On Decentralized Route Planning Using the Road Side Units as Computing Resources

Jose Paolo Talusan[1], Michael Wilbur[2], Abhishek Dubey[2], and Keiichi Yasumoto[1]

[1]Nara Institute of Science and Technology, Ikoma, Nara 630-0192, Japan
{*talusan.jose_paolo.tg3, yasumoto*}*@is.naist.jp*
[2]Vanderbilt University, Nashville, TN 37240, USA
{*michael.p.wilbur, abhishek.dubey*}*@vanderbilt.edu*

***Abstract*—Residents in cities typically use third-party platforms such as Google Maps for route planning services. While providing near real-time processing, these state of the art centralized deployments are limited to multiprocessing environments in data centers. This raises privacy concerns, increases risk for critical data and causes vulnerability to network failure. In this paper, we propose to use decentralized road side units (RSU) (owned by the city) to perform route planning. We divide the city road network into grids, each assigned an RSU where traffic data is kept locally, increasing security and resiliency such that the system can perform even if some RSUs fail. Route generation is done in two steps. First, an optimal grid sequence is generated, prioritizing shortest path calculation accuracy but not RSU load. Second, we assign route planning tasks to the grids in the sequence. Keeping in mind RSU load and constraints, tasks can be allocated and executed in any non-optimal grid but with lower accuracy. We evaluate this system using Metropolitan Nashville road traffic data. We divided the area into 500 grids, configuring load and neighborhood sizes to meet delay constraints while maximizing model accuracy. The results show that there is a 30% decrease in processing time with a decrease in model accuracy of 99% to 92.3%, by simply increasing the search area to the optimal grid's immediate neighborhood.**

I. Introduction

Half of the world's population reside in urban areas. By 2020 that number is expected to increase to 70% [1]. This rise of population also brings with it a rise in housing density and traffic. These strain city services so much so that cities are looking for more and more methods to meet growing demands. One such method is to use data-intensive applications to maximize the utility of the limited resources. Companies such as Google offer a variety of platforms that leverage data and offer cities and its citizens access to services. One service is route planning which is used to avoid the increasing traffic congestion. Cloud environments process millions of route queries per day, guiding vehicles to and from their desired destination.

While these are enough for user routing, future use cases such as autonomous vehicles demand less latency. Such constraints expose the limitations of the centralized cloud-based route planning models. There is also an increasing concern regarding data privacy issues since cities typically leverage third-party services from private cloud-based companies to providing services such as emergency dispatch services. These private companies often rely on centralized services in remote data centers which are more exposed to data leakage attacks and more susceptible to disruption and communication failure. Cities risk service disruption during disaster scenarios, which is when such emergency services are needed most.

In order to solve these issues, cities are already investing in deploying RSU networks in part for the preparation of autonomous and connected vehicles. These devices which are low powered Raspberry Pi-like devices placed all along city roads and highways. Since each RSU is capable of limited amount of processing and storage, when multiple RSUs are connected in a sub-network, they form a fog computer [2] and can be used for route planning services. This sub-network is a private, reliable and pervasive network that cities and its citizens can use.

While route planning [3] is a well-studied topic, state-of-the-art route planning algorithms [4], [5], [6] are developed typically as centralized approaches for centralized architectures such as cloud environments and data-servers. In these scenarios, data is shared and parallelized, allowing for typical search algorithms to access to a shared memory and direct communication between multiple processors. These algorithms are not suited for a distributed setting.

The major challenge is in designing route planning algorithms that work well in a distributed setting such as with RSUs. The idea behind our approach is to divide the city into grids which we then assign RSUs to. Each RSU has its own local data for the grid it covers as well as models for route planning. The goal and contributions of this paper are:

1) We describe a route planning algorithm designed specifically for RSUs at the network edge.
2) Our approach shows how various tasks related to this algorithm can be optimally scheduled on the cluster of road side units.
3) Evaluation shows that task allocation over neighbor grids provide a trade-off between processing delay and accuracy. The greater the search area, the less the processing delay but also the accuracy. When compared to central approaches, ours is able to respond with less latency.

In this paper we show that routing algorithms designed specifically for fog computing such as for the RSUs at the edge, require more attention to task allocation given the processing and memory constraints of each device. Due to

the distributed nature of the system, data is not stored in a central location so there will be trade-offs between processing delay and accuracy. Our approach has significant speed gain compared to centralized approaches and offer system reliability and data security.

