Reliable Resource Provisioning using Bankers' Deadlock Avoidance Algorithm in MEC for Industrial IoT

Emeka E. Ugwuanyi[1], Saptarshi Ghosh, Muddesar Iqbal, Tasos Dagiuklas
Division of Computer Science and Informatics, London South Bank University, London SE1 0AA, UK

ABSTRACT Multi-Access Edge Computing (MEC) is a new 5G enabling technology proposed to reduce latency by bringing cloud computing capability closer to IoT and mobile device users. MEC may be prone to unreliable communication as a result of deadlock during resource provisioning. Deadlock may occur due to a huge number of devices contending for a limited amount of resources if adequate measures are not put in place. It is crucial to eradicate deadlock while scheduling and provisioning of resources on MEC to achieve highly reliable and available system. In this paper, a deadlock avoidance resource provisioning algorithm is proposed for Industrial IoT devices using MEC platforms to ensure higher reliability of network interactions. The proposed scheme incorporates banker's resource-request algorithm using SDN to reduce communication overhead. Simulation Results have shown that system deadlock can be prevented by applying the proposed algorithm which ultimately leads to a more reliable network interaction between mobile stations and MEC platforms.

INDEX TERMS Network Reliability, 5G networks, Edge nodes, IIoT, MEC, Resource provisioning, Deadlock avoidance

I. INTRODUCTION

Reliable and instant communication has become more vital than ever in the fast-growing digital economy and connected society. Therefore, it is no surprise that network reliability is a major concern of network and internet service providers. According to [1], the key concerns of network service providers are network reliability, network usability and network fault processing. This paper aims at building a more reliable system by eliminating the chances of deadlock during resource provisioning of Industrial IoT (IIoT) to an MEC system.

Industrial IoT devices consist mainly of devices that have computation and resource limitations and therefore offload majority of their workload. In this research, we assume that the workload of these IIoT devices are offloaded to the nearest MEC node where they are provisioned resources for execution. This drastically increases the number of devices dependent on MEC node sharing and competing for resources. Tran, T et al [2] defines MEC as an emerging paradigm that provides computing, storage and networking resources within the edge of mobile Radio Access Network (RAN). The idea was to design mini servers known as edge nodes that would handle storage and computation for mobile devices. These edge nodes are in close proximity to the end users providing a platform for caching and offloading with the aim of reducing bandwidth consumption and latency of the network. The edge nodes complement the traditional cloud infrastructure by providing additional resources.

Resource provisioning in MEC depicts a multiprogramming environment where several resources may compete for reusable resources. The idea is to schedule application tasks from mobile devices to edge nodes for execution. Since there is a finite amount of resources in MEC, resources must be managed effectively to prevent scheduling a task to an edge node which does not have adequate available resources to execute the offloaded task. This environment is usually prone to deadlock because a process may request for resources which are held by another waiting resource thereby leading to a circular wait [3]. Deadlock is an undesirable problem that has been studied extensively in operating systems [3], resource allocation systems [4], and manufacturing systems [5] [6]. MEC is a distributed system [7] and studies on distributed systems have reported a chance of deadlock in such systems if proper measures are not put in place [8].

There are four necessary properties of a distributed system that could cause deadlock which includes no pre-emption, mutual exclusion, hold and wait and circular wait [3]. A simultaneous occurrence of these four leads the system to an *Unsafe State* where the system suffers from a probability of getting stuck due to unmanaged distribution of resources. Deadlock-free operation is a key characteristic for industrial sites that require high reliability and availability from its infrastructure to achieve the daily goal of the industry. The standard toolset for deadlock detection is the Wait for Graph (WFG) [3].

In the absence of algorithms to detect and recover from deadlocks, a situation may occur where the system is in a

deadlock state and yet there is no way of recognizing what has happened. In this case, the undetected deadlock will result in deterioration of the system's performance because resources are being held by processes that cannot fully execute. Therefore, if more and more processes make requests for resources, the system will enter a deadlocked state. Eventually, the system will stop functioning and would require a manual restart [3].

In this paper, a deadlock aware algorithm for scheduling resources for IIoT devices onto an MEC platform which incorporates banker's resource-request algorithm is presented. Banker's algorithm works by simulating and using specified resources to predetermine deadlock conditions for all pending activities and deciding if allocation should be allowed to continue. Banker's algorithm requires three important inputs for execution which are the *NEED matrix, MAX matrix* and *available vector (AVAIL vector)* [19]. The proposed algorithm is only favorable if implemented using Software Defined Networking (SDN) to reduce the communication overhead that would be generated by the resource-request algorithm. The remainder of this paper is structured as follows: in section II we have reviewed related work, listed our contribution and discussed the case study. In section III we presented the system model. In section IV we presented the proposed algorithm. We have simulated, tested and discussed the results in section V. We concluded in VI and future works in VII

II. Literature Review

Detection algorithms	Lamport's algorithm [20]
	Chandy-Misra-Haas algorithm [21]
	Parallel Deadlock Detection Algorithm [22]
	Detection in heterogeneous systems [23]
	Unstructured deadlock detection [24]
Prevention algorithms	Load balancing methods [25,26]
	Deadlock Prevention Algorithm in Grid Environment [27]
Avoidance algorithms	Banker's algorithm [19]

Table 1 Deadlock strategies

With the successful launch of 4G in 2010, approximately 800 telecommunication stakeholder companies around the world have formed consortiums such as 5G PPP Working Group to produce a draft for 5G architecture explaining the basic expectations [9]. This includes energy efficiency, low latency, high reliability and machine-centric communication design. To minimize latency in network communication, MEC and fog computing was proposed by ETSI and consortiums.

