2020

Motivations and Challenges in Unmanaged Edge Computing

Shikhar Suryavansh

Kwang Taik Kim

Motivations and Challenges in Unmanaged Edge Computing

Shikhar Suryavansh, Kwang Taik Kim -
Purdue University -

In this document, we consider a motivating example for the unmanaged edge computing scenario and look at the unique challenges introduced by the unmanaged edge.

Motivating Example

Consider a typical application from the domain of autonomous self-driving cars [30]. It has the tasks listed below and we use this application in our evaluation (one of three).

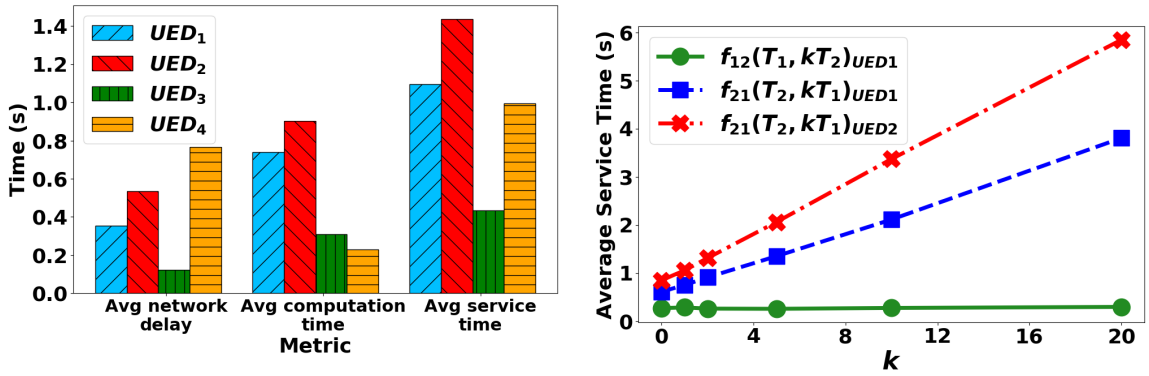
(a) Driver state detection using face camera

(b) Driver body position using driver cabin camera

(c) Driving scene perception using a forward-facing camera

(d) Vehicle state analysis using instrument cluster camera

Task (c) can further consist of multiple tasks like pedestrian detection, obstacle detection, traffic signs analysis, etc. All these tasks would operate on the same input data, *i.e.* the feed from the forward-facing camera. In this work, we focus on how to offload user requests pertaining to the latency-sensitive applications (such as the example above), in a heterogeneous unmanaged edge computing scenario. We aim at minimizing latency while providing a configuration parameter that determines how bandwidth conserving the allocation of tasks to UEDs is.



(a) Computational and geographical heterogeneity

(b) Heterogeneity in interference pattern

Fig. 3.1.: Challenges in unmanaged edge orchestration

3.0.2 Challenges and Responses

The notion of unmanaged edge introduces a set of unique challenges unseen in traditional edge computing. Following are the main challenges involved in the orchestration of tasks in an unmanaged edge scenario and a brief statement about how we handle each challenge.

Substantial heterogeneity in computational capacity and geographical distance of edge devices: The edge devices, which are personal laptops, tablets, desktops, etc., in our case, consist of heterogeneous hardware and hence, the performance of a task varies significantly on different edge devices. Also, different edge devices are at different geographical distances from the orchestrator. Consequently, the network delay also varies. Figure 3.1a shows the average service time (average network delay + average computation time) of executing an image classification task on four heterogeneous edge devices at varying distances from the orchestrator in a production setting. The four UEDs are Samsung Galaxy Tab S4-2018 (UED_1), Dell Inspiron 15R-2013 (UED_2), Macbook Pro-2018 (UED_3) and iMac-2017 (UED_4). Note the huge disparity between the average network delay (max-min ratio 6:1) due to geographical heterogeneity and the average computation time (max:min ratio 4:1) due to computational heterogeneity among the UEDs.

Heterogeneity in task interference pattern: Different tasks, when running on the same edge device, may interfere with each other affecting their service time. There is a heterogeneity in the interference experienced by different types of tasks on a UED. For instance, Figure 3.1b considers task T_1, an image segmentation task, which is simpler compared to T_2, an image classification task. It shows the difference between the interference of tasks of type T_1 on T_2 $\left(f_{21}(T_2, kT_1)_{UED1}\right)$ and T_2 on T_1 $\left(f_{12}(T_1, kT_2)_{UED1}\right)$ on UED_1. The interference is quantified using $f_{ij}(T_i, kT_j)_{UEDp}$ which gives the execution time of a new task of type T_i on UED_p, given that k tasks of type T_j are already running on the UED. It can be seen from the figure that there is a high interference of T_1 on T_2 but almost negligible interference of T_2 on T_1. Not only do different types of tasks interfere differently on the same device, but also there is variation in interference pattern across multiple devices. Figure 3.1b shows the comparison between the interference of T_1 on T_2 on two different $UEDs$ $\left(f_{21}(T_2, kT_1)_{UED1}\right.$ and $\left.f_{21}(T_2, kT_1)_{UED2}\right)$. The interference of T_1 on T_2 is higher on UED_2 than that on UED_1. Thus, interference depends on the ordered pair of tasks and also the UED. I-BOT performs a novel interference profiling of the UEDs to handle this heterogeneity in interference pattern (Section 5.3).

Online variations in the usable capacity of an edge device: Depending upon the personal applications that the owner is running on a UED, the amount of resources available for edge services will vary. To prevent a slowdown of the UED, we need to reduce the usage of the device if the owner starts running a computationally demanding personal application. I-BOT handles this using online readjustment based on a feedback mechanism (Section 5.6).

Lack of monitoring information from edge devices: Most of the current edge orchestration schemes [18–21] utilize monitoring information, such as CPU usage, frequency, memory consumption, etc., from the edge devices to make offloading decisions. However, we do not use any such information because of the following reasons:

1. As the edge devices in our case are not managed by a single entity, the monitoring information may not be readily available. Also, the owners of the devices

may be privacy sensitive about sharing such information with a third party. Note that they have signed up to contribute some compute resources to the unmanaged edge platform, but that can rarely be interpreted to mean that the device owners want the usage on their devices to be monitored.

