location, which can affect the performance of the Federated Learning application. Therefore there is a need to have a platform that maintains service quality independent of infrastructure status. We propose a self-adaptive system for rural FL, which employs the Greedy Nominator Heuristic (GNH) based optimisation to orchestrate application workflows across multiple resources that make up a rural computing environment. GNH provides distributed optimization for workflow placement. GNH utilises resource status to reduce failure risks and costs while still completing tasks on time. Our approach is validated using a simulated rural environment – composed of multiple decentralized controllers sharing the same infrastructure and running a shared FL application. Results show that GNH outperforms three algorithms for deployment of FL tasks: random placement, round-robin load balancer and simple greedy algorithm.**

***Index Terms*—Computing continuum, federated learning, serverless computing.**

I. Introduction

Cloud computing enables on-demand access to computational resources, delegating the maintenance of the infrastructure and services to an external provider. However, cloud service providers are increasingly used to build Internet of Things (IoT) applications which require low latency and privacy-preservation. These requirements are particularly significant in rural applications due to limitations in bandwidth available to support data communication and access to power to support large scale data centers. As the number of devices deployed in rural settings increases—e.g., in instrumented and precision farming and agricultural robotics—the limited Internet connectivity and performance can disrupt connectivity to the cloud. Edge computing provides a possible solution, extending the cloud to the low-cost hardware (e.g., single-board computers (SBCs)) deployed in rural settings.

Edge computing can effectively reduce latency between edge devices and cloud systems while also protecting data privacy as data and processing can stay at the edge. IoT applications are implemented as workflows comprised of *service functions* that may be executed across geo-distributed computing resources—from edge devices to the cloud. Coordinating such geographically distributed execution is complex as individual functions often fail or exceed deadlines. This results in delays, data loss during function migration, and higher costs.

Traditional machine learning algorithms combine data in a single location, generally a data center, which may violate user privacy and data confidentiality rules [1]. Also, the increasing cost of on-demand GPU encourages customers to deploy their own private GPU cluster [2]. Federated learning (FL) is a machine learning technique that trains models without transferring data to a central server, instead generating a model based on locally captured data. FL enables multiple actors to collaborate on the development of a single machine learning model without sharing data, which addresses critical issues such as data privacy and limited network infrastructure. We propose a self-adaptive system to support the use of FL in rural applications.

In our previous work, we described the Greedy Nominator Heuristic (GNH) [3], a distributed optimization algorithm for supporting function placement within a cloud-edge environment. GNH relies on redundancy in service functions and supports failure tracking on the connected computational resources. This is done to overcome the effect of resource failure and minimise the cost of supporting resilience by determining the number of replicas needed to overcome the effect of resource failures. A MapReduce-based mechanism is used to find and rank suitable locations (in parallel) for function replica placement. GNH has been validated with a real-world smart city application that monitors and manages surface water flooding data through the use of a centralized resource controller [4]. The results show that GNH is an efficient placement mechanism and supports proactive fault tolerance mechanisms. GNH has not, however, been evaluated in rural settings with unreliable network connectivity and mobile computing nodes as outlined in this work.

Internet outages in rural areas can occur at any time of day. Maintaining service quality regardless of infrastructure is critical. Furthermore, mobility has an impact on application performance because the quality of a wireless network varies depending on location. The requirements for FL workflows can also vary depending on the type of data being collected and the complexity of functions used in the workflow. A function

with a longer execution time, for example, may require more reliable computing resources to avoid reallocation and next task delay, such as in an FL context, where one workflow's output can be scheduled as an input to another. As a result, GNH must be evaluated in an environment composed of decentralized controllers sharing the same infrastructure and running a shared AI application.

The main contributions of this paper are as follows: (i) a scheduling strategy to support federated learning. The strategy makes use of resource properties to schedule and deploy applications; (ii) an adaptive system for rural AI applications using an intelligent scheduling system able to adapt to changing conditions; and (iii) evaluation of our approach via simulation, capturing intermittent connectivity and use of different system component behaviours. The rest of this paper is structured as follows: Section II includes a description of the the federated learning scenarios we consider. Section III describes the proposed adaptive system, and Section IV explains the simulation tools used in the validation process. Section V presents the experiment setup and includes an analysis of results. Before the conclusion section, related work is discussed in Section VI.

II. Weed Species Classification & Precision Agriculture

With increasing availability of data capture and analysis technologies (from specialist sensors to satellite-based data) to support agriculture, "precision agriculture" generally refers to the use of various information sources to improve the outcome of processes employed in farming and food production. This can include improving awareness of variation in soil and crop conditions, adapting fertilizer distribution and types to varying soil conditions across an agricultural field, automatic guidance of agricultural vehicles and implements, route guidance for autonomous machinery, product traceability and provenance (often associated with and the set of processes associated with "farm to fork"), on-farm research, and software for the overall management of agricultural production systems. Apart from field crop production, precision agriculture technologies have been applied successfully in viticulture and horticulture and in livestock production/ management, as well as pasture and turf management[1]. In this work we focus on one aspect of precision agriculture, namely identification and management of weeds – a time and labour intensive process.

Automatic weed control has the potential to significantly boost agricultural productivity. With efficient weed targeting, automatic weed control systems reduce labour costs while potentially lowering herbicide usage. Therefore, improving the effectiveness of weed control can be extremely beneficial. Successful agricultural robotics development should reduce losses while increasing output.