Considering the decentralized architecture of MEC as opposed to the traditional centralized cloud infrastructure, it is important to investigate an efficient mechanism to offload and execute mobile applications on the edge of a network.

There have been several proposals for resource provisioning techniques to offload mobile application workloads on MEC [10] [11] [12]. Nevertheless, none of the previous works on MEC considers deadlock during offloading and resource provisioning which is a concern for distributed systems as previously stated. There are four major strategies for handling deadlock in distributed systems. These include (i) ignore, (ii) detect and recover, (iii) prevention and (IV) avoidance. The first two are commonly used because the last two are difficult to implement [13]. Few researchers have opted for detect and recover strategies as shown in table 1. This is not always ideal because in a scenario where the system needs to be readily available, any amount of downtime can be very costly. Deadlock avoidance strategy is said to be the most effective, but it is difficult to implement in distributed systems because of communication overheads and therefore labelled impractical [14].

Researchers have previously used load balancing algorithms to level out the workload between servers in MEC and avoid resource over provisioning [15] [16]. C. Tham and R. Chattopadhyay [15] proposed a load-balancing scheme for distributed computing on the edge of a network based on heuristic algorithm. They used an edge model of a group of nodes connected over a wireless ad-hoc network with which they formed a convex optimization problem. The simulation results obtained show near-optimal performance in most cases. Load balancing schemes reduces the chances of deadlock but does not eliminate it entirely from the system. Deadlock prevention and/or avoidance scheme is a more suitable approach as it eliminates the chances of deadlock in the system [17].

With the advancement of 5G and Software Defined Networks (SDN), the communication overheads that was once a problem in the implementation can now be reduced thereby making it practical to implement the deadlock avoidance algorithms in a distributed system. The idea of separating the control plane from the data plane means there would be less communication between the routers and switches because they share a centralized control plane [18].

The current state of art shows that researchers have previously used load balancing to avoid over provisioning and deadlock in MEC. However, to the best of our knowledge deadlock avoidance have not been addressed in an MEC context. Therefore, in this study, a novel resource provisioning algorithm for deadlock avoidance on a multi-access edge computing is proposed in the context of IIoT. The proposed algorithm is different from load balancing because in load balancing there is a load balancer that first accepts the request and uses a mechanism to distribute it to servers. As opposed to this, in the proposed method, the task goes directly to the MEC servers for execution and only gets redirected if the time and resource constraints of the task cannot be satisfied.

The widely used deadlock avoidance algorithm due to its efficiency is the banker's algorithm proposed by Dijkstra

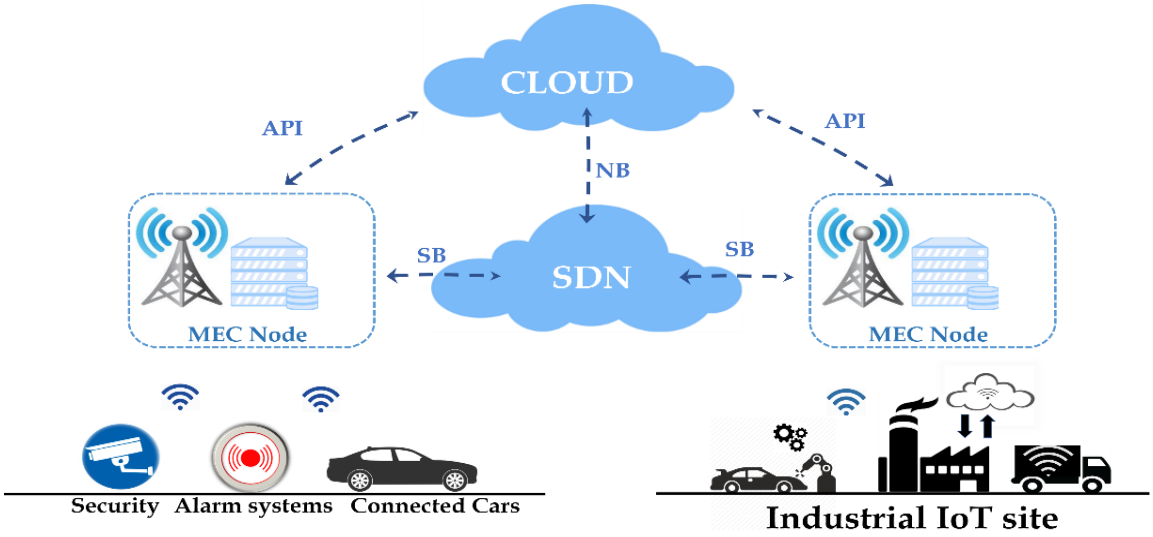

Figure 1 Case Study Architecture

[19]. Banker's algorithm is a resource allocation algorithm which simulates a system using predefined variables and predetermines the safeness of a system before granting a task allocation request [19]. It is mainly used in operating systems where it runs on a single machine. In this study, we used it in a distributed environment where resource information is shared by systems within the environment

A. *Contributions*

The main contributions of this paper are listed as follows:

a. Formulation of distributed task model in a MEC that ensures reliability by avoiding deadlock.
b. Adaptation of the banker's algorithm in the proposed solution and pushing its boundary by testing it in a new field (MEC) and obtaining an optimized solution for distributed systems.
c. Extensive simulations conducted on the algorithm shows reduced probability of deadlock occurrence.