2. Monitoring a large number of edge devices with the level of frequency needed to be useful would result in a huge overhead. The devices would have to transmit monitoring information continuously as their usable capacity is susceptible to variations, due to co-located applications starting up and other factors that do not occur at a set frequency.

In I-BOT, the orchestrator learns from external observation and predicts the service time of tasks without using any monitoring information from the edge devices (Section 5.5).

Sporadic availability of unmanaged edge devices: Unlike the traditional servers in a managed edge setting which are always available, the availability of an unmanaged edge device would depend upon the owner of that device. Hence, we cannot rely on the device being available for computation all the time. Depending upon the work pattern of the owner of a device, it may be available intermittently at different times of the day. Based on the history of the availability of UEDs, we predict their future availability and use it in our orchestration scheme (Section 5.4).

SYSTEM OVERVIEW

In this section, we present a high level overview of the main components of I-BOT. Figure 4.1 shows the timeline exhibiting the steps involved in adding a new UED to the system, orchestrating tasks to the available UEDs, performing online readjustment and gracefully removing a UED when it wishes to exit the system. As shown in Figure 4.1, when a new UED enters the system, our orchestrator profiles it using our novel interference-based profiling method (Section 5.3) and adds it to the UED profile database which stores the profiling information of all the added UEDs. This method of profiling handles the heterogeneity in the computational capabilities and interference patterns among the UEDs. When an application instance (consisting of N different tasks) from an end user arrives at the orchestrator, the orchestrator first predicts which UEDs would be available throughout the execution of the application instance. It then updates the available UED set to include only those UEDs which have a high probability of not leaving the system. This handles the sporadic availability of the UEDs, an inherent characteristic of unmanaged edge computing systems. An initial schedule for the N tasks is then determined using the UED profile database and the data structure containing the number of tasks of different types already running on the available UEDs. This data structure is updated by the orchestrator whenever it sends a new task to a UED or receives an execution result from a UED. The initial schedule is a many-to-one mapping of the N tasks to the available UEDs, aimed at minimizing the service time of the tasks. Next, I-BOT updates the schedule to reduce the bandwidth overhead at the cost of a slight increase in the service time by trying to schedule the tasks that require the same input data on the same UED. I-BOT includes a bandwidth overhead control parameter that manages this trade-off. The tasks are then sent to the selected UEDs. Upon receiving the execution results, the orchestrator sends them back to the end user. It then updates the UED

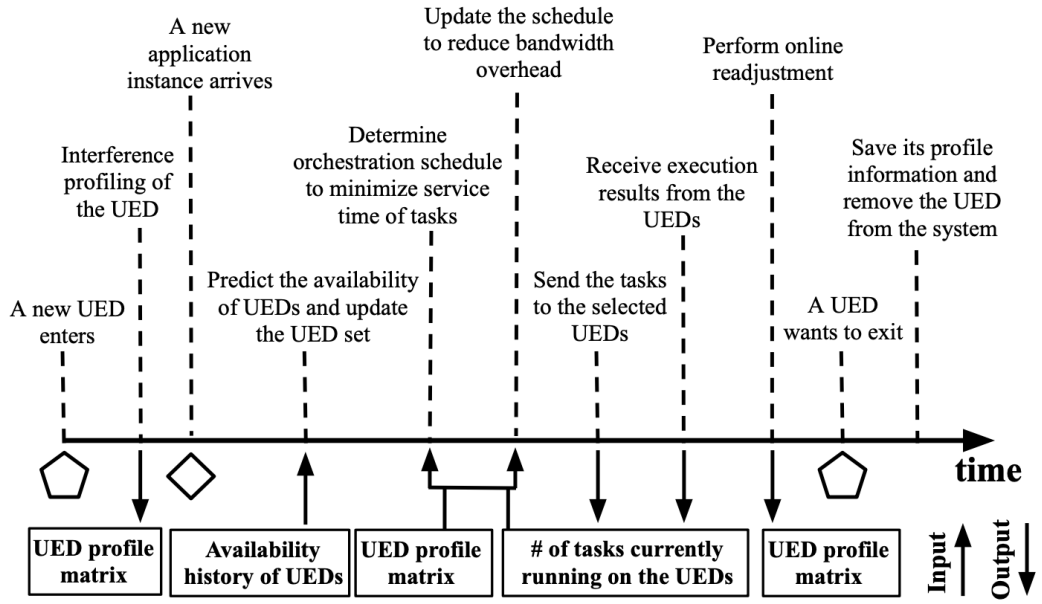


Fig. 4.1.: System Timeline

profile database based on the error between the estimated and actual service time of the tasks on the selected UEDs. The error in the estimation of the service time can occur because of inaccurate profiling of a UED or online heterogeneity such as a variation in the available capacity of a UED. Updating the UED profile based on the feedback error handles such heterogeneities. In the event that a UED wishes to exit the system, its profiling information is saved by I-BOT so that re-profiling is not required whenever the UED re-enters the system.

DESIGN

The system consists of our orchestrator running on a *managed* edge device that can offload tasks to multiple $UEDs$ connected to it, as shown in Figure 1.2. The managed edge device is controlled by an infrastructure provider and can be a wireless access point, switch, low to mid range servers installed at the cellular base stations, etc. The end users send application instances to the managed edge device acting as the orchestrator. The orchestrator serves the instances in the order in which they arrive. Our goal is to minimize the total service time of all the tasks in the application instances while reducing the bandwidth overhead. The symbols used in this thesis and their definitions are summarized in Table 5.1.

5.1 Application Structure

Each application instance consists of N tasks, some of which may require the same input data to execute. The structure of a typical application instance is shown in Figure 1.2. It is more bandwidth efficient to send the tasks that require the same input data to the same UED. In our current implementation, we use a linear chain of tasks, though this can be extended to a DAG of tasks with no conceptual novelty (but some engineering effort), as discussed in Section 7.

5.2 Pairwise Incremental Service Time Plots

We define pairwise incremental service time plots $f_{ij}(T_i, kT_j)_p$ to characterize the execution time of a new task of type T_i on UED_p, given that k tasks of type T_j are already running on the UED. This captures the heterogeneity in the interference caused by the tasks. Examples of such plots can be seen in Figures 3.1b and 5.1.