Outline: Fundamental notation and related work is presented in Section II. The urban middleware platform is discussed in Section III which includes the system architecture and deployment. Section IV outlines the task allocation algorithm for decentralized networks. A simulated case study is detailed in Section V and conclusions with future work are provided in Section VI.

II. Related Works

A. IoT Middleware and Task Assignment Problem

Currently IoT computation is typically offloaded to a centralized cloud for processing [7]. Cloud computing offers near unlimited processing and storage capacity for complex smart city applications, however latency can become a bottleneck for such applications. In this case, edge and fog networks offer the potential to move processing and computation to the network edge, thus reducing latency [8], [2]. These paradigms look to assign delay sensitive tasks to resources closer to the end user.

The challenges associated with edge and fog computing are primarily associated with usability, coordination and task assignment. Task scheduling in fog computing with the goal of optimizing resources to minimize tasks completion time [9] and efficiently utilizing resources to improve the performance of IoT services in terms of response time, energy, and cost reduction [10] have been studied. Therefore much research has been done on urban middleware designed to coordinate large systems of heterogeneous edge or fog networks [11], [12] and [13].

A primary goal of urban middleware is to formalize how best to assign computation tasks to available resources. We refer to this problem as the task allocation problem [14]. This problem has been extensively studied in the cloud [15], [16], [17]. The main purpose of such research is to adaptively provide the processing resources while meeting the deadline for all jobs while taking into account running costs. The task allocation problem in cloud computing therefore typically does not take into account data transfer delay, as typically the networking between virtual machines in a cloud environment is negligible. A key component of urban middleware is resource discovery, which is the method by which edge and fog networks are identified [18], [19].

Therefore, recent research in the provisioning of resources in edge and fog networks has become increasingly important. Skarlat et al. [20], [21] proposed a platform centered around the idea of fog colonies (sets of fog nodes) with a centralized cloud for additional resources when needed. Xu et al. [22] proposed a platform for location-based and latency sensitive applications which use micro data centers on the network edge or large centralized cloud for processing. This research includes a cloud component for additional processing and storage. Research on in-situ edge IoT devices looks at task assignment without relying on cloud resources [23] [24].

TABLE I: List of symbols

Symbol	Description
G_r	Grids making up the target area
RSU_i	Road Side Unit i
R	Road Side Unit assigned to a grid
N	Network graph, $N = (V, E)$
V	Road intersections
E	Set of roads
d_i	Local speed data generated by sensors in a grid
Q_i	Set of queries sent within in the same time period i
Query	
s	User source location
d	User destination location
τ_s	User desired travel time
Route Planning	
SG_q	Sequence of grids (s to d) generated per query q
k_q	Dynamic modified based on q
T_q	Sequence of tasks $t_{q,i}$ for each SG_q
$t_{q,i}$	Route planning tasks assigned to each grid in SG
Task Allocation	
T	Set of all tasks T_q for all queries Q_i
$x_{t,r}$	Assignment variable, 1 if task is assigned, 0 otherwise
$ST(t)$, $ET(t)$	Task execution start and end times
D_{th}	Delay threshold for query responses
$IT(q)$	Time when q was issued
$ct(t, r)$	Computation time of task t executed at RSU r
MA	Model accuracy of the route planning model
$MD(g, g')$	Manhattan Distance between the two different grids
$MaxDist$	Maximum distance a route for q travels from s to d

B. Centralized and Decentralized Routing

Dijkstra [3], Bellman [25] and Ford [26] proposed some of the first routing planning algorithms. Routing algorithms such as A* [27] use heuristics to guide the shortest path search while contraction hierarchies [4] simplify the graph for faster search.

Current state of the art route planning is typically deployed in centralized cloud systems [4], [5], [6]. In this architecture the routing algorithms are deployed in a central location from which it serves user queries. Within this context QoS improvements (e.g., in terms of query response time) have been made by parallelizing shortest path algorithms [28], [29], [30]. These parallelized algorithms split processing over multiple nodes. These approaches provide high scalability, optimal for cloud based services. However, these models assume a shared memory and do not take into account network latency between nodes, and therefore are not easily adaptable to edge or fog centric architectures.