Autonomous weed spraying requires machine learning methods to process images (often in real time) and determine which areas contain weeds. To build these models we used the

[1]https://www.science.org/doi/full/10.1126/science.1183899

	Raspberry Pi	*Jetson Nano*
Aggregate Models	22.33s	139.81s
Image Pre-processing	0.33s	0.48s
Evaluate Model	37.16s	170.60s
Compare Accuracy	0.10s	0.12s
Model Tuning	178.16s	212.8s

TABLE I
Benchmarking service functions on SBC platforms

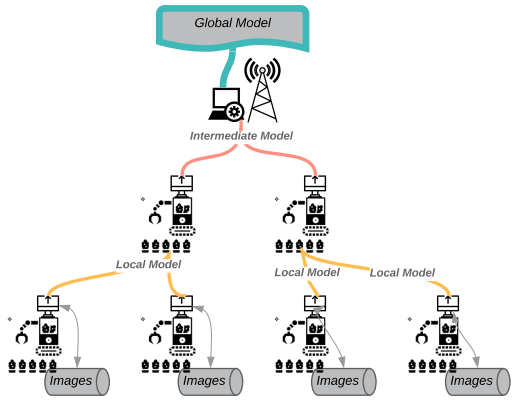

Fig. 1. Aggregating learned models across robots using Federated Learning

DeepWeeds [5] dataset: an image-based weed species recognition dataset. We used this dataset to train a weed classification model using an off-the-shelf deep learning model, ResNet. We then modified the training algorithm to use an online FL approach, which continuously adapts the model as new images are added to the system.

The federated learning approach uses hierarchical model aggregation, similar to FEDn [6], where a robot uses locally captured data to train a model. Models from individual robots can then be combined to improve accuracy – i.e. all of the models are aggregated into one single, global model. Individual models with low accuracy can be discarded during the creation of a global or intermediate model. Figure 1 shows our approach when applying hierarchical federated learning to a collection of robots – operating as edge devices in this application.

A. DeepWeeds Dataset

DeepWeeds [5] is an open dataset containing images of various weed species from the Australian Rangelands. The dataset includes 17,509 images capturing eight different weed species across eight different locations. ResNet-50 [7], which is accessible in TensorFlow/ Keras with pre-trained weights. The weights of the original models were trained using the ImageNet dataset [8].

B. Applications Workflows

Machine learning pipeline activities, including data preparation, training/tuning, validation and model aggregation, are necessary in FL applications. The model architecture is kept locally on each edge node in our application scenario. The

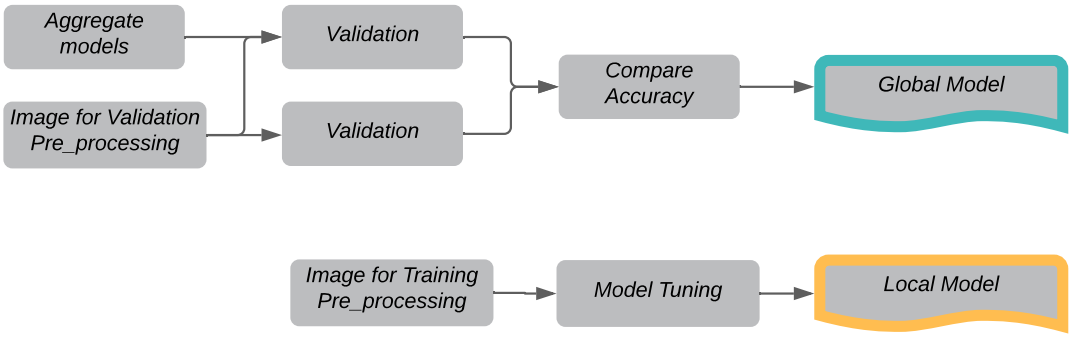

Fig. 2.

model weights are distributed as part of the task placement procedure. The robots utilize the same workflow function composition to create an intermediate model. The sole distinction between the Global and Intermediate Models is that the Global Model is constructed by aggregating the weights of the intermediate models, while the Intermediate Model is constructed by aggregating the weights of the local models. Figure 2 shows the workflow used in the hierarchical federated learning process.

We modified the training procedure in Section II-A to make it suitable for incremental fine-tuning, which results in the generation of the local model. The key difference between incremental learning and classical machine learning is that it does not require a large training set prior to the learning process, but rather training instances can emerge and be generated over time [9]. This is appropriate for IoT and service-based activities in edge computing, where data is analyzed as it is generated, and requires the incremental training phase to be completed within a time constraint/ deadline. Each function in the workflow can be described as follows:

1) **Image Pre-processing:** this step involves processing an image and preparing it for training or validation purposes. This method is used to transform the data to the 224x224 image size format required by the deep learning model.
2) **Model Tuning:** this step involves fine-tuning the learning model (a neural network) and includes the use of weights from previous models as initialization, leading to the development of a new model trained on the weed image data set.
3) **Model Aggregation:** involves calculating the average of the weights in all neural network models.
4) **Validation:** a trained model is tested with a new set of data.
5) **Comparing Accuracy:** this function takes as input the results of the validation outcomes, comparing the accuracy of the old and new models. The most accurate model will be saved as the global model.

C. Robot Mobility

We consider multiple autonomous robots that must complete tasks across an agricultural field, with an intention to optimise field coverage by determining a movement trajectory. The movements of these robots are governed by a random walk approach aiming to cover the field [10] in minimal time, whilst also completing the required tasks to a high accuracy. The use of a truncated random walk approach, which generates step lengths that follow a predetermined distribution and fall within a specific range, improves the efficiency of the search process [10]. The robot's movement affects the quality of data communication that can made to field-side units or to a cloud platform. The quality of communication declines as the robot advances farther from the computing resources (i.e. those resources that are not directly hosted on the robot). This emulates network latency in a mobile edge device.