Symbol	Definition
$T = \{T_1, T_2, ..., T_N\}$	N different types of tasks for a given application instance
$UED = \{UED_1, UED_2, ..., UED_Q\}$	Q is the total number of UEDs
$f_{ij}(T_i, kT_j)_p = m_{ij} * k + c_{ij} =< m_{ij}, c_{ij} >_p$	Pairwise incremental service time plots on UED_p characterized by slope m_{ij} and y-intercept c_{ij}
$A = [< m_{ij}, c_{ij} >_p]$	Pairwise incremental service time matrix (each row corresponds to a different UED; Figure 5.2)
$Z = [z_{pi}]$	(Task count matrix) Number of tasks of type T_i currently running on UED_p
$ST_{exp}(T_i)_p$	Expected service time of a task of type T_i on UED_p
$ST_{actual}(T_i)_p$	Actual service time of a task of type T_i on UED_p
$R(t)_p$	Probability that UED_p is available continuously between the current time and t time units in the future
Hyper-parameters: (i) δ (ii) β (iii) γ	(i) δ controls the amount of readjustment performed online (ii) β controls the amount of reduction in the bandwidth overhead (iii) γ is minimum threshold for a UED availability for it to be used

$i, j \in [1 : N]$; $p \in [1 : N]$

Table 5.1.: Symbols and their definitions.

We observed that these plots are always straight lines but with varying slopes and y-intercepts due to the task interference and heterogeneity in interference patterns, as elaborated in Section 3.0.2. On a given UED, for a new task T_i, we can plot N pairwise incremental service time plots, one for interference with every other type of task (including T_i). Hence, N^2 such plots exist for every UED and we need to store only N^2 pairs of m and c values to characterize all the plots for that UED. We compute the expected service time of any new incoming task T_i on UED_p, which has $\alpha_1, \alpha_2, \cdots, \alpha_N$ tasks of each type already running using the following equation:

$$f_{i,(1,2,\cdots,N)}\big(T_i,(\alpha_1 T_1,\cdots,\alpha_i T_i,\cdots,\alpha_N T_N)\big) = f_{i1}(T_i,\alpha_1 T_1)+ \\ \cdots + f_{ii}(T_i,\alpha_i T_i)+\cdots+f_{iN}(T_i,\alpha_N T_N). \quad (5.1)$$

This assumes that the interference patterns are independent and additive. We verify this experimentally as can be seen in Figure 5.1. The figure shows that the curve obtained by adding $f_{21}(T_2, jT_1)$ and $f_{22}(T_2, kT_2)$ is very similar to $f_{2,(1,2)}(T_2,(jT_1,kT_2))$.

We define a pairwise incremental service time matrix A, each row of which contains the N^2 pairs of m and c values for a particular UED. See Figure 5.2 for the structure of matrix A. The element $< m_{ij}, c_{ij} >_p$ means that if we want to schedule a new task of type T_i while k instances of task T_j are running on a UED_p, the service time of this task T_i will be estimated as $m_{ij} * k + c_{ij}$. We also define a task count matrix Z, each row of which contains the number of tasks of all the different types currently running on a particular UED. Since the orchestrator sends the tasks and receives the execution results from the UEDs, it keeps updating the matrix Z, whenever needed.

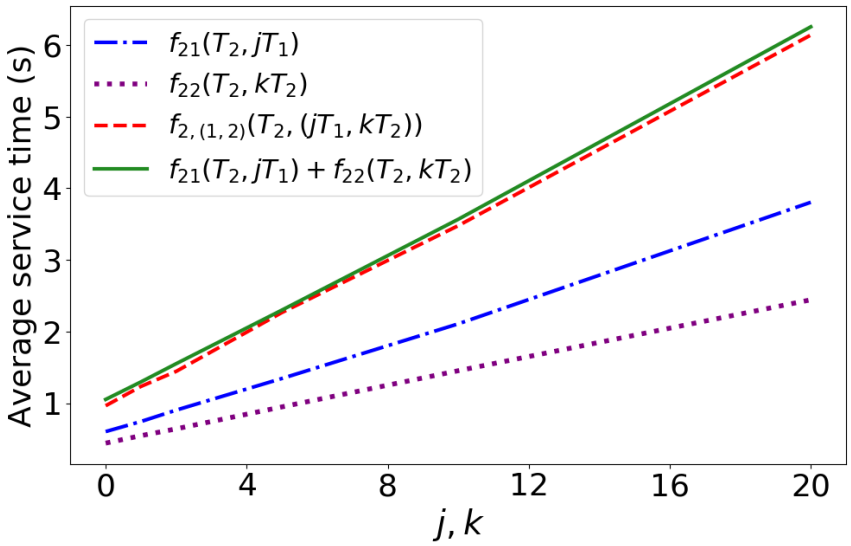


Fig. 5.1.: Experimental validation for computing the expected service time of a new incoming task using Eq. (5.1); j and k are the number of tasks of T_1 and T_2 already running on the UED respectively

$$A_{Q,N^2} = \begin{pmatrix} < m_{11}, c_{11} >_1 & \cdots & < m_{ij}, c_{ij} >_1 & \cdots & < m_{NN}, c_{NN} >_1 \\ < m_{11}, c_{11} >_2 & \cdots & < m_{ij}, c_{ij} >_2 & \cdots & < m_{NN}, c_{NN} >_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ < m_{11}, c_{11} >_p & \cdots & < m_{ij}, c_{ij} >_p & \cdots & < m_{NN}, c_{NN} >_p \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ < m_{11}, c_{11} >_Q & \cdots & < m_{ij}, c_{ij} >_Q & \cdots & < m_{NN}, c_{NN} >_Q \end{pmatrix}$$

Fig. 5.2.: Pairwise incremental service time matrix A; Q is the total number of UEDs and N is the total number of different types of tasks in each application instance

Note that, in practice, the application instances arriving at the orchestrator will not be of the same application type. The application instances can be of different types, each consisting of a different set of tasks. At the orchestrator, there will be a separate matrix A for each application type. However, for ease of exposition, we will present our algorithms as if all application instances that arrive belong to a single type of application consisting of N tasks.

5.3 Interference Profiling: Adding a New Unmanaged Edge Device

Adding a new UED to the system requires obtaining all the N^2 pairs of m and c values for the UED and adding them as a new row to matrix A (Figure 5.2). One way to obtain the N^2 pairs is to recreate all the required pairwise interference patterns by actually running tasks on the UED. Since each pairwise interference pattern is a straight line, the m and c values for that pattern can be obtained by extracting any two points on the plot. However, this method of profiling a new UED is not desirable for large N since it would require a lot of time and resources to obtain all N^2 pairs. For some UEDs, the amount of time needed to profile may be in the order of several minutes. Also, since the availability of UEDs in the unmanaged setting is sporadic, spending a lot of time in profiling a UED would be inefficient if the UED is not available for long.