B. *Case Study Architecture for Industrial IoT and MA-MEC*

Figure 1 shows a high-level view of the MEC topology adopted in this study. In this scenario, due to resource and computation limitation of the IIoT devices, they heavily depend on MEC nodes to execute their workload. Therefore, tasks are offloaded from the IoT devices to be executed on an MEC platform. The distributed edge nodes communicate with each other through SDN. IoT requests that are not available on the edge node would be forwarded to the cloud through the API (Application programming interface). To reduce latency in this research, traffic to the cloud is generally avoided. The SDN controller uses it North Bound (NB) interface to communicate with the cloud and communication with the edge nodes is done using the South Bound (SB) interface. Each edge node comprises of a monitoring tool which calculates the resource utilization of the node (CPU, RAM and Memory). This information is shared between the edge nodes as metadata. Therefore, each edge node that forms a part in the network is assumed to keep resource utilization information about the entire destination within the system. This helps the edge nodes decide the most suitable edge if re-offloading is required.

Optimal routes are also considered when sharing metadata among edge nodes. Each edge node in the network sends updated metadata after each event. This metadata describes the resource utilization of the edge node after the event. The term network is used here loosely to describe the Multi-access edge architecture.

III. SYSTEM MODEL

In this work, a distributed architecture which consists of a pool of Multi-Access Mobile Edge Computing (MEC) nodes is considered as a platform for resource provisioning. The tasks seeking to be offloaded will utilize the MEC resources through a request-response mechanism. Hence the problem can be modelled as a Directed Regular Graph. The target scenario stands out to be soft real-time and high volume of offloading traffic from an underlying scalable network.

Let's consider a mesh network of a finite non-empty set of edge nodes *CL* =$\{Cl_1, Cl_2 \dots Cl_n\}$ and a finite non-empty set of mobile station *M* = $\{ms_1, ms_2 \dots ms_n\}$ connected to the edge network such that $ms_i \in M$ and $Cl_j \in CL$ maintains a disjoint many-to-one cardinality. Here an edge node is connected to many mobile stations, but no mobile station is connected to multiple edge nodes. Communication between *CL* and *M* happens over a wireless band with a fixed number of channels $\{ch_i | 1 \leq i \leq k\}$ and collision is prevented by CSMA/CA protocol [28]. The CSMA/CA maintains a back off time less than the real-time deadline τ_{dl} making the system scalable and dynamic. The system model comprises of the communication model and

the computation model. The communication model deals with the optimization of communication parameters for better energy savings and the computation model for optimizing the execution time with deadlock immunity.

a. Communication Model

Let's consider a workload $W = \{T_1, T_2 \dots T_n\}$ which contain a set of tasks T_i to be offloaded by a mobile station. The workload, denoted by $W\ [c_i, m_i, n_i, d_i]$ is characterized by CPU, memory, network and data size respectively. During IIoT application development, the developer specifies which fraction of the total workload can be offloaded (*Remotable Object*) and which part should be executed locally. Therefore, the offloadable data size of any task T_i can be denoted as a fraction α_i of total workload data size d_i. Therefore, $l_i = \alpha_i d_i$ is the offloadable data size of T_i. The transmission time $t_i = \left(\frac{l_i}{r_i}\right)$ where r_i is the transmission rate which can also be expressed as

$$r_i = B \log_2 \left(1 + \frac{P_i g_i^2}{N_0 B}\right) \tag{1}$$

where B is the bandwidth, g is the gain and P is the transmit power. Hence the equation can be rewritten for P_i as (eq 2)

$$\frac{N_0 B\left(2^{\frac{r_i}{B}} - 1\right)}{g_i^2} = \frac{1}{g_i^2} h\left(\frac{l_i}{t_i}\right) \tag{2}$$

where

$$h(x) = N_0 B\left(2^{\frac{x}{B}} - 1\right) \tag{3}$$

which is monotonically increasing with x. Hence the energy consumption for the offloading task is (eq 4)

$$E_{i,off} = \frac{\alpha_i d_i P_i}{r_i} = t_i p_i = \frac{t_i}{g_i^2} h\left(\frac{l_i}{t_i}\right) \tag{4}$$

Therefore, $E_{i,off} = O(t_i)$ (Lemma 1). Energy optimization can be obtained by the following model.

$$maximise\ E_{saved} = \sum_i \left(E_{local} - E_{offload}\right)$$

subject to,

$$\tau_{local} + \sum \tau_{route} + \sum \tau_{wait} \leq \tau_{dl}.$$

where τ_{local} is the time spent to calculate if the task should be offload, while τ_{route} is the time spent in routing the task from the local device to the edge for execution. τ_{wait} is the time the task spends on the edge node before being executed. The edge nodes are assumed to be in a mesh topology, hence $\tau_{route} = O(1)$, whereas $\tau_{wait} = O(n^k)$. As deadlock freezes the system, the waiting time keeps increasing by 2^k until it reaches the maximum k value and times out due to CSMA *binary exponential back-off* characteristics [32].

b. Computation Model

Computation starts after the offloaded data stream is received by an edge node. Here a decision is made whether the requested task gets executed on the subjected edge node or re-offloaded to another one. The decision is made based on resource request WFG of each individual edge node and availability of the other nodes in the mesh. Hence the system is a mesh of interconnected priority queues. Note that the WFG is made for each MEC node and not distributed across all nodes. The priority is based on a safe sequence from banker's algorithm which guarantees no deadlock using a preventive and avoidance measure. The precomputing delay contributes to τ_{wait} and ensures it is below deadline. The edge node maintains two queues. First, a prioritized indefinite length job queue whose priority is maintained by the publisher (Rate Monotonic Criteria). To achieve real-time criteria, Rate Monotonic Scheduling (RMS) suggests that frequent occurring tasks should be given higher priority [29]. Tasks get popped out in Job queue in FIFO order and then checked if the requested resource can be accommodated by the subjected edge node Cl_i. If not, it finds another edge node Cl_j that is most eligible and offloads. If Cl_j executes the task on time, then Cl_i increases the j^{th} index on its $Affinity_i$ vector that it maintains, decreases otherwise. This affinity vector is initialized with 0 and used to maintain reliability record and tie-breaker purpose. A *Request* ≤ *Available* is said to be valid and put into the Ready Queue which is finite with size $Size_r$ and prioritized with Banker's generated safe sequence.

