Resource Inference for Task Migration in Challenged Edge Networks with RITMO

Resource Inference for Task Migration in Challenged Edge Networks with RITMO

Alessio Sacco† Flavio Esposito⋆ Guido Marchetto†
†Politecnico di Torino, Italy ⋆Saint Louis University, USA

***Abstract*—Edge computing, combined with the proliferation of IoT devices, is generating new business model opportunities and applications. Among those applications, Unmanned Aerial Vehicles (UAVs) have been deployed in several scenarios, from surveillance and monitoring to disaster response, to precision agriculture. To support such applications, however, edge network managers and application programmers need to overcome a few challenges, e.g., unstable network conditions, high loss rate, and node failures. Existing solutions designed to mitigate such inefficiencies by predicting future network conditions are often computationally intensive and hence less portable on constrained devices. In this paper, we propose RITMO, a distributed and adaptive task planning algorithm that aims at solving these challenges while running on a network of UAV devices. We model our system as a network of queues, and we exploit a simple yet effective ARIMA regressor, to dynamically predict the length of future UAV task queues. Such prediction is then used to proactively migrate the tasks in case of a failure or unbalanced loads. Our simulation results demonstrate how RITMO helps to reduce the overall latency perceived by the application and anticipates the node overloading by avoiding agents that are likely to exhaust their computational resources.**

***Index Terms*—task offloading, regression prediction**

I. Introduction

Distributed applications running on Internet of Things (IoT) devices that require to perform a mission independently are opening many applications, sometimes improving lives, sometimes even saving them. Typical examples are Unmanned Aerial Vehicles (UAV) networks, e.g., drones, equipped with cameras, sensors, or civilian tablets and smartphones [1], [2]. Such systems have been employed, for instance, in disaster response and environmental monitoring [3], [4], or to provide connectivity to ground stations [5]. Autonomous and semi-autonomous drones will continue to help humans accomplish many tasks, spanning from industrial inspection to survey operations, from rescue management systems to military or first responder support. A network of drones can be used, for example, to collect a massive quantity of data that needs to be offloaded at the network edge for heavy audio/video processing, where resources to execute Machine Learning (ML) algorithms are readily available.

While drone-based and IoT-based applications continue to grow exponentially, the challenge of keeping an acceptable quality of service with strict delay constraints for these network increases as well, especially in challenged scenarios [6]–[8]. This problem depends on the quality of connectivity among such devices and on the dynamic nature of the tasks that the drones are required to accomplish.

To provide a persistent and adaptive service, centralized [9], [10], and distributed [11], [12] solutions that allow an edge network of IoT, drones, or robots in general, already exist. These solutions share the use case of multiple IoT agents accomplishing a mission, but address the problem in different forms. Some of them focus on the resilient mission planning problem [12], others on agents' health-aware solutions [11], others yet [10] on the problem of enabling agents to autonomously tackle complex, large-scale missions, in the presence of actuator failures. However, none of them can anticipate demand fluctuations by looking at the past and learn from prior errors, such as orchestrating the task assignment through a resource usage prediction.

In this paper, we propose RITMO (Resource Inference for Task MigratiOn), an algorithm that proactively redistributes job loads among multiple processes running within distributed nodes. To efficiently share the load and minimize the task completion time, each agent predicts the future queue length and accordingly migrates jobs (i.e., drone tasks) to less loaded agents. Our system uses a predictor that determines the agent's future load based on time-series forecasting. In particular, we use the Autoregressive Integrated Moving Average (ARIMA) algorithm [13]. Unlike other machine learning-based methods, the features exploited by ARIMA are restricted to just one value in time-series forecasting. We have experimented that this property well fits constrained environments such as the drone swarms offloading tasks to the edge cloud, since this class of algorithms does not require a large amount of memory. Such information can then be used to adapt the agent's load to a policy profile that can minimize the task completion time and satisfy the strict time requirements of the task offloading scenario. Our results show how RITMO provides better performance with respect to the benchmark algorithms even for a large number of nodes and when the high rate of failures generates significantly changing conditions that are challenging to manage.