To quickly profile a new UED, we use a technique similar to [31], which relies on Singular Value Decomposition (SVD) and PQ reconstruction. This technique is based

on the algorithm Netflix uses to provide movie recommendations to new users who have only rated a handful of movies. The idea is to find similarities between the new user and the existing users who have rated a lot of movies. We profile the first few UEDs by actually obtaining all the N^2 pairs. Thereafter, for every new UED, we get as many pairs as possible within a fixed time bound (1 minute in our experiments and configurable) and estimate the missing pairs using SVD and PQ-reconstruction. The time complexity of SVD and PQ-reconstruction is linear in N and, in practice, only takes a few milliseconds even for a large N (~ 30). Hence, this scheme is much quicker than obtaining all the N^2 pairs. The inaccuracies in the estimation are handled by online readjustment (Section 5.6).

5.4 UED Availability Prediction

One of the challenges in unmanaged edge computing is the sporadic availability of the UEDs (Section 3.0.2). UEDs may enter or exit the system without prior notice. If a task is scheduled on a UED which is unavailable, or which exits the system before task completion, it would be required to reschedule the task thereby increasing the task completion time. I-BOT predicts the availability of the UEDs and schedules tasks on a UED only if there is a high probability of it being available throughout the task completion. We utilize a semi-Markov Process (SMP) model, similar to [32], to predict the reliability R of a UED. This is the probability of the UED being available throughout a future time window. In an SMP model, the next transition not only depends on the current state (as would happen for a pure Markov model) but also on how long the system has stayed at this state. We observed that the availability pattern of a UED is comparable in the most recent days. Hence, using the availability history of a UED on previous days, we calculate the parameters of the SMP to evaluate $R(t)$, the probability that the UED is available continuously between the current time and t time units in the future. Tasks are scheduled on a

UED only if the probability of it being available throughout the time that it takes to complete the most demanding task in the application is greater than a threshold γ.

5.5 Orchestration Scheme

The orchestration algorithm, the largest part of I-BOT, is shown in Algorithm 1. The algorithm consists of four segments: UED availability prediction, minimum service time scheduling, reduction in the bandwidth overhead, and online readjustment. When a new application instance arrives, we first predict the probability of each UED being available throughout the execution of the application instance. The UEDs for which this probability is lower than a threshold γ are dropped out of the scheduling for the current application instance. The orchestrator maintains a count (in matrix Z) of the number of tasks of different types currently running on the available UEDs. The orchestrator uses this count and the pairwise incremental service time matrix A to predict the service time of the tasks on every available UED and create an initial mapping between the tasks and the UEDs. This mapping assigns each task to a UED on which the expected service time for the task is minimum under the current state of other tasks running on each UED. Predicting the service time of a task involves extracting the corresponding entries from the A matrix and using Eq. 5.1.

Next, the orchestrator tries to reduce the bandwidth overhead by making modifications to the initial schedule. For every group of tasks that require the same input data but are scheduled on different UEDs, the orchestrator tries to schedule them on the same UED to reduce the bandwidth overhead. A change in the assigned UED for a task is made only if the relative increase in its service time due to the change is less than a threshold β, which is the bandwidth overhead control parameter. It decides the trade-off between the bandwidth overhead and the average service time. If β is higher, I-BOT becomes more bandwidth conserving at the expense of higher service time. Finally, the tasks are sent and executed by the assigned UEDs. Upon receiving the execution result, the orchestrator computes the actual service time for

Algorithm 1: *Main_Orchestrator*

Input: A new application instance T
Initialization: UED, A and Z
Let t_{max} be the maximum time to execute the most computationally intensive task on the devices in UED
`// UED availability prediction`
for $UED_p \in UED$ **do**
  Compute $R_p(t_{max})$ using semi-Markov Process (SMP)
  if $R_p(t_{max}) \leq \gamma$ **then**
    Remove UED_p from UED
  end
end
`// Minimum service time scheduling`
for $T_i \in T$ **do**
  for $UED_p \in UED$ **do**
    $ST_{exp}(T_i)_p = GetExpectedServiceTime(i, p)$;
  end
  $ST_{exp}^{min}[i] = \min_p \left(ST_{exp}(T_i)_p\right)$;
  $UED_{sel}[i] = \operatorname{argmin}_p \left(ST_{exp}(T_i)_p\right)$;
end
`// Reduction in bandwidth overhead`
Let $K = [k_1, k_2, ...k_R]$ be a group of tasks which require the same input data
for *every* K **do**
  $ued_1 = UED_{sel}[k_1]$;
  for j = 2, ..., R **do**
    $ued_j = UED_{sel}[k_j]$;
    if $ued_j \neq ued_1$ **then**
      $ST_{min} = ST_{exp}^{min}[k_j]$;
      $ST_1 = GetExpectedServiceTime(k_j, ued_1)$;
      if $\frac{ST_1 - ST_{min}}{ST_{min}} \leq \beta$ **then**
        $UED_{sel}[k_j] = ued_1$;
      end
    end
  end
end
`// Online readjustment`
for $T_i \in T$ **do**
  $p = UED_{sel}[i]$;
  Schedule task T_i on UED_p and compute the actual service time $ST_{actual}(T_i)_p$
  $ST_{exp}(T_i)_p = ST_{exp}^{min}[i]$;
  if $\frac{\left|ST_{exp}(T_i)_p - ST_{actual}(T_i)_p\right|}{ST_{actual}(T_i)_p} > \delta$ **then**
    $PerformGradientDescent\big(i, p, ST_{exp}(T_i)_p, ST_{actual}(T_i)_p\big)$;
  end
end

each task. If the difference between estimated and actual service times for a task is greater than an error threshold (δ), then the orchestrator updates A as described (Section 5.6). For Q total number of UEDs and N tasks in each application instance, the time complexity of our orchestration scheme is $\mathcal{O}(NQ)$. Hence, our scheme can easily scale up without significant overheads.