$$Size_r = \left\lfloor \frac{BDP}{mean(l)} \right\rfloor = \left\lfloor \frac{nB\ RTT}{2\sum_{i=1}^{n} l_i} \right\rfloor \tag{5}$$

BDP shows the number of bits the channel can accommodate, hence the ratio of BDP and average task is the number of task that can be queued ensuring mutual exclusion property. When a task is inserted into a ready queue, it gets an index based on its resource requirement. Starvation is handled with aging. If a task T_i gets placed into a ready queue with index i, then the expected turnaround time $TT(T, k) = i * awt_k$. Where awt_k is the average waiting time of edge node Cl_k.

In worst case scenario, for n processes and m resources *Banker's algorithm* takes $O(n^2 m)$ time. Since the number of resources are fixed (k), hence the time complexity is $O(n^2 k) = O(n^2)$. Since the algorithm is applied on the ready queue the maximum task it can retain is $Size_r \times \alpha\ delay = t_i$, Hence banker's algorithm takes $O(t_i^2)$ to generate a safe sequence.

Lemma 1: *The consumed energy for offloading and the transmission time shares a linear relationship.*

Proof. From equation 3 & 4 it can be inferred that, the partial relationship between $E_{i,off}$ & t_i for a given gain (g_i) and offload length (l_i) is,

$$E_{i,off} = \frac{t_i}{g_i^2} h\left(\frac{l_i}{t_i}\right) = t_i 2^{\frac{1}{t_i}}$$

Using asymptotic analysis of the given function,

$$O(E_{i,off}) = O(t_i) \times O\left(2^{\frac{1}{t_i}}\right)$$

Now the second element is a monotonically decreasing sequence with lower bound 0. Hence, it has a constant asymptotic upper bound $c \in R$, therefore $O(1)$.
Hence,

$$O(E_{i,off}) = O(t_i) \times O(1) = O(T_i) \quad (6)$$

This can be verified by plotting equation 4. (Figure 2)

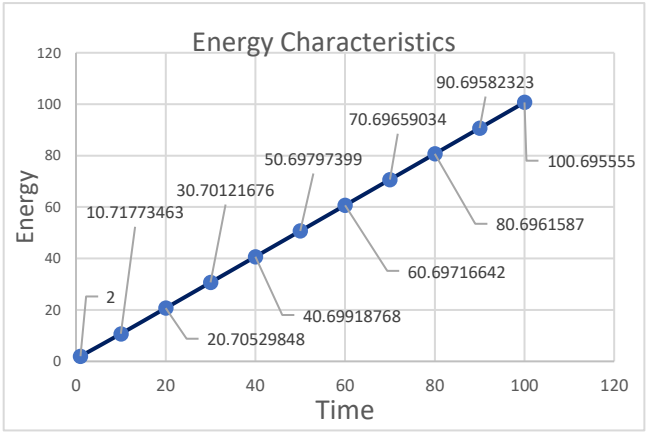

Figure 2 Energy characteristics vs time

IV. Proposed algorithm

In this section we discuss the design and analysis of the proposed resource provisioning algorithm (RPA). The algorithm fetches tasks from the task queue which is RMS scheduled, therefore most frequently used tasks get higher priority. Tasks from Job queue then migrates to ready queue. The proposed algorithm alters the order in which the tasks leave the job queue and stays in the ready queue. The following are the criteria used for the ordering.

Case 1. *Overdemand*: each task comes with its maximum resource need, recorded in the *MAX* vector. If the maximum need exceeds the total available resources, then it searches for an MEC node which satisfies the constraint. If no such MEC node is found the task waits for a certain amount of time which increases in a binary exponential order with each iteration of request before it times out.

Case 2. *Unsafe Request:* if the *MAX* is less than the current node's *AVAIL* then the tasks enters Banker's safe state algorithm and be given a safe sequence index at which the task gets executed. Banker's algorithm guarantees a safe sequence never causes deadlock.

Case 3. Time feasibility: A resource hungry task in a resource constrain MEC may suffer from starvation by waiting. Aging is used here to improve waiting time, although it requires the process to stay waiting to age. Hence the algorithm calculates waiting time by the product of the average waiting time of the current node and the index of the task. If the waiting time exceeds the soft deadline of the task, it finds an alternative node to meet the criteria.