D. Application Requirements

Rural AI applications require a platform that is both reliable and able to adapt to changes in the infrastructure. The following describes a set of Rural AI application requirements:

Operation during system failure: The application must continue execution in the event of a partial network & computational resource outage. Fault tolerance ensures that the system continues to function by avoiding the assignment of tasks to a failed component. The task scheduler should minimise use of resources that are more prone to failure.

Adapt to rural areas conditions: Rural locations might suffer from internet outages at any time of the day, requiring a mechanism to maintain a connection to the service that is hosted on an external system. It is also important to maintain service quality within a pre-defined threshold despite the state of the infrastructure.

Mobile edge device: Mobility affects application execution performance and the type of data that can be exchanged over a wireless network. The location of the robot impacts the types and size of data, and latency that can be supported.

Meeting FL workflow requirements: Different FL workflows may differ in their computational requirements. For example, a workflow function with a longer comparative

execution time may necessitate the use of more dependable computing resources to avoid resource reallocation and the resulting delay in task completion. The result of one workflow can be scheduled to be the input to another process in FL, for example, if the workflows are interdependent.

Protecting data privacy: The edge network should have the capacity to protect data privacy. Since a single application failure might have an adverse effect on the speed at which others execute, it is necessary when switching from one services instance to another that data privacy requirements are adhered to.

III. Adaptive Edge-Cloud Infrastructure

At the core of adaptive systems is a closed-loop process. Adaptation loops are composed of several processes, including observation and action. Adaptation management consists of a series of processes for implementing and collecting observations, as well as analyzing and monitoring those observations, planning modifications, and deploying/ actuating change. The four phases of autonomic control are as follows: (1) data collection, (2) analysis, (3) decision-making, and (4) action.

Data collection entails monitoring the deployment process and recording the state of system components, such as the capacity of the processing node over time, in order to detect and decide on the appropriate response (learning or immediate reaction) to changes in the environment.

Analysis is the process of determining how data will be used during the decision-making process. For instance, calculating the risk of failure of a particular component and using this to support scheduling and function placement. This phase may also include updating and encoding decision variables associated with the placement decision.

Decision-making typically includes a strategy or algorithm that generated the decision based on the data collected and analyzed. Between the time it takes to plan actions (decision-making) and the time it takes to act—e.g. deploy a predefined service—the environment can change. As a result, the decision-making component should respond to changes in the environment (re-optimize or perform dynamic optimization) and avoid maintaining the initial decision (optimization using static data).

Action executes the decision as part of the active loop. This may involve managing a series of events in order to execute the action determined during the decision-making phase. For instance, utilizing data streaming and computational tools. We make use of Parsl and FuncX to deploy application requests.

A. Parsl

Parsl [11] is a parallel scripting library for Python, designed for high-performance and distributed environments. Parsl relies on its DataFlow Kernel (DFK) to manage data exchange between service functions in a workflow and can be used to orchestrate individual Python functions across multiple geographically distributed locations. Parsl implements a modular runtime model using which different *executors* can be configured to execute tasks on computational resources. We use executors to represent different locations. Service functions are implemented as Parsl apps. The DFK is informed of the app's desired execution location at runtime by specifying the appropriate Parsl executor in the app `decorator`. Invoking a Parsl app returns a `future` reference which can then be polled for results once results are available. Futures allows non-blocking, asynchronous execution of the function while providing a reference that can be passed to other functions, and establishing a dependency between functions. Further, this model allows other processing to continue while the task execution is still in progress. Building on Parsl, funcX [12] is a federated function-as-a-service platform enabling high-performance, distributed serverless applications.

B. Placement Strategy

Rural applications deployed on edge computing resources have limitations that are not present in traditional systems, many of which have a negative impact on Quality of Service (QoS). For example, nodes may be unreliable, causing functions to fail to meet deadlines or simply fail outright; edge infrastructure is more prone to additions, removals or relocations (i.e., mobile device); and exploring a large search space for possible deployment locations is impractical. Placement strategy decides where and when to place applications in a manner that fulfils application deployment objectives and requirements. This decision-making may need to address conflicting metrics during scheduling, e.g. when service replication (to enhance reliability and availability) conflicts with a need to minimize deployment costs and avoid overwhelming backend systems. This requires placement strategies that are aware of backend capacities and capabilities and that can be guided by application-defined priorities. Application placement takes into account privacy and security constraints. This keeps sensitive data securely contained at the edge. Rapid and scalable decision-making will provide fast response times and low end-to-end latencies (e.g., for selective harvesting or precision-spraying at scale).

We use the Greedy Nominator Heuristic (GNH) [3] to solve the placement problem. GNH relies on redundant virtual function deployment and failure tracking. GNH replicates each function at numerous locations based on predicted completion time, risk of failure and cost of deployment. We employ a MapReduce-based approach in which Mappers search for potential placement locations concurrently and the Reducer then ranks these locations, then selects the best placement.

The rural AI requirements described in Section II-D lead to the following three objectives addressed by GHN: minimising overall workflow makespan, reducing risk of non-completion (i.e. supporting workflow resilience) and minimising cost by reducing the number of replicas needed to achieve resilience. We define each objective as follows.

- **Makespan** is the total completion time of the workflow execution across distributed resources; this is the primary goal of the scheduler.