5.6 Online Readjustment

Online readjustment of the p^{th} row of matrix A is needed when there is a large difference (greater than δ) between the expected and the actual service time of a task T_i on UED_p. This difference arises if there is an inaccuracy in the N incremental service time pairs $< m, c >$ corresponding to T_i in the p^{th} row of A. Following are the main reasons for the inaccuracy:

Imperfect information: As described in Section 5.3, most of the $< m, c >$ pairs in the row added for a UED are computed using SVD and PQ reconstruction and may not be completely accurate.

Online variation: Even if all the $< m, c >$ pairs are correctly profiled initially, the true values may change over time if the owner of the UED starts using a larger portion of the device's compute capability for his/her personal applications. This will result in a change in the pairwise incremental service time plots, thereby changing the $< m, c >$ values.

Algorithm 2: $PerformGradientDescent(i, p, ST_{exp}, ST_{actual})$

Input: $i, p, ST_{exp}, ST_{actual}$
$M = [< m_{ij} >_p]$;
$C = [< m_{ij} >_p]$; $j \in 1, 2, ..., N$
// M and C extracted from p^{th} row of A
$X = TaskCountUED_p = Z[p, :] = [Z_{pj}]; j \in 1, 2, ..., N$
// p^{th} row of Z
$M^{new}, C^{new} = GradientDescent(M, C, X, ST_{actual}, ST_{exp})$;
Update A with M^{new} and C^{new};

Therefore, we need to make online adjustments to the matrix A. For this, we use gradient descent as described in Algorithm 2. For a task T_i scheduled on UED_p, if the difference between the expected and the actual service time exceeds δ, gradient descent is performed to minimize the error between the expected and actual service time and obtain the new values of $< m, c >$ for task T_i on UED_p.

5.7 Unmanaged Edge Device Exit

A UED may leave the system if there is a sudden unexpected crash or if the owner of the UED exits the system. Not much can be done in the case of an unexpected crash. However, in the other case, we perform an additional step for a graceful exit which can save us from re-profiling the UED if it re-joins the system in the future. When the owner of the UED_p wants to exit the system, the information corresponding to the UED stored in the p^{th} row of the A matrix is saved by the system. The row can then be removed from A in the orchestrator. Later, if the UED rejoins the system, its profiling information can be loaded to the orchestrator during the entry phase which significantly reduces the time needed to profile the UED on the system.

8. RELATED WORK

In this section we contrast our work with the other efforts in the field of task scheduling in heterogeneous edge computing systems.

Low latency edge scheduling: Petrel [13] and LAVEA [14] propose orchestration schemes aimed at minimizing the service time in a multi-edge collaborative environment. We have shown that I-BOT outperforms these schemes in terms of the service time and bandwidth overhead in a heterogeneous unmanaged edge computing setting. MSGA [15] jointly studies the task and network flow scheduling and uses a multi-stage greedy algorithm to minimize the completion time of the application. In [17], a gateway-based edge computing service model has been proposed to reduce the latency of data transmission and the network bandwidth. Low latency task scheduling schemes for edge have also been proposed in [38–40]. However, all of these works are in the context of managed edge and do not consider the unique challenges introduced by unmanaged edge, such as the lack of monitoring information, heterogeneity, and unexpected entry-exits. One exception to this is CoGTA [41], which considers scheduling of delay-sensitive social sensing tasks on a heterogeneous unmanaged edge. However, its main focus is on devices that are not trusted and therefore it formulates a game-theoretic technique to perform the task allocation. Its performance in a benign setting like ours is likely to be sub-optimal.

Availability and Interference based edge scheduling: There have been a few efforts that take into account the availability and interference while devising strategies for task scheduling on the edge. An overhead-optimizing task scheduling strategy has been proposed in [18] for ad-hoc based edge computing nodes formed by a group of mobile devices. [19] proposes a score based edge service scheduling algorithm that evaluates network, compute, and reliability capabilities of edge nodes. However, these works rely on sharing monitoring information which can be a huge overhead in highly

dynamic environments. Also, the time and energy consumption models are theoretical and have not been tested on real systems. INDICES [42] proposes a performance-aware scheme for migrating services from cloud to edge while taking into account the interference caused by co-located applications. However, this is geared towards service migration and not task scheduling. Also, it does not consider the impact of online variations in the availability and compute capabilities of edge devices.

Energy efficient edge scheduling: A lot of existing works [43–46] utilize dynamic voltage-frequency scaling (DVFS), which is an attractive method for reducing energy consumption in heterogeneous computing systems. ESTS [47] deals with the problem of scheduling a group of tasks, optimizing both the schedule length and energy consumption. They formulate the problem as a joint linear programming problem and propose a heuristic algorithm to solve it. In [48], a computational offloading framework has been proposed which minimizes the total energy consumption and execution latency by coupling task allocation decisions and frequency scaling. The paper [16] also performs joint optimization of energy and latency through a rigorously formulated and solved mixed integer nonlinear problem (MINLP) for computation offloading and resource allocation. However, the execution models used in these works do not consider the impact of online heterogeneties in the computation capacity or the effect of interference.

Volunteer or opportunistic computing: In a completely different context, under the moniker "volunteer computing", a slew of works designed solutions to utilize under-utilized compute nodes (such as, on a university campus) or mobile devices to run large-scale parallel applications. An example of the former is HTCondor [49] and an example of the latter is Femtocloud [50]. Our design borrows some features from Femtocloud (identifying devices with spare capacity and some stability); however, Femtocloud did not have to deal with the majority of the challenges that we solve here (great heterogeneity from a compute, network, and application standpoint, unknown tasks, runtime variations due to interference).

REFERENCES

[1] eukhost, "New statistics: Show the advance of cloud computing," https://www.eukhost.com/blog/webhosting/new-statistics-show-the-advance-of-cloud-computing/, 2020, accessed: 2020-06-28.

[2] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan 2017.

[3] C. Sonmez, A. Ozgovde, and C. Ersoy, "Edgecloudsim: An environment for performance evaluation of edge computing systems," in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, May 2017, pp. 39–44.

[4] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct. 2009. [Online]. Available: http://dx.doi.org/10.1109/MPRV.2009.82

[5] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: bringing the cloud to the mobile user," in *3rd ACM Workshop on Mobile Cloud Computing and Services, Proceedings*. Ghent University, Department of Information technology, 2012, pp. 29–35.

[6] M. Aazam and E. Huh, "Fog computing micro datacenter based dynamic resource estimation and pricing model for iot," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, 2015, pp. 687–694.