Algorithm 1: Resource Provisioning Algorithm (RPA)

Input: *W* $[c_i, m_i, n_i, d_i]$
Output: *Resource Provision Plan for* t_i
Steps
1. Do

2. *Job. Insert(*t_i*)*
3. $k \leftarrow 0$ *; max_k = input('maximum retry attempt :')*
4. ***While*** *(Ready.isfree() = true)* ***do***
5. *Ready. Insert(Job.delete(*t_i*));*
6. *J-cur* ← *Ready.delete (*t_i*);*
7. *J-Cur. Status = Assigned ;*
8. ***If*** J_{cur} *.MAX< node.AVAIL:*
9. *Ind = banker's (*j_{cur}*)*
10. *Time = (AWT)X(Ind)*
11. ***If*** *Time <* t_i^l *;*
12. *Assign;*
13. ***Else*** *Goto step 14*
[End If]

Else

14. *Find* Cl_i *from CL [nodes] :*
15. $\max_i(Cl_i.AVAIL - J_{cur}.MAX)$
16. *Send(*J_{cur}*);*
17. *Wait until (response)*
18. ***If*** *response = Success :*
19. *Return result*
20. ***Else*** *wait*(2^{k++}) *// k : iteration count*
21. ***If*** *(Timeout OR k = max_k):*
22. *Return "Fail"*

[End if]
[End if]
[End If]
[End Loop]
While (True)

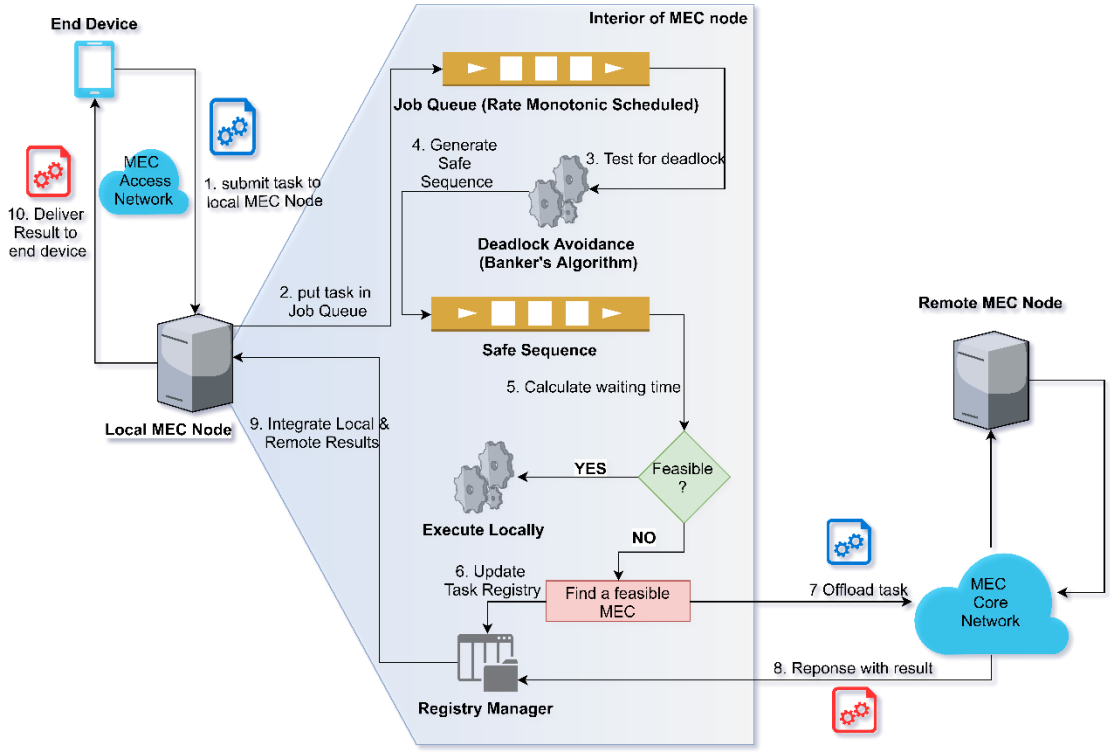

Figure 3 RPA workflow

A task is said to be *feasible* if it doesn't *overdemand* and the generated waiting time is less than its latency constraint. The waiting time of a task is the product of average waiting time of the executing node Cl_i and the safe index bankers' algorithm produces. The algorithm allows a *feasible* task to execute locally else it gets executed remotely. A task that demands resources that are not available on the local MEC or a task with unsuccessful execution by a remote MEC must be kept on waiting until it's timeout. The waiting period increases with a binary exponential order with each attempt. A registry is also maintained to keep track of the tasks submitted for remote execution and their status. Figure 3 depicts the complete workflow of RPA.

Lemma 2: *RPA is not suitable for hard real-time but soft real-time tasks.*

The response time of the algorithm depends on various timing factors such as

i. *Queuing Delay:* Takes place due to processing overhead, context switching etc. of other processes rather than the subjected one. It also depends on system specification and load.
ii. *Transmission Delay:* An offloaded task's total execution time includes the transmission delay which varies with network conditions.

The given uncertainty conditions makes a hard deadline infeasible as opposed to a soft deadline (lemma 3), hence the statement.

Lemma 3: *If there exists a feasible MEC node for a task, RPA handles the task within a finite time.*

To prove the lemma, we'll prove for each three feasibility cases discussed earlier, a task waits a finite amount of time under RPA.

Case 1: If the task over demands resources to its original MEC node and a remote node failed to execute, it must wait twice the time for resubmission hence the timeout occurs in $\log_2 timeout$ iteration.

Case 2: if the task makes an unsafe request, it looks for a remote node to get offloaded. Since all the *AVAIL* information are reactively shared and the decision is made based on the global map of *AVAIL*s. Therefore, the task gets offloaded only once and onto the optimal remote MEC node. This prevents node hopping and total execution time can be $T_{total} = T_l + T_r + 2C_{lr}$ where T_l, T_r & C_{ij} are local execution, remote execution and transmission time respectively.

Case 3: If a task makes a safe request but has a large *NEED*, it must wait for the resources to be available. If a remote node can execute it in less time, it is offloaded ($T_l < T_r$). Therefore, this guarantees the optimal remote node selection.