- **Risk** is measured by the likelihood of failure to complete within the deadline. This is regarded as a proactive fault-tolerance mechanism designed to reduce service outages.
- **Cost** is determined by the number of resources used to deploy the workflow. As rural areas have access to limited computational resources and network bandwidth, reducing resource use also minimises network traffic congestion on the infrastructure. Redundant deployment lowers the risk of failure but increases the cost. As a result, the number of redundant copies of functions must be balanced with the capacity of the available resources and the cost of deployment on these resources.

Greedy Nominator Heuristic (GNH): The proposed placement strategy aims to optimise on three key metrics: (i) reducing end-to-end latency, (ii) ensure continued operation during failure,(iii) minimising the number of worker nodes required to execute the distributed application. The platform uses GNH to determine the placement location of functions. GNH uses function replication and failure monitoring to tolerate partial failures during run-time; this is done in two phases nomination and announcement.

Nomination phase: the decision variables are divided into a number of concurrent processes, with each process acting as a nominator. A process is an instance of a program that generates a partial decision, which results in the nomination of a collection of k-resources to deploy a service function. All nominators are running in parallel.

Announcement phase: out of the nominees only k are announced as winners and execute the function. k-winners have the lowest latency, the lowest failure rate and host a function that is part of the application workflow. The number k is based on identifying $MaxReplicas$ where ($MaxReplicas = 1 + \lceil(1 - \frac{i}{n+1})m\rceil$), where n represents the number of time slots (sequences) in the workflow, where 'i' is a specific sequence or slot. The sequences are time windows where independent functions can run in parallel. Redundancy is achieved through the deployment of a number of replicated functions, as illustrated in Figure 3. The initialized functions (i.e., those that are executed early in the process) have the most replicas, $MaxReplicas$. As we move through an application composed of n functions, this value drops, as presented in Figure 3. A key assumption is that in a workflow, functions that execute first are more significant, as their failure will lead to failure of the complete workflow (due to data dependencies captured in the workflow graph). Consequently, functions that occur early in the workflow have a greater number of replicas.

Based on [3], the optimisation problem can be formulated as minimisation of *C, R* and *O* (based on symbols in table II).

$$C = \sum_{j \in F} \sum_{k \in L} T_{j,k} \cdot y_{j,k} \cdot x_{j,k} \tag{1}$$

$$R = \sum_{j \in F} \sum_{k \in L} Risk_{j,k} \cdot x_{j,k} \tag{2}$$

$$O = \sum_{k \in L} o_k \tag{3}$$

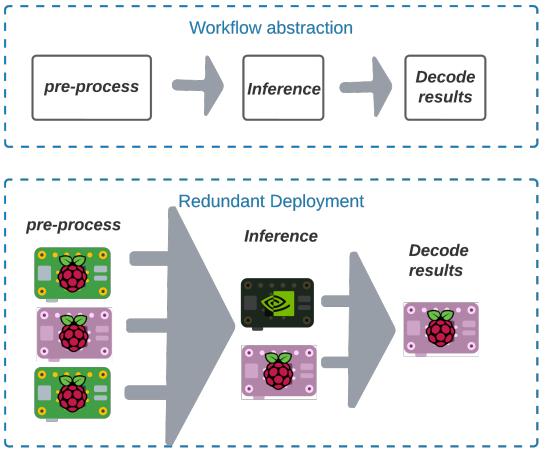

Fig. 3. The location of the function in the workflow influences how many replicas are creating – resulting in a *funnel-shaped* workflow structure using the $MaxReplicas$ formula. Functions at the earlier stages of a workflow have a greater number of replicas.

Symbol	Description
F	The set of all functions in the workflow
D	The set of pairs representing dependencies between functions
A	The graph of the workflow, which is $A = (F, D)$
L	The set of all locations that execute functions
i	The sequence of function f_j^i in the workflow
j	The type of the function f_j^i
k	The index of Location l_k
$T_{j,k}$	Processing time for f_j in location l_k
$x_{j,k}$	Binary variable: function f_j^i is executed on location l_k
$y_{j,k}$	Binary variable: $T_{j,k}$ contributes to longest path
o_k	Binary variable: location l_k is used by the application
E	The set of all placed paths of A
$Risk_{j,k}$	Risk of executing f_j in location l_k
$MaxReplicas_{i,j}$	The maximum replicas for f_j^i
m	Constant to adjust maximum possible replica
r_k	The number of failures per allocations
M	The longest path in the application, $M = \{T_{1,1}, T_{2,1}, \ldots T_{i,j}\}, M \in E$

TABLE II
NOTATION USED IN GNH FORMULATION

Where objective function C (formula 1) is the total completion time, and objective function R (formula 2) is the total risk of application A completing successfully. The number of locations used to execute A, including redundancy, is represented by objective function O (formula 3).

C. System Architecture

Robots host computational infrastructure to execute functions, which can be used to analyze, optimize, configure and maintain specific application requests (e.g. identify weeds, identify soil conditions, pick fruit, etc). Application software can range from image processing functions that execute on images captured by a co-hosted camera, to communication functions that are able to interact with other robots and field side units. A robot has limited on-board processing capability and needs to interact with nearby infrastructure or a cloud

data centre to offload complex tasks that cannot be processed locally.

A controller component houses the utility software, which can be divided into two main categories: Online Optimizer and Resource Monitoring, as illustrated in Figure 4. An application's workflow is specified as a Directed Acyclic Graph (DAG). In a DAG, each node represents a single service function that the controller is capable of deploying in the scheduling process. Identifying where a functions is executed is determined by a controller and the application scheduler.