[7] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, p. 68–73, Dec. 2009. [Online]. Available: https://doi-org.ezproxy.lib.purdue.edu/10.1145/1496091.1496103

[8] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 13–16. [Online]. Available: https://doi-org.ezproxy.lib.purdue.edu/10.1145/2342509.2342513

[9] M. Aazam and E. Huh, "Fog computing and smart gateway based communication for cloud of things," in *2014 International Conference on Future Internet of Things and Cloud*, 2014, pp. 464–470.

[10] "Amazon: Lambda@edge," https://aws.amazon.com/lambda/edge/, 2020, accessed: 2020-06-28.

[11] "Cisco: Establishing the edge," https://www.cisco.com/c/en/us/solutions/service-provider/edge-computing/establishing-the-edge.html, 2020, accessed: 2020-06-28.

[12] "Google: Edge network," https://peering.google.com/#/, 2020, accessed: 2020-06-28.

[13] L. Lin, P. Li, J. Xiong, and M. Lin, "Distributed and application-aware task scheduling in edge-clouds," in *2018 14th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN)*, 2018, pp. 165–170.

[14] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3132211.3134459

[15] Y. Sahni, J. Cao, and L. Yang, "Data-aware task allocation for achieving low latency in collaborative edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3512–3524, 2019.

[16] J. Zhang, X. Hu, Z. Ning, E. C.-H. Ngai, L. Zhou, J. Wei, J. Cheng, and B. Hu, "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2633–2645, 2017.

[17] C.-W. Tseng, F.-H. Tseng, Y.-T. Yang, C.-C. Liu, and L.-D. Chou, "Task scheduling for edge computing with agile vnfs on-demand service model toward 5g and beyond," *Wireless Communications and Mobile Computing*, vol. 2018, p. 7802797, Jul 2018. [Online]. Available: https://doi.org/10.1155/2018/7802797

[18] L. Tianze, W. Muqing, Z. Min, and L. Wenxing, "An overhead-optimizing task scheduling strategy for ad-hoc based mobile edge computing," *IEEE Access*, vol. 5, pp. 5609–5622, 2017.

[19] A. Aral, I. Brandic, R. B. Uriarte, R. De Nicola, and V. Scoca, "Addressing application latency requirements through edge scheduling," *Journal of Grid Computing*, vol. 17, no. 4, pp. 677–698, Dec 2019. [Online]. Available: https://doi.org/10.1007/s10723-019-09493-z

[20] A. J. Page and T. J. Naughton, "Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing," in *19th IEEE International Parallel and Distributed Processing Symposium*, 2005, pp. 8 pp.–.

[21] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, "Zenith: Utility-aware resource allocation for edge computing," in *2017 IEEE International Conference on Edge Computing (EDGE)*, 2017, pp. 47–54.

[22] R. D. Schlichting and F. B. Schneider, "Fail-stop processors: An approach to designing fault-tolerant computing systems," *ACM Trans. Comput. Syst.*, vol. 1, no. 3, p. 222–238, Aug. 1983. [Online]. Available: https://doi-org.ezproxy.lib.purdue.edu/10.1145/357369.357371

[23] F. B. Schneider and Lidong Zhou, "Implementing trustworthy services using replicated state machines," *IEEE Security & Privacy*, vol. 3, no. 5, pp. 34–43, 2005.

[24] R. Xu, J. Koo, R. Kumar, P. Bai, S. Mitra, S. Misailovic, and S. Bagchi, "Videochef: Efficient approximation for streaming video processing pipelines," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, 2018, pp. 43–56. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/xu-ran

[25] R. N. Calheiros, R. Ranjan, C. A. F. D. Rose, and R. Buyya, "Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services," *CoRR*, vol. abs/0903.2525, 2009.

[26] M. T. Diallo, F. Fieau, and J. Hennequin, "Impacts of video quality of experience on user engagement in a live event," in *2014 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, July 2014, pp. 1–7.

[27] "Aws iot greengrass," https://aws.amazon.com/greengrass/.

[28] "Aws iot greengrass usage," https://discovery.hgdata.com/product/aws-iot-greengrass.

[29] P. Wood, H. Zhang, M. Siddiqui, and S. Bagchi, "Dependability in edge computing," *CoRR*, vol. abs/1710.11222, 2017. [Online]. Available: http://arxiv.org/abs/1710.11222

[30] L. Fridman, D. E. Brown, M. Glazer, W. Angell, S. Dodd, B. Jenik, J. Terwilliger, A. Patsekin, J. Kindelsberger, L. Ding, S. Seaman, A. Mehler, A. Sipperley, A. Pettinato, B. D. Seppelt, L. Angell, B. Mehler, and B. Reimer, "Mit advanced vehicle technology study: Large-scale naturalistic driving study of driver behavior and interaction with automation," *IEEE Access*, vol. 7, pp. 102 021–102 038, 2019.

[31] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," in *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 77–88. [Online]. Available: https://doi-org.ezproxy.lib.purdue.edu/10.1145/2451116.2451125

[32] Xiaojuan Ren, Seyong Lee, R. Eigenmann, and S. Bagchi, "Resource availability prediction in fine-grained cycle sharing systems," in *2006 15th IEEE International Conference on High Performance Distributed Computing*, 2006, pp. 93–104.

[33] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, 2001.

[34] R. Sakellariou and H. Zhao, "A hybrid heuristic for dag scheduling on heterogeneous systems," in *18th International Parallel and Distributed Processing Symposium.* IEEE, 2004, p. 111.

[35] G. Bosilca, A. Bouteiller, A. Danalis, T. Herault, P. Lemarinier, and J. Dongarra, "Dague: A generic distributed dag engine for high performance computing," *Parallel Computing*, vol. 38, no. 1-2, pp. 37–51, 2012.

[36] S. Khare, H. Sun, J. Gascon-Samson, K. Zhang, A. Gokhale, Y. Barve, A. Bhattacharjee, and X. Koutsoukos, "Linearize, predict and place: minimizing the makespan for edge-based stream processing of directed acyclic graphs," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 2019, pp. 1–14.

[37] B. Falsafi and T. F. Wenisch, "A primer on hardware prefetching," *Synthesis Lectures on Computer Architecture*, vol. 9, no. 1, pp. 1–67, 2014.