III. System Model

This section describes assumptions on the target regional area, decentralized route planning service, and the tasks generated by user queries. Table I summarizes the symbols used throughout the paper.

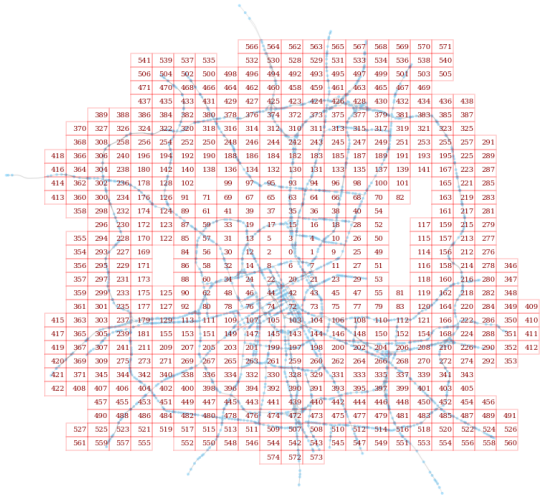

Fig. 1: Target spatial area is divided into grids

A. Spatial Region

The target area is split into equidistant *grids*, as shown in Fig. 1, which we denote as $G_r = \{g_1, g_2, \cdots, g_m\}$. *RSUs* are then deployed on these grids, the typical distribution is one RSU per one grid. Each RSU is connected to a *regional area network*[1].

Each RSU is assumed to have storage and computational resources to store sensor data as well as execute tasks from a task queue. The distributed network is denoted as a resource graph with a set of vertices $R = \{r_1, r_2, \cdots, r_m\}$. Each vertex represents an RSU over the subarea, while each edge is an undirected link between any two RSUs. The physical area map is represented by a network graph $N = (V, E)$ where V are road intersections and E are road segments.

RSUs are assumed to be both resource and memory constrained (e.g., Raspberry Pi-like computation power). As such, each RSU can only accommodate a finite amount of simultaneous tasks and hold a finite amount of sensor data and machine learning models.

B. Decentralized Routing Service

This service allows users to access to time-dependent and privacy-preserved routing services for smart transportation, even without the Internet. This does not limit the network only to routing services. Other decentralized services that utilize geo-spatial data can also take advantage of fog computing using RSU network, however for this paper we focus on a *decentralized routing service*.

Each RSU $r_i \in R$ receives data d_i from a set of sensors (e.g., TMC) that gather speed data across roads within a designated subarea (grid) g_i. To add an additional layer of resiliency in case of RSU failures, speed data is shared to neighbor RSUs. We assume that data is compressed and distributed during hours where queries are minimal. We also assume that data shared to neighbors will not be as accurate as data in the original RSU. This loss in accuracy relates to the quality of the computation when tasks are executed in a grid other than the most optimal one. If some RSUs fail or cannot handle queries, the system can still perform their route planning tasks by transferring tasks to its neighbors.

[1]We assume that RSUs are connected with wired links.

C. User and Query Tasks

Each RSU $r \in R$ is able to receive some query q with parameters (id, s, d, τ_s). id is used to differentiate the queries while s and d are the route's desired start and destination points respectively, and τ_s is the user's desired departure time. Queries can be received asynchronously by multiple RSUs at some time window i, we denote these sets of queries as $Q_i = \{q_{i,1}, q_{i,2}, \cdots, q_{i,n_i}\}$, where i is the time window the particular set of queries were received.

For each received query, we assume that a corresponding sequence of grids where optimal travel path is likely included will be generated[2]. We call this $SG_q = \langle g_{q,1}, g_{q,2}, \cdots, g_{q,k_q} \rangle$ where q is the query while k_q is a modifier that is dynamically generated and is based on the query as well. This sequence of grids corresponds to a grid level view of the route the application serves the user.

For every SG_q, a sequence of tasks $T_q = \langle t_{q,1}, t_{q,2}, \cdots, t_{q,k_q} \rangle$ where $t_{q,i}$ is the route planning task in grid $g_{q,i}$ is also generated. These tasks include getting the travel time from one vertex $v \in V$ via a road segment $e \in E$ as well as creating the path through these using path search algorithms such as Dijkstra's.