The rest of the paper is structured as follows: Section II presents the most related solutions to RITMO, Section III introduces the RITMO's model and formalizes the problem described in the paper. The algorithm utilized to solve such a problem is then described in Section IV, while Section V outlines the main components of our solution. Then, Section VI shows the performance of RITMO and the advantages over

978-1-7281-9486-8/20/$31.00 ©2020 IEEE

similar solutions. Finally, Section VII concludes our paper.

II. Related Work

The problem of providing a persistent and adaptive service resilient to failure is crucial for any IoT network in general, and robotic or drone networks in particular; so it is not surprising that there are several proposed solutions to tackle this problem. A proper architecture for edge offloading is crucial for critical applications, e.g., real-time video conferencing with the incident commander to recognize faces of disaster victims [14], or the detection of children in an attempt to reunite them with their families [15], whereas virtual beacons can be mainly used to track their location.

Recently, decentralized approaches have been proposed to improve the adaptability and the persistence of distributed IoT systems [9], [10], [16]–[19]. For example, [20] addresses the problem of task allocation and scheduling for a heterogeneous team of human operators and robotic agents. Unmanned agents interact with the human operator that acts as the centralized component. Similar to [20], our solution can also be used to distribute workload efficiently among agents, but our predictive system exploits a time series prediction approach to optimize the system load. Being agnostic to the agent architecture, our solution can manage both centralized and distributed management architectures. Inspired by [21], we utilize a network queuing model to estimate tasks that will temporarily or permanently disappear from the agent's queue; however, our load prediction model is different, as we model the failure and the overloading of agents and the consequent reassignment of its task with a regressor algorithm [13]. In [19] the authors proposed the use of Jackson's network model to estimate the number of tasks in the system for a replanning algorithm that proactively distributes tasks among the agents. We share with this solution the idea of proactively migrating tasks, but we differ in the model, the algorithm, and the architecture presented.

A concurrent learning adaptive control architecture is presented in HAP [17], which establishes close feedback between the high-level planning based on Markov Decision Processes (MDP) and the vehicle-level adaptive control algorithm. This feedback enables anticipating the failures and proactively reassessing vehicle capabilities after the failures, for an efficient replanning schema that accounts for changing capabilities. However, while HAP estimates vehicle capabilities using the adaptive controller's model of vehicle health, RITMO explicitly predicts future load on an agent to adapt the overall load to the system's situation.

III. Model

This section discusses the task migration problem amongst the UAVs and formulates a mathematical model to solve this problem. The system we envision is shown in Fig. 1.

A. System Model

Let A be the set of N nodes denoted as $A = \{a_1, a_2, ..., a_N\}$, where each node has to complete a set of tasks T_i. CPU, memory, and bandwidth available on the node a_i, are X_i, Y_i, Z_i respectively, where $1 \leq i \leq N$. While, $x_{m,i}$, $y_{m,i}$ and $z_{m,i}$ represent the usage of CPU, memory, bandwidth of task m on the node a_i. Let us denote the amount of tasks of the i-th node in the t-th time slot as $q_i(t)$, which are independent and identically distributed in different time slot within $[0, q_i^{max}]$, $q_i^{max} \in \mathbb{R}^+$. We consider that each task m has a processing time of $t_{proc}(m)$ seconds. The total time of execution for the task m that traversed P agents is hence defined as:

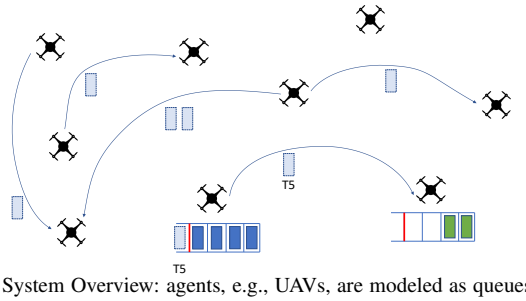


Fig. 1: System Overview: agents, e.g., UAVs, are modeled as queues containing tasks. Tasks are migrated from (currently or likely to be) overloaded or failing nodes to available nodes.

$$\mathcal{N}_m = t_{proc}(m) + \sum_{k=1}^{P} t_w(k, m) + \sum_{k=1}^{P-1} m_k, \tag{1}$$

where $t_w(k, m)$ denotes the waiting time for task m on the node k and m_k refers to the migration time when the task leaves the node k. Due to the application requirements, each task should be performed in up to R seconds.