V. SIMULATION

Simulations were performed to demonstrate the validity of the proposed technique. The simulations were based on the complexity analysis of the algorithm and energy optimization as discussed in the previous section. The energy, $E_{i,off}$ required by an edge node Cl_i to offload a task of l_i $size$ for t_i unit time through a channel of B_i bandwidth using an antenna of g_i and a signal to data ratio N_0 is (eq 7).

$$E_{i,off} = \frac{t_i}{g_i^2} N_0 B \left(2^{\frac{l_i}{Bt_i}} - 1 \right) \tag{7}$$

Since gain, bandwidth, data size and signal-to-data ratio is predetermined by the communication system hence the relation can be squeezed into an asymptotic upper bound form as (eq 8).

$$E_{i,off} = O\left(t_i^l . 2^{\frac{l_i}{t_i^l}} \right) \tag{8}$$

The graph in figure 2 shows a critical value of transmission time and payload length the energy consumption by the antenna starts rising exponentially. Context suggests that if there's a deadlock then the waiting time component will increase indefinitely resulting to a significantly large energy consumption. Since the transmission time is a function of the data length and a constant data rate, therefore the transmission time is a random variable distributed over a Bernoulli's probability density function (collision control is CSMA/CA). To find the expectation (E) this can be shown that the surface integral mentioned below cannot be expressed in a closed form (eq 9).

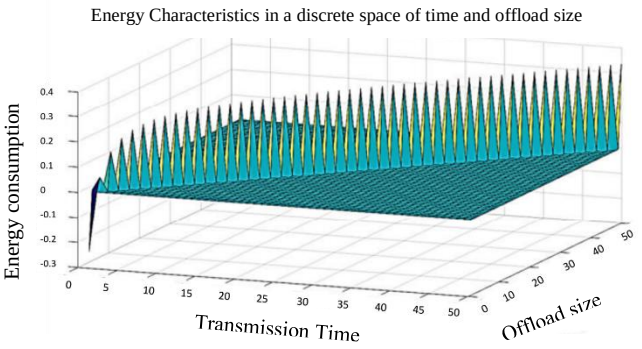

Figure 4: offload energy characteristics

$$\int_0^{\frac{BDP}{n}} \int_0^{latency} t_i 2^{\frac{l_i}{t_i}} dt_i dl_i \tag{9}$$

Equation 9 states the Growth rate of $E_{i,off}$. Plotting this growth characteristic within a close range of [0, 50], the response characteristic surface in figure 4 is obtained. Each spike on the graph depicts the exponential growth of energy discussed earlier. With an increase of transmission time and length the peak energy consumption grows at a constant rate of *ln 2*. The growth characteristics of the $E_{i,off}$ in equation 8 can also be shown by the partial derivatives with respect to latency (t_i) and offload length (l_i)

$$\frac{\partial^2 E_{i,off}}{\partial l_i \partial t_i} = -\left(\ln 2 \frac{2^{\frac{l_i}{t_i}}}{t_i} \right) \tag{10}$$

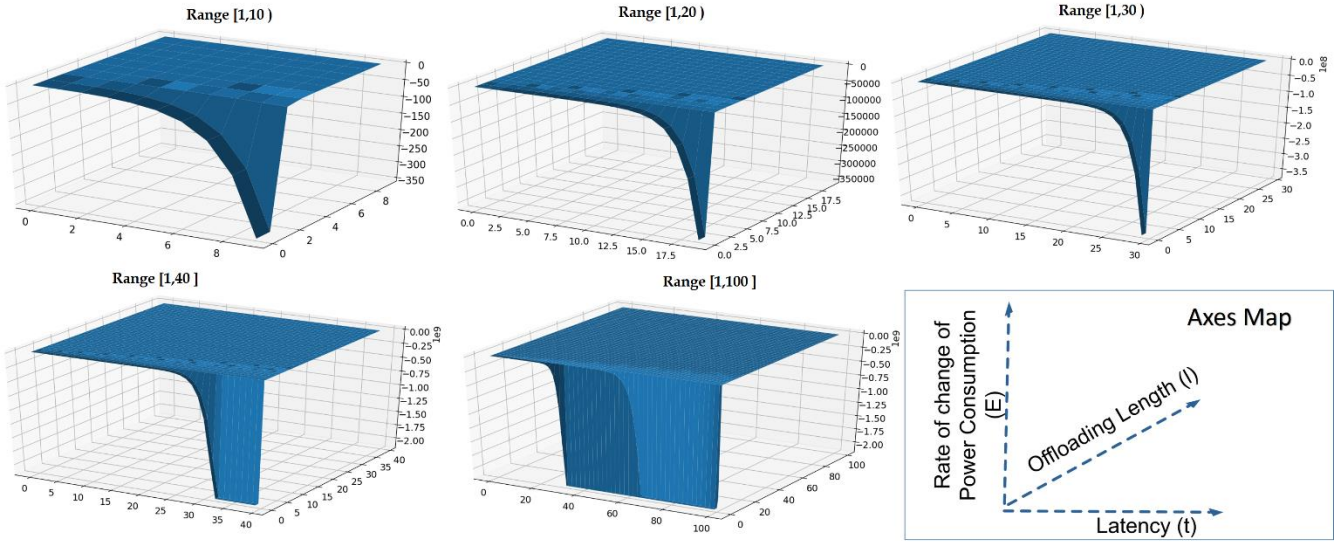

Figure 5 growth characteristics of the rate of change in offload energy with varying latency and offload length in a close range. Slope gets steeper with increasing range, saturates at [1,40]