IV. Distributed Task Allocation

This section describes the assumptions on the decentralized edge networks including the RSUs, Manhattan Distance and sequence grid generation. Also we design and define the distributed task allocation problem which breaks down queries and assigns them as tasks to various nodes within the edge network.

A. Definition of the problem

Given the resource graph with vertices R, a set of queries Q_i that generates task graph (sequence) T_q, the problem is defined as identifying the most efficient and optimal task assignment to various RSUs of R. For all tasks of T over R, optimal assignments should satisfy the query response time (delay) constraints and maximize the computation accuracy for all queries.

1) Definitions of task assignment and delay: We set Eq. 1 as the first constraint. We define T as the set of all tasks T_q of all queries Q_i.

$$T \triangleq \bigcup_{q \in Q_i} T_q \tag{1}$$

For every pair of task $t \in T$ and RSU $r \in R$, we define a variable $x_{t,r}$ which becomes 1 if t is assigned to r and 0 otherwise.

[2]We assume that each RSU has a model to compute the next grid toward the destination point with inputs of the current grid, the source and destination points and start time. By repeating this next grid computation, we can get the sequence of grids from the source to the destination point.

Given this, we assume that every task from T is assigned to one RSU, thus the following condition must hold.

$$\forall t \in T, \sum_{r \in R} x_{t,r} = 1 \tag{2}$$

In each task graph T_q, tasks must be sequentially executed. Hence the following equation must hold. Here, $ST(t)$ and $ET(t)$ represent task execution start and end times, respectively.

$$\forall T_q (q \in Q_i) \forall i (1 \leq i \leq k_q - 1)\, ET(t_{q,i}) < ST(t_{q,i+1}) \tag{3}$$

Upon assignment of all tasks in T, we define that the overall service delay for queries Q_i, should not exceed some delay threshold D_{th}. Here, $IT(q)$ is the time when the query is issued.

$$\forall q \in Q_i, ET(t_{k_q}) - IT(q) \leq D_{th} \tag{4}$$

Each RSU r executing task $t_{q,i}$ will also execute other tasks assigned to it, so the time until finishing execution of $t_{q,i}$ will be the sum of execution times of these tasks for the worst case. Then, task execution start and end times of each task $t_{q,i}$ can be defined as follows.

$$\begin{aligned} ST(t_{q,i}) \overset{def}{=} IT(q) + \sum_{j=1}^{i-1} \sum_{r \in R} \sum_{t' \in T} ct(t', r) \cdot x_{t',r} \cdot x_{t_{q,j},r} \\ + \sum_{j=1}^{i-1} \sum_{r \in R} \sum_{r' \in R} delay(r, r') \cdot x_{t_{q,j},r} \cdot x_{t_{q,j+1},r'} \end{aligned} \tag{5}$$

$$ET(t_{q,i}) \overset{def}{=} ST(t_{q,i}) + \sum_{r \in R} ct(t_{q,i}, r) \cdot x_{t,r} \tag{6}$$

where $ct(t, r)$ represents the computation time of task t executed at RSU r and $delay(r, r')$ is the communication delay to transfer the computation result from RSU r to r'. These are given in advance.

Equation 5 describes that the task $t_{q,i}$ can only be executed after tasks preceding it $(t_{q,1}, ..., t_{q,i-1})$ are executed and the data computed by the preceding grids, such as travel time and shortest paths, has been received.

2) Accuracy and RSU Manhattan Distances: Given a set of queries Q_i, the most ideal assignment of tasks T_q to RSUs R is one which matches the generated grid assignments $SG(q)$ perfectly. However due to the amount of queries and computation capacity of RSU, tasks can be reassigned to other RSUs. We compute these differences in the expected and assigned grids as the following[3]: First, computation accuracy denoted by MA is defined as follows.

$$MA(g, g') \triangleq 1 - dist(g, g') \tag{7}$$

[3]As described before, data of each grid is replicated in other grids (RSUs) but the replication data updates will be less frequent in further grids.