B. Problem Formulation

In the light of the aforementioned characterization, we are ready to expose the problem that RITMO aims to solve. Formally, the optimization problem whose goal is minimizing the completion time for all the tasks in the system can be described as:

$$\min_{x} \quad \sum_{m} \mathcal{N}_m \tag{2}$$

$$\text{s.t.} \quad \mathcal{N}_m \leq R \tag{3}$$

$$\sum_{m=1}^{M} a_i x_{m,i} \leq X_i \quad \forall i = 1, \ldots, N \tag{4}$$

$$\sum_{m=1}^{M} a_i y_{m,i} \leq Y_i \quad \forall i = 1, \ldots, N \tag{5}$$

$$\sum_{m=1}^{M} a_i z_{m,i} \leq Z_i \quad \forall i = 1, \ldots, N \tag{6}$$

$$x_{m,i}, y_{m,i}, z_{m,i} \geq 0 \quad \forall i = 1, \ldots, N, m = 1, \ldots, M \tag{7}$$

We can observe that Eq. 3 imposes the completion time of any task to be below the maximum possible time R. The subsequent constraints limit the usage of resources to be at most the maximum available resources. Given these conditions, the agent i must take offloading decisions to

minimize the total completion time, and the problem can be summarized as follows.

Problem III.1. *For each period r, the node a_i has to choose if performing locally the tasks currently enqueued or migrating them to another available node. In the case of migration, the source node has to select the destination according to some pre-configured policies.*

IV. The RITMO Algorithm

Based on the previous problem and concepts, we design an algorithm to establish the migration decision. Such a decision has to determine when the migration starts and where the task should migrate.

A. Predicting the agent's load

Our migration mechanism is based on traditional regression algorithms whose aim is to predict the future values using the history and the evolution of such value in the past. The history used is composed of past values associated with the timestamp, and the presence of such a tuple $< timestamp, value >$ leads to the name time series. Among the possible methods in this class of regressor, we select ARIMA [13] for its ability to account for trend and noise in collections of data. Hence, the task of load prediction, i.e., queue's length prediction, can be formulated as a regression problem, where a real value number (future load) is predicted on the basis of many single input features (past load values).

Each epoch t, the monitoring agent collects information on the queue length. The frequency in the collection of these metrics largely depends on the time to process a task. For this reason, in the experiments, we set t to be half of the processing time to collect fresh data but not overload the node with the assignment of metrics collection. Data acquired are inserted in chronological order and comprise the historical dataset used to build the model and perform the prediction.

The prediction occurs every r seconds. We set this time interval different to t to decouple the two actions. In the experiments, we set r to be the processing time. However, this value can be relaxed in order to predict and migrate tasks less frequently. At each prediction time, the one-step-ahead forecasting is computed using ARIMA's model trained on the data collected over time. If such a predicted length exceeds a defined threshold z or the node runs out of available resources, the migration process begins.

B. Selecting the next node

The destination node of the task migration mechanism can be chosen according to different policies, herein described, where each migration policy represents a different profile. The profile refers to the desirable load on each node, considering the available CPU, memory, and bandwidth resources of the node. Hence, prior to the selection of the destination node, the system computes the time the hosting node would keep the job queued before execution. It is also possible to assign priority to avoid large queues that can hinder fast execution. In this regard, load balancing is one particular case that can happen when jobs are equally assigned among all the available nodes. The migration decision is thus performance-agnostic and only considers the current and estimated node load. Although more policies can be included in our solution, e.g., by creating and requesting via our provided APIs, in this paper we limit the focus on a small yet representative set of migration policies. The destination node can be chosen according to the following criteria:

(i) Load Balancing: the easiest schema where tasks are equally distributed among all nodes. In this case, the migration manager selects as destination the node with less enqueued tasks, and in case of more idle nodes, the destination is uniformly selected within this subset.