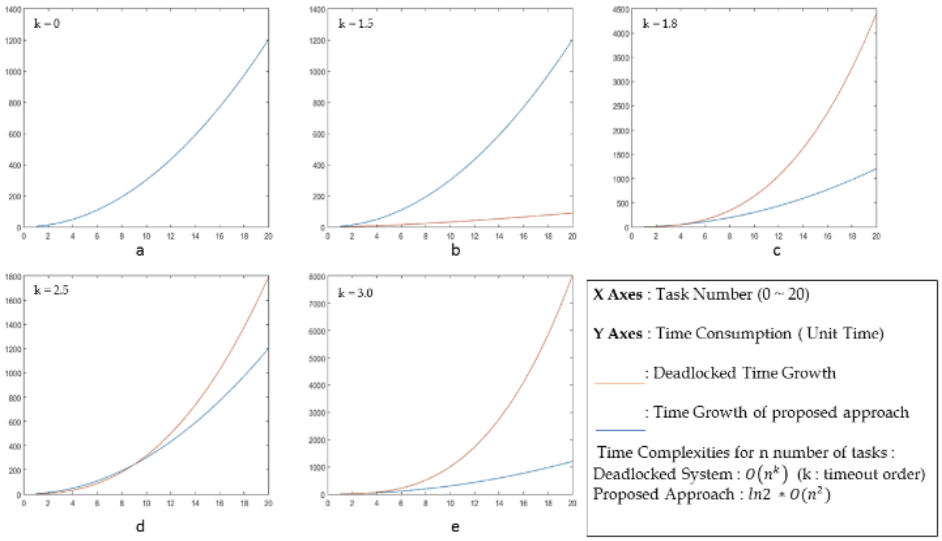

Figure 6: comparison of time consumption of system with & without using RPA

The surface plot of the equation 10 is depicted on figure 5. Analytically equation 10 signifies the rate of change of energy consumption with respect to varying offload length and latency. The value of (t_i, l_i) is taken in a close range. It can be shown in table 2 that the growth gets steeper as the range increases. It can be observed that the plot saturates after the range [1,40].

Table 2: growth characteristics of change in energy in discrete time & size.

(t_i, l_i) range	Rate of change of $E_{i,off}$
[1,10)	3.5×10^2
[1,20)	3.5×10^5
[1,30)	3.5×10^8
[1,40]	3.5×10^9
[1,100]	3.5×10^9

c. Experimental Results

Since there exist no related experiments with MEC context in the literature, we performed the experiment by comparing results of a system with and without using RPA.

Fig 6 shows time comparison graphs between a system with no deadlock prevention measures and a system running the proposed algorithm. The graphs were plotted with their corresponding time complexities for *n* number of tasks subject to a constant *k* (timeout order: this value is application dependent). It can be seen in fig 6 that as *k* increases, the time consumption of the system with no deadlock measures surpasses the system running the proposed algorithm. Since time is directly proportional to energy, it can be deduced that the algorithm optimizes the energy of a system by eliminating deadlock.

d. Complexity analysis of the proposed model

If N_{Total} tasks are submitted to an edge node, the job queue will hold them in priority as generated by Rate Monotonic Scheduling (RMS) Algorithm which takes t_{rms} time. Based on tasks' request and the subjected edge node's availability or resources, N_{oc} tasks are offloaded to an eligible edge node Cl_j as overcommitted task. An efficient binary search implementation can find such Cl_j in $\log_2 c - 1$ time. The remaining $N_{Total} - N_{oc}$ tasks will be put into banker's algorithm that takes t_{ba} time to find the safe sequence in worst case scenario. If a task T_k gets a safe index *k*, and the *TT(k, i)* > *Q*(deadline of T_k) then, the task will be offloaded to another MEC node that can perform the execution within the deadline. The function queue calculates the probability of executing the task and maintaining the deadline after all the communication and queuing. This is done by maintaining the affinity matrix. Hence, the time complexity of Q is $N_{reoff} \log_2 N_{reoff}$ where N_{reoff} is the number of tasks to be re-offloaded. Therefore, the maximum time a task can take to be executed if it got offloaded twice and being the lengthiest task can be expressed as

$$T_{max} = 2[ln2 + \log_2(c-1) + 3(N_{Total} - N_{oc})^2 + N_{reoff} \log_2 N_{reoff} + rtt] + t_{exec} \quad (11)$$

The worst-case complexity of RMS and Bankers algorithm can be deduced to *ln2* and $3n^2$ respectively

VI. CONCLUSION

In this paper, a Resource Provisioning Algorithm for Deadlock Avoidance for Multi-Access Edge Platform was presented with an aim to maintain a more reliable network system for IIoT devices. As edge nodes have a finite amount of resources, continuous increase in the number of users dependent on the edge resources might lead to over-provisioning which may result in a system deadlock because of many devices contending for limited and shared resources. The simulation results confirm this behaviour. In this paper we build on previous work on MEC by using a modified resource request banker's algorithm which can also re-

distribute tasks to satisfy the latency constraint. Simulation test confirms deadlock if the system is in an unsafe state and there is a continuous increase of IIoT applications dependent on the edge node. On applying the proposed algorithm, results show that system deadlock can be avoided, which ultimately leads to a more reliable network interaction between IIoT devices and MEC platforms.

VII. FUTURE WORKS

The proposed RPA is an algorithm for distributed systems and not a distributed algorithm. Further works on this topic will be to improve the algorithm into distributed algorithm for MEC which can map the WFG of all the MEC nodes together rather than individual MEC WFG node mapping. Another direction would be comparison of RPA with Banker's integration to another version of RPA using different deadlock avoidance algorithms.