Algorithm 1: Optimal Sequence Grid Generation

Input: Source $s \in V$, Destination $d \in V$, Time: τ
Output: Optimal sequence grid OG

```
Initialize SeqGrids list;
i ← 0;
SeqGrids[i] ← GetGrid(s);
g_final ← GetGrid(d);
while SeqGrids[i] ≠ g_final do
    currentGrid ← SeqGrids[i];
    SeqGrids[i + 1] ←
    GetNextGrid(currentGrid, g_final, τ);
    i ← i + 1;
end while
SeqGrids[i] ← g_final;
return SeqGrids
```

where $dist(g, g')$ is given by:

$$dist(g, g') \triangleq \frac{MD(g, g')}{MaxDist} \tag{8}$$

$MD(g, g')$ is the Manhattan Distance between the two different grid assignments. While *MaxDist* is the maximum distance a route needs to traverse to go from source to destination and is based on the service area covered by the application.

Eq. 7 shows the estimated model accuracy difference due to utilizing an RSU that is assigned to a grid different from the expected grid given by SG_q.

*3) **Utility Function**:* In our target environment, the user's primary desire is to receive the response to their query within a preferable delay time D_{th} given by Eq. 4 and a secondary desire of achieving high accuracy of computed path given by Eq. 7. Based on this we design the utility function $U(q)$. For every time window i, every task sequence T_q generated by q should be assigned to RSUs such that it meets the constraints given by Eqs. 1, 2 and 4, while maximizing the accuracy of the generated route. Then, we define $U(q)$ as follows:

$$U(q) = \sum_{j=1}^{k_q} \frac{MA(g_{q,j}, g(t_{q,j}))}{k_q} \tag{9}$$

where $g(t)$ is the grid of the RSU to which task t is assigned.

*4) **Objective Function**:* The purpose of the distributed task allocation is to find optimal assignments of tasks T to RSUs R, in order to satisfy the given constraints and maximize the model accuracy of the generated routes. We define the objective function as:

$$\textbf{Minimize}: \sum_{q \in Q_i} U(q) \text{ subject to } (2) - (4) \tag{10}$$

B. Task Assignment Algorithm

Given a network map which has been divided into grids (Fig. 1), and using Algorithm 1, we generate an optimal

Algorithm 2: Get Next Grid

Data: Equivalent Grid Routing model $\hat{E}$
Input: Current Grid: g_{curr}, Destination: g_{dest}, Time: τ
Output: Next Grid: g_{next}

```
1 begin
2     g_next ← Ê.predict(g_curr, g_dest, τ);
3     return g_next
```

Algorithm 3: Sequence Grid Task Assignment

Input: Set of queries: Q;
Output: Modified Grid: MG

```
1 begin
2     foreach q ∈ Q do
3         OG_q ← GenOptGSeq(q.s, q.d, q.τ);
4         if Delay(OG_q) > D_th then
5             OG_q ← ModGSeq(OG_q);
6         AssignTasks(OG_q);
```

Algorithm 4: Modified Sequence Grid Generation

Input: Sequence Grid: SG
Data: Neighbor Level: L;
Output: Modified Grid: MG

```
1 begin
2     MG ← SG;
3     while Delay(MG) > D_th and |SG| ≠ ∅ do
4         g ← a grid randomly selected in SG;
5         ns ← GetGridNeighbors(g, L);
6         bg ← GetLeastUtilized(ns);
7         MG ← modified MG by replacing g with bg
8         SG ← SG − {g};
9     return MG
```

sequence of grids based on the user's desired source s, destination d and time τ. Line 3 in Algo. 1 gets the equivalent grid from coordinate s. It builds the optimal sequence grid by appending the next best grid one at a time in Line 7. Algorithm 2 receives the current grid, d and τ from Algo. 1. It uses a predefined routing model $\hat{E}$, which is assumed to be trained on trips over N, to predict the next best grid to use given the inputs.

Algorithm 3 is the start point of the system, it generates OG in line 3 using Algo. 1. If the OG does not satisfy the delay constraint D_{th}, then it passes the OG_q to Algorithm 4 which uses the set of constraints for processing, query and delay in order to identify which RSU in each grid can handle the routing tasks while satisfying the delay constraint D_{th}.

Algo. 4 modifies the SG by selecting the best possible neighbor node for randomly selected $g \in SG$ and repeats this until the delay constraint D_{th} is met. We select the best grid bg based on the grid's RSU utilization from L level neighbors of the selected grid by using $GetLeastUtilized()$.

V. Experiments and Results

In this section, we evaluate our task allocation algorithm for decentralized architectures. We discuss the experimental setup and data used in our simulation. We evaluate different task allocation configurations used in our algorithm and finally we evaluate our approach and compare it to a centralized task allocation solution.