(ii) Harmonic: it is a well-known randomized algorithm often employed to solve the k-server problem [22]. This class of problem denotes the problem of efficiently move k servers over nodes of a graph G according to a set of requests, where a request is a sequence of k-points. It aims at minimizing the total distance covered by the servers to reach the requested points. Our strategy differs from the k-server problem but the policy still uses a version of the Harmonic algorithm to select the destination. The probability of selecting the node j as destination is given by:

$$p_j = \frac{q_j(t+1)^{-1}}{\sum_i q_i(t+1)^{-1}}, \tag{8}$$

where $q_j(t+1)$ is the predicted queue's length of node j at time $t+1$.

(iii) Cost Minimization: during the execution, a profile of the available nodes is shaped, which takes into account the computation resources. Assuming the cost of migrating depends on the average service time, the destination is chosen according to the cost of migrating: a node with a lower average service time has a higher probability to be selected. We denote the cost of migrating task m from node i to node j as $c_{mig}(i,j,m)$. The cost is computed as the sum of: *(i)* transmission time of task m migrating from i to j, $t_{tra}(i,j,m)$, *(ii)* waiting time on the node j, $t_w(j,m)$, *(iii)* processing time for task m, $t_{proc}(m)$. Formally:

$$c_{mig}(i,j,m) = t_{tra}(i,j,m) + t_w(j,m) + t_{proc}(m). \tag{9}$$

The processing time depends on the task to be executed, but for simplicity in the following we consider a fixed quantity and we often refer to it as t_{proc}. The waiting time on the destination node $t_w(j,m)$ is estimated by using our regressor algorithm. Hence, with this policy, the prediction is not only used for establishing when to migrate, but also for estimating the waiting time on the possible destination nodes.

(iv) Closest Node: the migrating task is assigned to the closest agent to the source node.

(v) Random: task is migrated to a randomly selected node. Despite being extremely easy, this strategy may result in good performance due to the small overhead introduced by the process of destination selection.

C. Overall Procedure

The algorithm's scope is to minimize the completion time of any task by proactively migrating tasks between nodes to speed-up the computation. The migration is intended to release resources of overloaded agents and exploiting spare resources of other available nodes. Two main questions underpins such a strategy: *(i) when* and *(ii) where*. *(i)* The first aspect is about when to perform the migration. We start a migration either when the predicted queue length outstrips a threshold or when the available resources on the node are insufficient to perform the task. In these circumstances, the tasks in the queue are migrated to another node of the system, whose capacity can fulfill the demand. *(ii)* When a task is migrated from a known source, the destination must be selected in order to satisfy the system requirements. However, since the decision about the destination node often privileges a key metric at the price of other quantities, the options described in Section IV-B can be chosen by the user according to the business logic. It can be, for example, that the key metric is the speed in deciding, the average usage of resources, or the average task completion time. This multitude of options originates diverse policies for the controller logic that we implemented in the system.

Algorithm 1 Prediction-based migration decision on any node

```
1:  Let t be the epoch, and r the prediction period
2:  Let z be the queue size threshold
3:  for every epoch t do
4:      Monitor the queue and node state
5:      if notAvailableResources then
6:          dst ← get_dst(node, t)
7:          migrate remaining tasks in the queue to dst
8:      if r has elapsed since last prediction then
9:          q_{t+1} ← future predicted queue size on the node
10:         if q_{t+1} > z then
11:             dst ← get_dst(node, t)
12:             migrate remaining tasks in the queue to dst
13:             close;
```

Algorithm 1 summarizes our procedure. Every epoch t, the module running on each node obtains the statistics and saves them for the next prediction. Such a prediction occurs every period r and estimates the queue size at time $t+1$. Once the regressor computes q_{t+1}, it compares this value to the threshold z, set as a quantity that notifies when many tasks are enqueued. If q_{t+1} exceeds z, tasks that are present in the queue at time t are moved to another node, as previously explained. The function $get\_dst(node, t)$ returns the destination node according to the selected policy, for example, one of the profiles described in Section IV-B. Moreover, the migration can be triggered by the absence of available CPU or memory resources on board of the node i. In this case, the $notAvailableResources$ function returns true, initiating a new migration.