Field-side units are generally single-board computers to which mobile robots offload some of their service functions and data. Hence, the computational capacity hosted on a robot acts as a "worker node" in the edge computing layer, combined with field-side units that serve as an external platform for storing and accelerating additional resources for computationally intensive tasks. The resources provided by field-side units are used to compute service functions and the robot handles task allocation. It selects a suitable unit to process the next function in the DAG after receiving all data from the previous field-side units.

A function execution unit (referred to as the controller above) is hosted on a robot and makes use of Parsl [11] to access the resources of the field-side units throughout the edge infrastructure. It is based on a master-worker architecture for communication, where the controller hosted on a robot is the master.

We consider five main steps in the application deployment process: receive application request, decide where to place it, run the application, monitor its state and update decision variables based on the status of the deployment. When the controller receives the request, it configures the decision-making parameters and the deployment components in advance of any use. The execution then proceeds to iterate through all of the workflow's service functions. The decision-making process will determine placement. Deployment components then execute the decision via Parsl. The monitoring component will track the execution of the function and report on the outcomes of the execution to the controller. Additionally, the monitoring components is able to record the robot's communication with the field side units. Using data from the monitoring process, the decision-making components can determine the placement of the next function to be executed.

IV. Simulation

In the simulation we consider intermittent connectivity between a robot and field-side units, and varying computational capacity on the robot. Both of these parameters influence the offloading decision made by the robot. Our simulation is able to capture the dynamic nature of these parameters.

A. Failure Model

This section describes the failure model that was used in this study, indicating when each processing node (i.e., field-side unit) experiences a failure or a recovery event. We use field-side unit failure to model and validate the performance of the scheduler making use of federated learning; however, we do not include software defects caused by a faulty service function implementation in this model.

TABLE III
Simulation parameters

Variable	Number/Ranges
Field-side units (RPi)	100
Robots (RPi)	10
MTTF	(250 - 500 s)
MTTR	(20 - 100 s)
Random walk steps	(1 - 10 steps)

If a field-side unit does not respond to requests submitted by a robot within a time threshold, this is considered as task failure. This lack of response adds to the time it takes to restore the field-side unit, which is specified as the recovery time. The Mean Time To Failure (MTTF) is the average time that the system is up and running, and the Mean Time To Recovery (MTTR) is the time until the field-side unit becomes available again to process tasks (MTTR). The mean time between two failures is calculated as the sum of MTTF and MTTR (MTBF).

Every field-side unit has a repeating timer known as the MTBF-clock, which indicates when requests arrive at a field-side unit. The arrival time of the function request in the field-side unit, according to MTBF-clock, will determine whether the allocated field-side unit has failed or succeeded. Deploying a function successfully means that the function began execution and was completed within the field-side unit's MTTF period. Failure of function deployment, on the other hand, indicates that the function execution overlapped with the MTTR period of the field-side unit's MTBF-clock. Figure 5 shows service functions that were submitted over three different MTBF-clock periods [3].

B. Random Walk and Link Delay

The robot's movement is modelled as a stochastic process that advances through a random walk. The simulation consists of a small number of random steps, with each iteration of the simulation projecting the robot's coordinates onto a 2D plane/ grid. The quality of communication degrades as the robot advances farther from the field-side unit. This emulates varying network latency in a mobile edge device. A pseudo-random number generator is used to influence the choice of step made by a robot on a 2D grid at every simulation step.

Robots and field-side units in the 2D plane are randomly initialized when the simulation begins. locations in 2D plane represented as *(x, y)* coordinates. The robots move, whereas the field-side units remain stationary. A random walk begins with the selection of a random number of steps to be taken. The number of steps is [1–10]. After deciding on the steps, each step can be north, south, east, or west. For example, if the robot moves to the north, it will add (0, 1) to its location. After the walk, the simulation calculates the distance between the robots and each field-side unit. The distance value is used to determine the quality of a connection as follows: quality = 1 - distance/unreachable. The point at which the connection delay

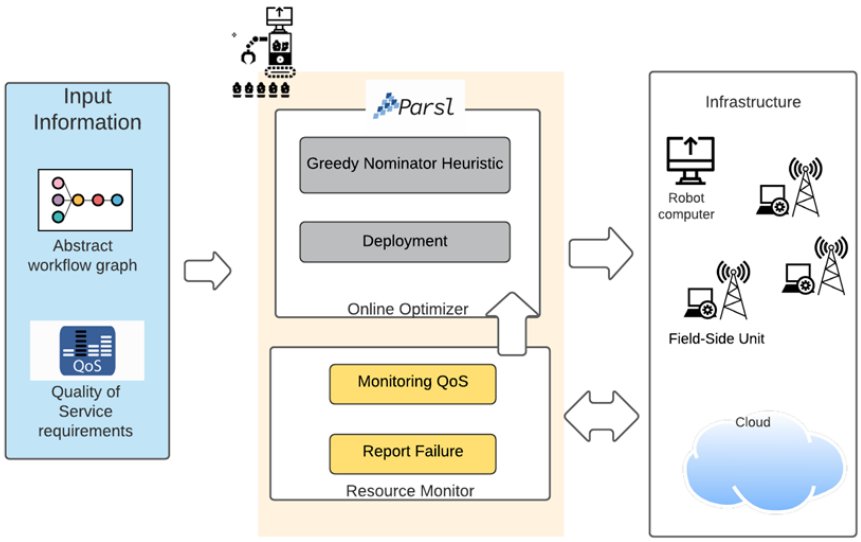

Fig. 4. The proposed adaptive controller architecture

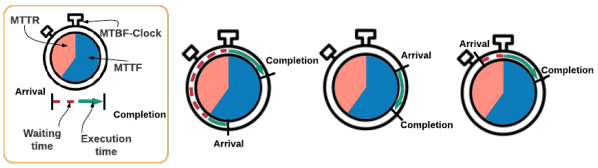

Fig. 5. The execution of a service function at a field-side unit, with completion times taking account of Mean Time To Failure (MTTF) and Mean Time to Recovery (MTTR). Arrival and completion times influence successful completion of a task.