[38] S. Wang, Y. Li, S. Pang, Q. Lu, S. Wang, and J. Zhao, "A task scheduling strategy in edge-cloud collaborative scenario based on deadline," *Scientific Programming*, vol. 2020, p. 3967847, Mar 2020. [Online]. Available: https://doi.org/10.1155/2020/3967847

[39] J. Han and D. Wang, "Edge scheduling algorithms in parallel and distributed systems," in *2006 International Conference on Parallel Processing (ICPP'06)*, 2006, pp. 147–154.

[40] T. He, H. Khamfroush, S. Wang, T. La Porta, and S. Stein, "It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 365–375.

[41] D. Zhang, Y. Ma, C. Zheng, Y. Zhang, X. S. Hu, and D. Wang, "Cooperative-competitive task allocation in edge computing for delay-sensitive social sensing," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 243–259.

[42] S. Shekhar, A. D. Chhokra, A. Bhattacharjee, G. Aupy, and A. Gokhale, "Indices: Exploiting edge resources for performance-aware cloud-hosted services," in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, 2017, pp. 75–80.

[43] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto, "Survey of energy-cognizant scheduling techniques," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 7, pp. 1447–1464, 2013.

[44] D. Li and J. Wu, "Energy-aware scheduling for frame-based tasks on heterogeneous multiprocessor platforms," in *2012 41st International Conference on Parallel Processing*, 2012, pp. 430–439.

[45] N. B. Rizvandi, J. Taheri, and A. Y. Zomaya, "Some observations on optimal frequency selection in dvfs-based energy consumption minimization," *Journal of Parallel and Distributed Computing*, vol. 71, no. 8, pp. 1154 – 1164, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731511000165

[46] H. Kimura, M. Sato, Y. Hotta, T. Boku, and D. Takahashi, "Emprical study on reducing energy of parallel programs using slack reclamation by dvfs in a power-scalable high performance cluster," in *2006 IEEE International Conference on Cluster Computing*, 2006, pp. 1–10.

[47] K. Li, X. Tang, and K. Li, "Energy-efficient stochastic task scheduling on heterogeneous computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 11, pp. 2867–2876, 2014.

[48] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 3571–3584, 2017.

[49] D. H. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne, "A worldwide flock of condors: Load sharing among workstation clusters," *Future Generation Computer Systems*, vol. 12, no. 1, pp. 53–65, 1996.

[50] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *2015 IEEE 8th international conference on cloud computing*. IEEE, 2015, pp. 9–16.

A. APPENDIX

A.1 Theoretical Analysis

We present the theoretical analysis of our solution under the following simplifying assumptions. First, we assume that the UEDs are homogeneous and a task of type k has exponentially distributed processing rate μ_k for $k = [1 : N]$, where $\frac{1}{Q}\sum_{k=1}^{N} \lambda/\mu_k < 1$. We further assume that tasks of type 1 to N are dispatched to the chosen UED's queue in order. Queue state for UED_q, $q = [1 : Q]$, is then defined by

$$\phi_n = \{(0)\} \cup \{(t_1, t_2, \ldots, t_n) | n \geq 1\},$$

where t_i is the type of the ith task in the (type independent) FIFO order (t_1 is the type of a task being served) and (0) represents the empty system. Under these assumptions, queue length determines the expected service time. Specifically, the expected service time for all tasks in UED is a monotonically increasing function of the UED queue length.

The evolution of the system over ϕ_n is an irreducible Markov chain. Using the Lyapunov theorem, it can be verified that the Markov chain is positive recurrent, and thereby has a unique stationary distribution. Let $\pi(t_1, t_2, \ldots, t_i)$ denote the stationary distribution of UED_q, i.e., the probability that the queue state is $(t_1, t_2, \ldots, t_i)$ at UED_q. Here, the index q is ignored because the stationary distributions are identical across UEDs. We have $\sum_i i\pi(t_1, t_2, \ldots, t_i) = \sum_i i\pi(\phi_i) = \sum_i i\pi_i < c$, where a constant $c > 0$.

Consider the queue evolution of one UED in the system. At steady state, each queue forms an independent Markov chain, as described in the following lemma:

Lemma 2 *Under our proposed solution, the transition rates $q_{i,j}(\boldsymbol{\pi})$ given distribution $\boldsymbol{\pi}$ for $j \neq i$ is given by*

$$q_{i,j}(\boldsymbol{\pi}) = \begin{cases} \mu_{\lceil \frac{i}{N} \rceil N-i+1} & \text{if } j = i-1, \\ \frac{1-(Q-1)\sum_{l=0}^{i-1}\pi_l}{1+(Q-1)\sum_{l=0}^{i}\pi_l} & \text{if } j = i+N,\ i < \tau_{\boldsymbol{\pi}}, \\ 0, & \text{otherwise,} \end{cases}$$

where $\tau_{\boldsymbol{\pi}} = \min\{j : \sum_{l=0}^{j-1}\pi_l \geq \frac{1}{Q-1}\}$ and π_l denotes the stationary distribution of UED queue, i.e., the probability that the queue size is l at a UED.

Proof The transition rates will be determined by our solution used to dispatch tasks to UEDs. We will derive the transition rates for our strategy. First, the down-crossing transition rate from state i to state $i-1$ is

$$\begin{aligned} q_{i,i-1} &= \mu_{t_1} \\ &= \mu_{\lceil \frac{i}{N} \rceil N-i+1} \end{aligned}$$

because the processing time of a task of type t_1 is exponentially distributed with mean μ_{t_1} and the type of a task being served is uniquely determined by queue length i as $\lceil \frac{i}{N} \rceil N - i + 1$ due to our dispatch strategy.

Second, the up-crossing transition rate from state i to state j for $j > i$ is

$$q_{i,j} = \lambda \sum_{\boldsymbol{\eta}} \mathsf{P}(\boldsymbol{\eta}) \cdot \mathsf{P}(j|\boldsymbol{\eta}, i),$$

where $\boldsymbol{\eta}$ is a $(Q-1)$ vector that denotes the queue lengths of the other $Q-1$ UEDs; thus,

$$\mathsf{P}(\boldsymbol{\eta}) = \prod_{q=1}^{Q-1} \pi_{\eta_q}$$

and $\mathsf{P}(j|\boldsymbol{\eta}, i)$ is the probability that a UED's queue length becomes j when the UED is in state i and the states of the other $Q-1$ UEDs are $\boldsymbol{\eta}$.