V. RITMO Architecture

We designed an architecture whose aim is to enable policy-based destination decisions, based on the peculiarities of the use case. For this reason, the system consists of multiple modules that can be replaced on-demand and in a short time. In the following, we summarize our network components of the system, e.g., the APIs, and the agent mission services offered.

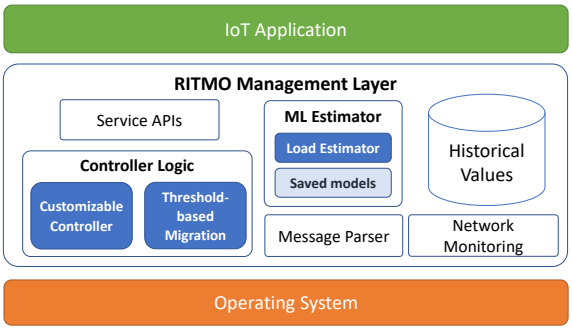
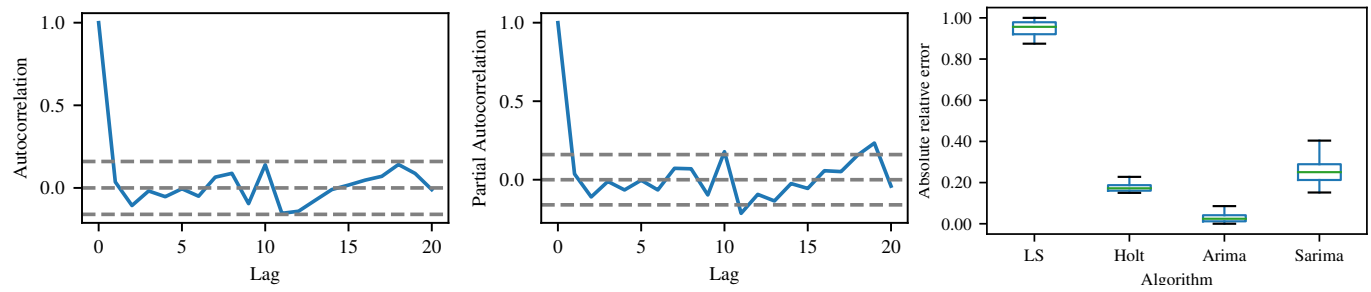


Fig. 2: System Architecture: the management layer sits between the IoT application and the operating system with the aim of monitoring network connectivity, estimating the load and migrating tasks with (re-)planning based customizable controller logic.

Fig. 2 depicts our management architecture, which enables the monitoring of network connectivity and the replan of the mission, via estimation of the load on nodes and customizable controller logic. In fact, our proposed management layer is located between the *Operating System* (at the bottom), e.g., Robotic Operating System (ROS) [23], and the *IoT application* (at the top). The IoT application running on top can take advantage of the provided API to customize the logic of such controllers, adapting to diverse business logics, as well as to adapt the mission planning logic to a centralized or distributed fashion. An example of these applications is the set-up of disaster response running live audio/video analytic.