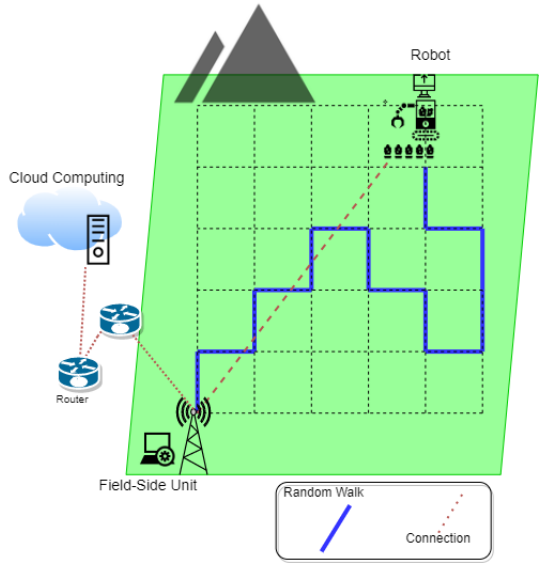

Fig. 6. A robot is performing a random walk, which affects the quality of the connection to a field-side unit.

exceeds a pre-defined threshold (indicating that connection has been lost) is considered *unreachable* (we set this to 3 times the average delay observed between field-side units at opposite ends of the field). The time to send data over the network in an ideal situation is *link time = data size/link speed*. In the simulation, link delay is calculated using *link delay = link time/quality*. When the quality is set to 1.0, the wireless link uses its entire capacity, and operating at maximum available speed. Increasing the value results in a shorter latency.

V. Evaluation

In the following sections we describe the experimental setup used to validate our work and our results. We evaluated the proposed adaptive system in a simulated environment, where the field-side units have failure models that affect system performance. Variable network connectivity and disruption is simulated by dynamically varying the quality of the network link connecting the robot to the field-side units. The performance measurements include execution time, failure risk/ resilience, and cost of deployment.

A. Experimental Setup

We evaluate GHN using a Raspberry Pi model 4B. The simulation generates application requests dynamically, 3000 of which train a local model and 300 of which aggregate models to generate a global model. It also varies resource availability and failure profile using a clock mechanism divided into two periods: (i) a failure event (MTTR) and (ii) normal resource operation (MTTF). We run several scenarios and compare GNH to random placement (RP), use of a simple Greedy algorithm, and Round-Robin load balancing (RR). The simulation produces records containing application request arrival times and workflow compositions. The deployment of workflows is constrained by a deadline, which is the expected completion time defined by the user. During evaluation, the

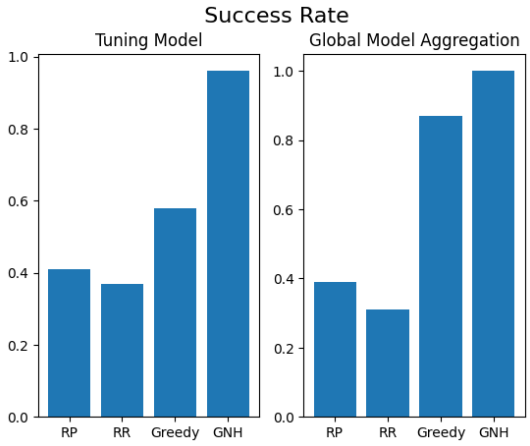

Fig. 7. Success rate for each algorithm – Tuning Model refers to the locally generated machine learning model

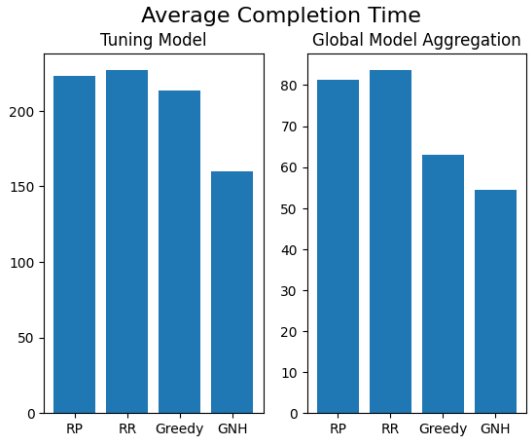

Fig. 8. The average completion time of tasks under each placement strategy

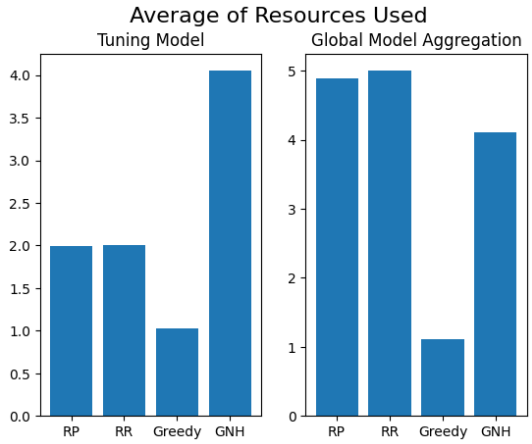

Fig. 9. Average cost – based on the number of locations utilized

expected and actual completion times are compared to the deployed states. If the deadline has passed, the workflow deployment is considered a failure. Each field-side unit has a failure model parameter that is chosen at random from a pre-defined range. On a single day, application requests are distributed uniformly. This simulates data streams generated at infrastructure controllers.