A. Experiment Setup

In this section we discuss the necessary setup for our experiment.

1) **Network Setup**: We used HERE API [31] data which includes speed data for each road segment within the Nashville Metropolitan area with a bounding box of $(-87.04999, 35.97, -86.510, 36.42)$. A network graph is generated from this data which consists of a total of 2623 nodes and 10880 edges. We partition this network graph into equidistant grids using geohashing with a precision of 5 shown in Fig.1. The grid area affects total number of sensors which in turn affects the processing time for each RSU. We may vary the area and number of grids in order to identify the limitations and capabilities of RSUs in maintaining a particular area or number of sensors. For the simulation, we partitioned the Metropolitan area into 613 grids each with an area of $600m^2$.
2) **RSUs**: RSUs used in this are simulated in a centralized configuration. Each grid above, is assigned to a corresponding RSU. RSUs are assumed to have infinite memory and accept and execute tasks generated by the task allocation algorithm. RSUs are assigned a grid's sub-graph which hold location and speed data for that particular grid.
3) **Speed Data and Accuracy**: Speed data utilizes data provided by the HERE API for the month of March 2018. We assume that as the $MD(g, g')$ increases, the model accuracy decreases by an equivalent factor.
4) **Trip Data**: To simulate routing queries, we synthetically generate up to 100 source and destination pairs at random from the Nashville Metropolitan Area. A time window was chosen randomly out of 24 hours. An optimal sequence grid is generated from each pair and we perform our task allocation algorithm based on this optimal sequence grid.

B. RSU constraints

In order to set the value of the constraints, we run 100 queries through different neighbor levels in order to identify the delay's cumulative distribution function (CDF) shown in Fig. 8. We then set different D_{th} as 200ms, 100ms and 60m for 0^{th}, 1^{st} and 2^{nd} neighbor levels respectively. These constraints limit the amount of tasks that can be assigned to a particular RSU.

C. Processing time and Delay calculations

These delay take into account both processing times and communication delay between grids. Processing times, described as $ct(t', r)$, are computed based on the number of nodes, V_g, in a sub-graph $N_g = (V_g, E_g)$ of each grid. The greater the number of V_g that longer the processing times for routing within grid g. Delay times, $delay(r, r')$, are given by $MD(g, g')$ between the optimal RSU in SG_q and actual RSU it was assigned to, as well as the distance from the actual RSU to the next RSU which contains the task's next step. The values for $ct(t', r)$ range from 0.026ms to 0.570ms per task (these values are based on actual measurement on Mac mini). While $delay(r, r')$ ranges from 0 to 3 hops per task that correspond to 0 to 3ms communication delay (we assume 1ms for 1 hop communication).

D. Grid Neighbor Levels

Neighbor levels vary how many other RSUs are taken into account during task allocation. For this use case we attempt up to 3 levels of neighbors. As the level of neighbors increase, the larger the search area for task allocation is, similar to [23]. Fig. 2 shows the number of neighbor nodes (RSUs) for the 0th and 1st levels. For the 0th level, only RSU (i, j) is selected, 1st increases this number to the surrounding 8 nodes, and a further 24 nodes for 2nd.

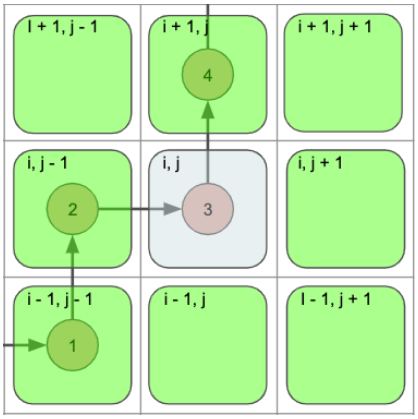

Fig. 2: Neighboring Nodes of Node (i, j). The sequence with arrows shows an example of an optimal grid sequence. The task for grid (i, j) can be assigned to RSUs in its neighbors like $(i, j+1), (i-1, j+1)$ if the neighbor level is 1.