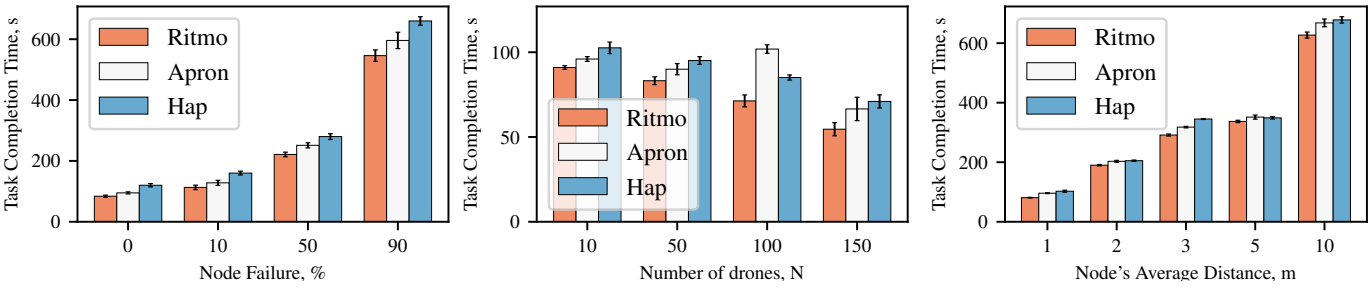
Service APIs allow the customization of two of the main components: (i) the controller logic to fit multiple challenged scenarios, (ii) the logic of the mission planning algorithm, either in a centralized or distributed fashion. By interacting with this module, the same program can tailor different contexts, adapting to different requirements and network conditions. Another relevant component is represented by the *Historical Values*, committed to the maintenance of the past network states and of the partially replicated database. These values are then used for the prediction of future states, which leverages historical dynamic states $i.e.$, states that depend on the network, configuration, and connectivity condition. Responsible for filling this database is *Network Monitoring*, that runs a watchdog process to monitor the connection states. To the rescue of understanding the messages received as heartbeat comes the *Message Parser* module. Our object model is defined through Google Protocol Buffers [24], which delimit, serialize, and deserialize the messages.

Controller Logic constitutes the adaptive component that can module the mission replanning rate of the network of IoT devices, e.g., drones. As explained in our algorithm (see Section IV-C), we employ a *Threshold-based Migration*, that impose a migration every time that the predicted number of tasks on the agent exceeds the value of a threshold z. Although our system provides a default replanning logic, it is modular



(a) (b) (c)

Fig. 3: Regressor analysis. (a) Autocorellation function (ACF) and (b) Partial autocorellation function (PACF) of measured data, used for tuning the predictor's parameters p and q (c) Error box for different time series algorithms, where ARIMA provides the higher accuracy.



(a) (b) (c)

Fig. 4: System performance evaluation. (a) Comparison of different solutions in terms of time to complete tasks at varying percentage of node failures. (b) Completion time of different algorithms for an increasing number of nodes. (c) Effect of the average distance between nodes on the task completion time.

and pluggable, and the architecture can be extended with other user-defined controllers.

The prediction of future tasks into the network occurs on the basis of the *ML estimator*, consisting of two sub-modules. *Load Estimator* represents the main feature of the solution, as it predicts the future load exploiting the current and historical values. Such a prediction attempts to estimate the relationships between the features, i.e., system state, and a dependent variable, system load. This prediction leverages *Saved models* that are already trained so that the training time can be not considered.

VI. Evaluation Results

A. Experimental Setup

To evaluate the performance of the proposed mission planning, we developed a C++ event-driven simulator, where a networked fleet of drones tries to accomplish a mission, represented by a set of geo-locations to reach. For example, in disaster response, each drone has indeed to explore an area with a camera and microphone looking for signals indicating survivors. The migration of drone's tasks is triggered when the queue length exceeds a threshold, by using our network model, and the destination node selection follows a *load balancing* profile. In this context, all drones cooperate for the completion of assigned tasks in the shortest possible time. If not otherwise specified, during our experimental campaign, we use as default values a fleet of 50 drones, whose average distance is 1 m and the percentage of failure is 0%; the destination node is selected according to the Load Balancing schema. Reported results are obtained after 35 trials and the graph's bars refer to a confidence interval of 90%.

B. Prediction Analysis & Accuracy

The choice of the ARIMA's parameters requires an initial study of the prediction performance on a validation set. In particular, the ARIMA algorithm consists of three parameters: *(i)* p captures the number of lag observations included in the model and is then denoted as auto-regressive component; *(ii)* d captures the integrated part of the model, i.e., the number of times that the raw observations are differenced, also denoted as the degree of differencing; *(iii)* q captures the moving average part of the model and refers to the extent of the moving average window, also called the order of moving average. To choose the optimal parameters, the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots were used to determine the parameters (Fig. 3a): ACF is used to determine q while PACF for p. ACF is a common method to establish how well the present value of the series is related to its past values. On the other hand, PACF measures the correlation between the time series with a lagged, i.e., past, version of itself, but after eliminating the already found. The selection of optimal p and q values occurs as follows: p is the x-value at which the function of the PACF graph crosses the upper confidence interval for the first time [25]. Similarly, q is the x-value where the function of the ACF chart crosses the upper confidence interval for the first time. Results in Fig. 3 refer to our collected dataset made of more than $40,000$ historical samples, then split into training (80%), validation (10%), and test (10%) set, and the error is computed on the test only. From Fig. 3a and Fig. 3b, it is possible to identify that $p = q = 1$. We further investigate empirically the optimal value of d using the cross-validation, and we found