The experiment runs as follows: first, components for monitoring, deployment, decision-making and request generation are set up. This is followed by iterating through all of the requests, and selecting a controller at random to handle each function. As each controller may be located at a different location on the agricultural field, the response time will be affected based on the distance between the robot and the field-side unit. The controller needs to determine which functions are executed on the robot, and which need to be offloaded to the field-side unit. Following the deployment of each function, a random walk with steps ranging from 1 to 10 will be performed. Failure rate records are updated after each function placement to assist decision-making components improve the accuracy of the outcome. Finally, the workflow deployment states are saved by the monitoring component.

B. Results

Figure 7 shows the success rate of each algorithm. Most of the requests supported using GNH were completed ahead of schedule, with the simple Greedy method achieves the second best performance with a 58% success rate. It is worth noting that both GNH and Greedy are heuristic-guided search algorithms. In terms of success rates, RR has the worst performance followed by RP. Over half of all RR and RP requests were completed after the deadline. Randomization has a good chance of outperforming strict sequential execution in terms of overall performance. The heuristic function determines how long and how much each deployment will take. Greedy, despite being informed, has a much higher failure rate than GNH. GNH outperforms the simple Greedy approach in terms of performance because it selects multiple field units in case one fails. As a result, all global Model Aggregation requests were completed in less time than expected, while only 96% of Tuning Models requests took longer than expected. GNH completes more work in less time because redundant functions are used.

The failure of a function increases the completion time, as shown in the Figure 8. GNH is the best on average; it is 53seconds faster than Greedy in Tuning Model execution and 8.6 seconds faster in global model aggregation execution. When compared to RR and RP, GNH can save up to 67 seconds for model tuning and 29 seconds for model aggregation.

GNH deploys 4.06 field-side units on average, as seen in Figure 9. The only advantage of the other approaches over GNH is that they use fewer field-side units per deployment, in case the workflow size is less than GNH's MaxReplicas, Tuning Model is a case in point, because the average number of resources used are twice the size of the workflow. The workflow of global model aggregation is made up of five functions. As the random placement can still sometimes use the same resource for two functions, RP uses 4.89 resources on average. Whereas the Greedy algorithm uses 1.11 field-side

units on average, this means that if the resource is considered reliable in the first function deployment, it will be used again.

VI. Related Work

Multi-robot systems: Operation control strategy for a multi-robot system is either centralized or decentralized [13]. Conesa-Munoz et al. [14] described a platform with centralized strategy for managing the multi-robot system workflow from a base station. The robots in the system can be either aerial or ground vehicles. By fully automating the integration of both types of platforms, this combination enables autonomous tasks in large outdoor areas. The entire workflow was automated, with no human intervention required except as required by Spanish law for safety reasons. In the case of a decentralized cooperative control of heterogeneous robot groups. Tanner et al. [15] coordinate the interaction of two heterogeneous groups of mobile agents in discrete time. A group of ground vehicles and a group of unmanned aerial vehicles interact using time-invariant, nearest-neighbor rules. A Lyapunov analysis is also used to ensure that ground vehicles track the centroid of the ground group.

Drones/aerial vehicles collect environmental data as part of their plan, while ground units are tasked with carrying out the operation in farming areas. All of the vehicles are controlled by a Mission Manager, which is linked to a Base Station computer. In the Operation Manager, each aerial and ground unit is assigned a planner. The Operation Manager sends data to a planner, which then divides the fields into tiny grids based on orientation and image resolution. A Harmony Search Algorithm is used to generate the best plan for flying units to cover the entire region [16]. Ground planners distribute tasks among ground vehicles using a meta-heuristic optimization strategy. When creating this planning tool, both location and power consumption were considered. The ground planner determines the best path for each ground vehicle to take to cover an operating area.

Simulation: In simulation-based experiments, the random walk method of swarm robots is used to quantify the area search efficiency [10]. The search efficiency is increased by employing a truncated random walk method that generates step lengths that conform to a specified distribution and fall within a specified range.

Plant datasets: Computer vision technologies have attracted significant interest in precision agriculture in recent years. The scarcity of public image datasets remains a crucial bottleneck for fast prototyping and evaluation of computer vision and machine learning algorithms [17]. The Open Plant Phenotype Database [18] is a publicly accessible dataset for plant detection and classification. The dataset includes 7,590 RGB images of 47 different plant species. Each species is cultivated under three different growth conditions to provide a high degree of visual diversity. Furthermore, the individual plants have been tracked over time. The dataset is accompanied by two experiments for detecting plant instances and classifying plant species. DeepWeeds [5], Australia's first large public multiclass image dataset of weed species, enables the development of robust classification methods for robotic weed control. DeepWeeds includes 17,509 tagged images of eight nationally significant weed species found in northern Australia. The classification performance on the dataset was evaluated using the Inception-v3 and ResNet-50 deep learning models. These models had classification accuracy of 95.1 percent and 95.7 percent, respectively. ResNet-50 performance in real-time, with an average inference time of 53.4 ms per image.

Federated Learning FEDn [6] is a hierarchical FL framework designed to address the scalability and resilience issues commonly found in FL systems. With the framework's emphasis on horizontally scalable distributed deployments, FEDn makes local development easier.