E. Evaluation of Neighboring Nodes on Task Allocation

In order to quantify the task allocation algorithm we use 100 random trips consisting of $(source, destination, time)$ that we assume will be queried almost simultaneously. These 100 trips will then be allocated to RSUs. Allocation will depend on the neighbor levels. Trip data for all the neighbor level configurations remain consistent, as such only the task allocation varies between tests. For 0th level task allocation, we force the task to be assigned to the optimal grids SG_q without considering any processing constraints. For the other levels (1st and 2nd) we set constraint and distribute tasks to neighbor nodes if the optimal RSU for use is already over-utilized.

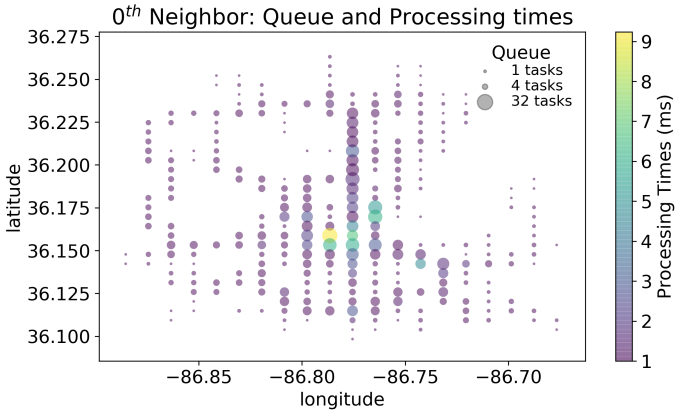

Fig. 3: 0th Neighbors Utilization

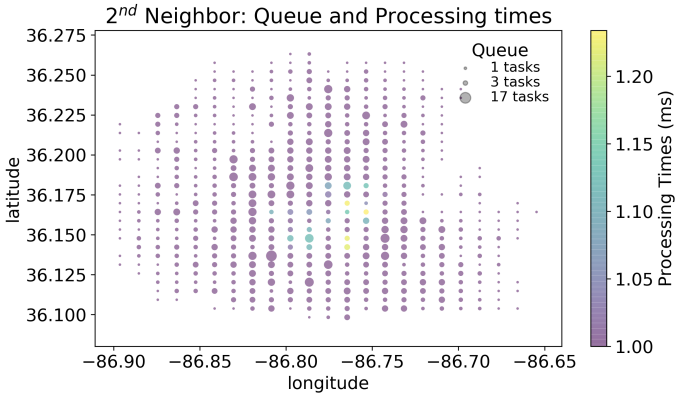

Fig. 4: 2nd Level Neighbors Utilization

Figure 3 shows the limited number of RSUs being used for task allocation. In addition, the RSUs with the heaviest load in the Downtown Nashville areas have much more utilization compared to the surrounding RSUs. Comparing it to Fig. 4, we can see that a larger number of RSUs across the map are being utilized for task allocation. The downtown grids that were over-utilized in Fig. 3 still see heavy usage, however the neighboring RSUs allow the system to meet D_{th} by accepting more tasks.

We can see in Fig. 5 that the addition of neighboring nodes for task allocation has a direct effect on the average processing time per RSU. As the number of neighbors increase RSU count increases and in turn average processing time decreases substantially.

However this decrease in processing time comes at a price. The delay between the optimal and actual RSUs grids increase as the neighbor level increases. Fig. 6 shows the decrease in average processing time of RSU, but at the same time showing the decrease in model accuracy. However, comparing 0th and 1st level neighbor cases, the 30% decrease in processing time