that $d = 1$ provides the best performance. We then evaluate the accuracy of the selected time-series method to compare it against other time-series algorithms. A good predictor should at least outperform a very trivial algorithm in which the next value is the exact replica of the Last Sample (LS). Although this is not considered as a statistical algorithm given the simplicity of the method, it is a recommended baseline to establish the quality of the regressor method. The other time-series alternatives are: *(i) Holt-Winters (Holt)*, a basic model that captures three submodels (also known as influences) to fit a time series, i.e., an average value, a slope (or trend) over time and a cyclical repeating pattern (seasonality); *(ii) SARIMA*, which follows the same definition of the analogous ARIMA but includes seasonal components of the time series to deal with seasonal effects. In Fig. 3c we display the error of the prediction of the mentioned algorithms. The results show that the ARIMA outperforms the other solutions, as the considered metric does not exhibit seasonality and ARIMA can hence fit this context.

C. RITMO Performance

Furthermore, we compare our solution against two other solutions: HAP [17] and APRON [19]. The former anticipates failures at the planning level by establishing close feedback between the high-level planning based on a Markov Decision Process (MDP) and the execution level. The latter approach exploits a Jackson's network model to control operations of a network of IoT devices while the network states evolve. Although APRON offers several policies to select the destination, in the following, we set the closest node as the destination, since it has been shown that this setting provides better results [19].

Fig. 4a shows the time to complete tasks for the three algorithms at varying the percentage of failures. We can observe how RITMO provides the shortest completion time compared to analogous solutions, given RITMO's ability to handle a large number of failures by pro-actively and re-actively reassign the uncompleted tasks. Moreover, we compare the completion time for an increasing number of nodes in the system in Fig. 4b. RITMO can exploit all the available resources of agents without overloading them, diminishing tasks' completion time. In the graph, it can be seen how RITMO outperforms the other solutions. Besides, the advantage of our architecture with a more profitable migration comes higher when the number of nodes increases. In such a case, APRON is not always capable of exploiting the more available resources given by more drones in the system. Moreover, in Fig. 4c we analyze how the average distance among two nodes affects the system. As expected, when the distance increases, the task completion time increases as well. However, our results show how RITMO provides better performance even when the agent locations may become challenging to manage, indicating the efficiency of our proposed system.

VII. Conclusion

This paper presented RITMO, a management architecture whose attempt is to increase the resilience in task replanning and migration problems in the presence of challenged edge networks. RITMO exploits a network queue model and predicts the number of future tasks in the agent's queue. This information is thus used (by the application or the administrator) to determine the IoT device's future utilization and estimate the execution time of coming tasks that need to be executed or offloaded to the edge cloud. In the use case of a network of IoT nodes, e.g., UAVs, our results showed how RITMO is an effective mechanism for policy programmability of the mission replanning problem for any IoT device deployed in challenged networked environments.

VIII. Acknowledgement

This work has been partially supported by NSF awards CNS-1647084, CNS-1836906 and CNS-1908574.

References

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, Fourthquarter 2017.

[2] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, Feb 2018.

[3] A. Ventrella, F. Esposito, and A. Grieco, "Load profiling and migration for effective cyber foraging in disaster scenarios with formica," in *IEEE 4th Conf. on Network Softwarization (NetSoft 2018).*, Montreal, Canada, June 2018.

[4] W. Muhammad, F. Esposito, S. C. Rajashekar, and S. Gururajan, "Vocal intent programmability for uas in disaster scenarios," in *AIAA Scitech 2020 Forum*, 2020, p. 0736.