VII. Conclusion

Edge computing, which employs low-cost hardware such as single-board computers (SBCs) in proximity to the data capture source, has the potential to extend cloud computing to regions with limited computing infrastructure, such as rural areas. We describe a software platform using Parsl and funcX to execute machine learning applications to support precision agriculture. The approach is demonstrated using the DeepWeed data set, but can be generalised to a number of other similar agritech. applications. We consider a robot operating on an agricultural field having variable data transfer latency to one or more field-side unit(s) and a cloud data centre. Our approach considers how offloading decisions can be made by a controller on the robot, taking account of variable failure rate of field-side units.

Use of cloud-hosted centralized machine learning methods can be expensive and risk user privacy and data confidentiality. Using the federated learning (FL) approach, multiple SBCs (each hosted on a robot) can collaborate on the same machine learning model without transmitting data to the cloud, ensuring data privacy and security. Nonetheless, the quality of a wireless network varies greatly from location to location, especially in rural areas, which may have an impact on the performance of the FL application. The Greedy Nominator Heuristic (GNH) is used to manage deployment of a set of functions (as a workflow) on our proposed self-adaptive rural FL platform.

Using metrics makespan, risk and cost, we have shown that GNH outperforms three competing algorithms for federated learning deployment. During local model generation, we found that GNH was completed ahead of schedule, outperforming the simple greedy approach (second best) by 65.52%. GNH, on the other hand, outperformed a greedy search by 14.94% for generating global model. Furthermore, the results show that GNH is the best approach out of the three in terms of balancing the number of field-side units used and the number of redundant deployments of service functions, particularly as we increase the number if functions in a workflow. Finally, because GNH has a low number of deployment failures, the workflow placement decision made by GNH has the shortest execution time of all the decision-making algorithms considered.

REFERENCES

[1] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, "Federated learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 13, no. 3, pp. 1–207, 2019.

[2] J. Emeras, S. Varrette, V. Plugaru, and P. Bouvry, "Amazon elastic compute cloud (ec2) versus in-house hpc platform: A cost analysis," *IEEE Transactions on Cloud Computing*, vol. 7, no. 2, pp. 456–468, 2016.

[3] O. Almurshed, O. Rana, and K. Chard, "Greedy nominator heuristic: Virtual function placement on fog resources," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 6, p. e6765, 2022.

[4] 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 multi cloud edge environment," *IEEE Internet Computing*, 2021.

[5] A. Olsen, D. A. Konovalov, B. Philippa, P. Ridd, J. C. Wood, J. Johns, W. Banks, B. Girgenti, O. Kenny, J. Whinney *et al.*, "Deepweeds: A multiclass weed species image dataset for deep learning," *Scientific reports*, vol. 9, no. 1, pp. 1–12, 2019.

[6] M. Ekmefjord, A. Ait-Mlouk, S. Alawadi, M. Åkesson, D. Stoyanova, O. Spjuth, S. Toor, and A. Hellander, "Scalable federated machine learning with fedn," *arXiv preprint arXiv:2103.00148*, 2021.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[9] R. Ade and P. Deshmukh, "Methods for incremental learning: a survey," *International Journal of Data Mining & Knowledge Management Process*, vol. 3, no. 4, p. 119, 2013.

[10] B. Pang, Y. Song, C. Zhang, and R. Yang, "Effect of random walk methods on searching efficiency in swarm robots for area exploration," *Applied Intelligence*, vol. 51, no. 7, pp. 5189–5199, 2021.

[11] Y. N. Babuji, K. Chard, I. T. Foster, D. S. Katz, M. Wilde, A. Woodard, and J. M. Wozniak, "Parsl: Scalable parallel scripting in python." in *IWSG*, 2018.

[12] R. Chard, Y. Babuji, Z. Li, T. Skluzacek, A. Woodard, B. Blaiszik, I. Foster, and K. Chard, "Funcx: A federated function serving fabric for science," in *Proceedings of the 29th International symposium on high-performance parallel and distributed computing*, 2020, pp. 65–76.

[13] G. Pantelimon, K. Tepe, R. Carriveau, and S. Ahmed, "Survey of multi-agent communication strategies for information exchange and mission control of drone deployments," *Journal of Intelligent & Robotic Systems*, vol. 95, no. 3, pp. 779–788, 2019.

[14] J. Conesa-Muñoz, J. Valente, J. Del Cerro, A. Barrientos, and A. Ribeiro, "A multi-robot sense-act approach to lead to a proper acting in environmental incidents," *Sensors*, vol. 16, no. 8, p. 1269, 2016.

[15] H. G. Tanner and D. K. Christodoulakis, "Decentralized cooperative control of heterogeneous vehicle groups," *Robotics and autonomous systems*, vol. 55, no. 11, pp. 811–823, 2007.

[16] A. Nabaei, M. Hamian, M. R. Parsaei, R. Safdari, T. Samad-Soltani, H. Zarrabi, and A. Ghassemi, "Topologies and performance of intelligent algorithms: a comprehensive review," *Artificial Intelligence Review*, vol. 49, no. 1, pp. 79–103, 2018.

[17] Y. Lu and S. Young, "A survey of public datasets for computer vision tasks in precision agriculture," *Computers and Electronics in Agriculture*, vol. 178, p. 105760, 2020.

[18] S. Leminen Madsen, S. K. Mathiassen, M. Dyrmann, M. S. Laursen, L.-C. Paz, and R. N. Jørgensen, "Open plant phenotype database of common weeds in denmark," *Remote Sensing*, vol. 12, no. 8, p. 1246, 2020.