Assume ties are broken uniformly at random. If $\sum_{q=1}^{Q-1} \mathbf{1}_{\eta_q \leq i-1} \geq 1$, then

$$\mathsf{P}(j|\boldsymbol{\eta}, i) = \begin{cases} 1 & \text{if } j = i, \\ 0 & \text{if } j \neq i \end{cases}$$

because the tasks will be dispatched to UEDs, original queue lengths of which are smaller than i. On the other hand, if $\sum_{q=1}^{Q-1} \mathbf{1}_{\eta_q \leq i-1} < 1$, then the UED with queue length i will receive N tasks, and $\mathsf{P}(j|\boldsymbol{\eta}, i) = 1$ for $j = i + N$.

WLOS, we assume UED_Q has queue size i. Given any $j \geq 0$, we define $T_j = \sum_{q=1}^{Q-1} \mathbf{1}_{\eta_q = j}$, which is the number of UEDs with queue length j excluding UED_Q. T_j is then the sum of $Q-1$ i.i.d. Bernoulli r.v.'s with mean π_j; thus, $\mathsf{E}\, T_j = (Q-1)\pi_j$. Now, the probability that UED_Q receives N tasks is given by

$$\mathsf{E}\left(\frac{1 - \sum_{j=0}^{i-1} T_j}{1 + \sum_{j=0}^{i} T_j}\right)^+,$$

which, at steady state, can be approximated by

$$\left(\frac{1 - (Q-1)\sum_{j=0}^{i-1} \pi_j}{1 + (Q-1)\sum_{j=0}^{i} \pi_j}\right)^+$$

because T_j converges to $(Q-1)\pi_j$ in distribution and the term inside the expectation is bounded and continuous in terms of T_j. This concludes the proof. ■

According to Lemma 2, the queue length dynamic of a single UED can be represented by the Markov chain in Figure A.1. Intuitively, $\tau_{\boldsymbol{\pi}}$ indicates the queue length so that the probability that a UED with queue size $i(\geq \tau_{\boldsymbol{\pi}})$ receives N tasks is 0. Based on Lemma 2, we can calculate the stationary distribution of the queue length of a single UED numerically by finding $\hat{\pi}$ that satisfies the global balance equation.

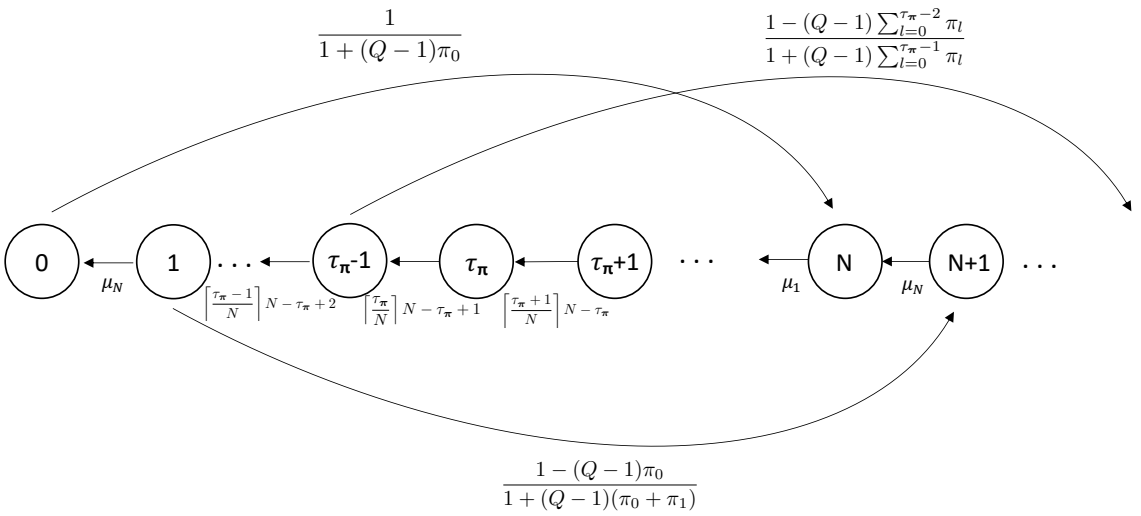


Fig. A.1.: The Markov chain representing the system

Lemma 3 *The expected service time* $T_Q(\lambda, \mu_1, \cdots, \mu_N)$ *of an application instance that is dispatched to* Q *UEDs is given by*

$$\sum_{r=0}^{N-1}\left[\sum_{i=1}^{\infty}\left(\lfloor\frac{i-1}{N}\rfloor\sum_{l=1}^{N}\frac{1}{\mu_l}+\mathbf{1}_{\lceil\frac{i}{N}\rceil N-i+1-r\geq 1}\sum_{m=\lceil\frac{i}{N}\rceil N-i+1-r}^{N-r}\frac{1}{\mu_m}+\right.\right.$$
$$\left.\left.\mathbf{1}_{\lceil\frac{i}{N}\rceil N-i+1-r<1}\sum_{m=N-r}^{\lceil\frac{i}{N}\rceil N-i+1-r+N}\frac{1}{\mu_m}\right)\cdot\left\{\left(\sum_{j=i-1}^{\infty}\pi_j\right)^Q-\left(\sum_{j=i}^{\infty}\pi_j\right)^Q\right\}\right].$$

Proof Task of type N becomes the ith task in the queue with probability $\left(\sum_{j=i-1}^{\infty}\pi_j(t)\right)^Q - \left(\sum_{j=i}^{\infty}\pi_j(t)\right)^Q$. Thus, the expected time a task spends in the system under our dispatch solution is

$$\sum_{i=1}^{\infty}\left(\lfloor\frac{i-1}{N}\rfloor\sum_{l=1}^{N}\frac{1}{\mu_l}+\sum_{m=\lceil\frac{i}{N}\rceil N-i+1}^{N}\frac{1}{\mu_m}\right)\cdot\left\{\left(\sum_{j=i-1}^{\infty}\pi_j\right)^Q-\left(\sum_{j=i}^{\infty}\pi_j\right)^Q\right\}.$$

For other type $N-r$ of a task, the cyclic structure in queue should be taken into account and there by changes an expression for the summation $\sum_{m=\lceil\frac{i}{N}\rceil N-i+1}^{N}\frac{1}{\mu_m}$, which leads to the desired result. ■