REFERENCES

[1] C. R. Kalmanek and R. Y. Yang, Guide to reliable internet services and applications: The Challenges of Building Reliable Networks and Networked Application Services. [Place of publication not identified]: Springer London Ltd, 2013.

[2] Tran, T., Hajisami, A., Pandey, P. and Pompili, D. (2017). Collaborative Mobile Edge Computing in 5G Networks: New Paradigms, Scenarios, and Challenges. IEEE Communications Magazine, 55(4), pp.54-61

[3] A. Silberschatz, P. Galvin and G. Gagne, Operating system concepts, 8th ed. Hoboken, N.J: Wiley, 2014, pp. 283-310.

[4] S. Reveliotis and Z. Fei, "Robust deadlock avoidance for sequential resource allocation systems with resource outages", *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, 2016.

[5] Y. Yang and H. Hu, "Distributed deadlock avoidance in automated manufacturing systems with forward conflict free structures using Petri nets", 2016 European Control Conference (ECC), 2016.

[6] Zhonghua Huang and Zhiming Wu, "A New Distributed Deadlock Avoidance Strategy for Flexible Manufacturing Systems Using Digraph Models", 2006 8th International Workshop on Discrete Event Systems.

[7] W. Yu, F. Liang, X. He, W. Hatcher, C. Lu, J. Lin and X. Yang, "A Survey on the Edge Computing for the Internet of Things", IEEE Access, vol. 6, pp. 6900-6919, 2018

[8] V. Kate, A. Jaiswal and A. Gehlot, "A survey on distributed deadlock and distributed algorithms to detect and resolve deadlock", *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, 2016.

[9] Group, 5. P. A. W., 2017. 5G PPP Architecture Working Group View on 5G Architecture (Version 2.0), s.l.: 5G PPP Architecture Working Group

[10] J. Liu, Y. Mao, J. Zhang and K. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems", 2016 IEEE International Symposium on Information Theory (ISIT), 2016.

[11] A. Al-Shuwaili and O. Simeone, "Energy-Efficient Resource Allocation for Mobile Edge Computing-Based Augmented Reality Applications", IEEE Wireless Communications Letters, vol. 6, no. 3, pp. 398-401, 2017.

[12] Y. Mao, J. Zhang and K. Letaief, "Dynamic Computation Offloading for Mobile-Edge Computing With Energy Harvesting Devices", IEEE Journal on Selected Areas in Communications, vol. 34, no. 12, pp. 3590-3605,2016.Meulen, R. (2017).

[13] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. YANG and W. Wang, "A Survey on Mobile Edge Networks: Convergence of Computing, Caching and Communications", IEEE Access, vol. 5, pp. 6757-6779, 2017.

[14] C. Sanchez, H. Sipma and Z. Manna, "Generating Efficient Distributed Deadlock Avoidance Controllers", 2007 IEEE International Parallel and Distributed Processing Symposium, 2007.

[15] C. Tham and R. Chattopadhyay, "A load balancing scheme for sensing and analytics on a mobile edge computing network", 2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2017.

[16] R. Beraldi, A. Mtibaa and H. Alnuweiri, "Cooperative load balancing scheme for edge computing resources", 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), 2017.

[17] M. Altamimi, "A Task Offloading Framework for Energy Saving on Mobile Devices using Cloud Computing", Ph.D, University of Waterloo, 2017.

[18] J. Doherty, SDN and NFV Simplified. [S.l.]: Addison-Wesley Professional, 2016, pp. 149 - 154.

[19] Dijkstra, Edsger W. "Cooperating sequential processes." The origin of concurrent programming. Springer New York, 1968. 65-138.

[20] L. Lamport, "Time, clocks, and the ordering of events in a distributed system", *Communications of the ACM*, vol. 21, no. 7, pp. 558-565, 1978.

[21] K. Chandy, J. Misra and L. Haas, "Distributed deadlock detection", *ACM Transactions on Computer Systems*, vol. 1, no. 2, pp. 144-156, 1983.

[22] H. Nguyen, H. Dang, N. Pham, V. Le and T. Nguyen, "Deadlock Detection for Resource Allocation in Heterogeneous Distributed Platforms", Advances in Intelligent Systems and Computing, pp. 285-295, 2015.

[23] H. Nguyen and V. Le, "Detection and Avoidance Deadlock for Resource Allocation in Heterogeneous Distributed Platforms", International Journal of Computer Science and Telecommunications, vol. 6, no. 2, 2015.

[24] J. Lim, T. Suh and H. Yu, "Unstructured deadlock detection technique with scalability and complexity-efficiency in clouds", International Journal of Communication Systems, vol. 27, no. 6, pp. 852-870, 2013.

[25] K. Rashmi, V. Suma and M. Vaidehi, "Enhanced Load Balancing Approach to Avoid Deadlocks in Cloud", International Journal of Computer Applications, 2012. https://arxiv.org/ftp/arxiv/papers/1209/1209.6470.pdf

[26] O. Mahitha and V. Suma, "Deadlock avoidance through efficient load balancing to control disaster in cloud environment", 2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT), 2013.

[27] D. Malhora, "Deadlock Prevention Algorithm in Grid Environment", MATEC Web of Conferences, vol. 57, p. 02013, 2016.

[28] B. Forouzan and S. Fegan, *Data communications and networking*, 4th ed. Boston [Mass.]: McGraw-Hill Higher Education, 2007. 370-379

[29] J. Lehoczky, L. Sha and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior", [1989] Proceedings. Real-Time Systems Symposium.