[5] N. H. Motlagh, T. Taleb, and O. Arouk, "Low-altitude unmanned aerial vehicles-based internet of things services: Comprehensive survey and future perspectives," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 899–922, 2016.

[6] A. Sacco, M. Flocco, F. Esposito, and G. Marchetto, "An architecture for adaptive task planning in support of iot-based machine learning applications for disaster scenarios," *Computer Communications*, vol. 160, pp. 769 – 778, 2020.

[7] J. Franz, T. Nagasuri, A. Wartman, A. Ventrella, and F. Esposito, "Reunifying families after a disaster via serverless computing and raspberry pis (demo)," in *IEEE International Symposium on Local and Metropolitan Area Networks*, Washington, DC, June 2018.

[8] A. Sacco, F. Esposito, and G. Marchetto, "A federated learning approach to routing in challenged sdn-enabled edge networks," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020, pp. 150–154.

[9] J.-S. Marier, C. A. Rabbath, and N. Léchevin, "Health-Aware Coverage Control With Application to a Team of Small UAVs," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 5, pp. 1719 – 1730, September 2013.

[10] N. Kemal Ure *et al.*, "Decentralized learning-based planning for multiagent missions in the presence of actuator failures," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, 2013.

[11] N. K. Ure, G. Chowdhary, J. P. How, M. A. Vavrina, and J. Vian, "Health aware planning under uncertainty for uav missions with heterogeneous teams," in *European Control Conference (ECC)*, 2013.

[12] Choi, Han-Lim *et al*, "Consensus-based decentralized auctions for robust task allocation," *IEEE Trans. on Robotics*, Aug 2009.

[13] G. E. Box, G. M. Jenkins, and G. C. Reinsel, *Time series analysis: forecasting and control*. John Wiley & Sons, 2011, vol. 734.

[14] H. Trinh *et al.*, "Energy-aware mobile edge computing for low-latency visual data processing," in *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE, 2017, pp. 128–133.

[15] S. Chung, C. Mario Christoudias, T. Darrell, S. I. Ziniel, and L. A. Kalish, "A novel image-based tool to reunite children with their families after disasters," *Academic emergency medicine*, vol. 19, no. 11, pp. 1227–1234, 2012.
[16] S. S. Ponda, H.-L. Choi, and J. P. How, "Predictive planning for heterogeneous human-robot teams," in *InfoTech*, 2010.
[17] N. K. Ure, G. Chowdhary, J. P. How, M. A. Vavrina, and J. Vian, "Health aware planning under uncertainty for uav missions with heterogeneous teams," in *2013 European Control Conference (ECC)*. IEEE, 2013, pp. 3312–3319.
[18] A. Sacco, F. Esposito, and G. Marchetto, "Rope: An architecture for adaptive data-driven routing prediction at the edge," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 986–999, 2020.
[19] A. V. Ventrella *et al.*, "Apron: an architecture for adaptive task planning of internet of things in challenged edge networks," in *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*. IEEE, 2019, pp. 1–6.
[20] C. J. Shannon, L. B. Johnson, K. F. Jackson, and J. P. How, "Adaptive mission planning for coupled human-robot teams," in *American Control Conference (ACC), 2016*. IEEE, 2016, pp. 6164–6169.
[21] A. Duminuco *et al.*, "Proactive replication in distributed storage systems using machine availability estimation," in *CoNEXT*, 2007.
[22] Y. Bartal and E. Grove, "The harmonic k-server algorithm is competitive," *Journal of the ACM (JACM)*, vol. 47, no. 1, pp. 1–15, 2000.
[23] Robotic Operating System. http://www.ros.org/.
[24] Google Protocol Buffers. http://code.google.com/apis/protocolbuffers.
[25] J. H. F. Flores, P. M. Engel, and R. C. Pinto, "Autocorrelation and partial autocorrelation functions to improve neural networks models on univariate time series forecasting," in *The 2012 International Joint Conference on Neural Networks (IJCNN)*, June 2012, pp. 1–8.