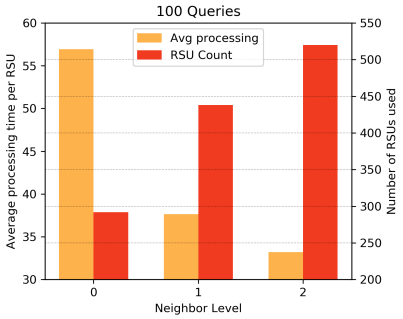

Fig. 5: Processing times and RSUs used vs Neighbor Level

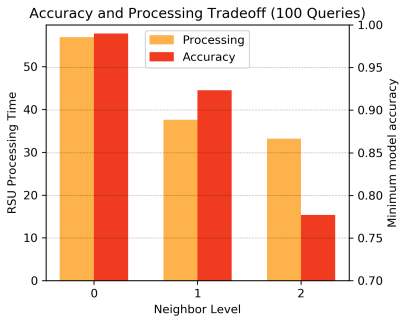

Fig. 6: Accuracy and Delay trade-off

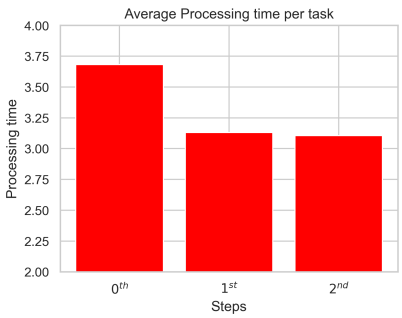

Fig. 7: Processing time per task step

outweighs the decrease in accuracy from 99 to 92.3%. One of the possible ways to limit the effect of this trade-off is to add weights to the neighbor selection algorithm since right now only the processing time constraint is being taken into account after randomly selecting a neighbor.

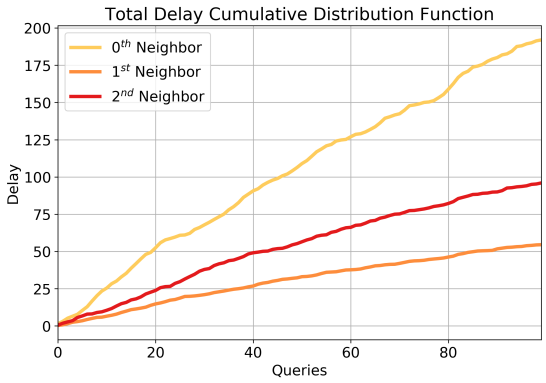

Fig. 8: Query Delay Cumulative Distribution Function. As D_{th}, 200ms, 100ms and 60ms are used for 0th, 1st and 2nd neighbor levels, respectively.

Finally breaking down every query into multiple tasks each consisting of separate route steps, we can get a better view of the effect of task allocation on meeting the delay constraints. As we execute tasks steps in parallel through the various RSUs, the increased number of RSUs allow for greater parallelization which results in tasks being executed faster when compared to a central approach. As shown in Fig. 7, 2nd level neighbor allocation, RSUs process each task 17% faster on average compared to 0th level allocation. This translates to large speed benefits the greater the number of Q.

For a centralized approach, we perform route planning for all queries in a sequential manner with only device and then monitor the processing time. While a centralized approach can produce lower delays and higher accuracy, it suffers from slower processing times especially for higher number of simultaneous queries as shown in Fig. 9.

Limitations of the current approach: Currently, the algorithm for task allocation, Algorithm 3, is centralized and must

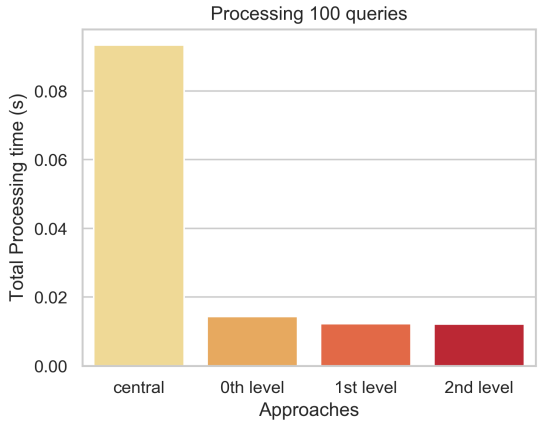

Fig. 9: Comparing with centralized approach

be run at one RSU after collecting all queries within each time slot i. However, the computation time of the algorithm is not big (93 ms for 100 queries) and the queries are transmitted in parallel to the computation RSU, so the proposed approach is still feasible. However, to treat more number of queries in wider target area with more grids, this approach will induce long query response time. In the future, we will develop the decentralized version of the task allocation algorithm to reduce the overall delay for computing the task allocation and collecting the queries even for such a case.

VI. Conclusion and Future Work

In this work we provided a task allocation algorithm for a resilient, route planning service for edge networks. We utilize decentralized road side units (RSU) as computing resources. However these RSUs are resources (memory and computational) constrained. We investigated three possible constraints on the network: processing time, communication delays and model accuracy. We simulated using a real world data set and found that our approach is able to provide a substantial increase in processing speed and decrease in delay compared to traditional centralized approaches. In addition, our approach can be scaled at configured to meet various constraints of the edge networks.

Experiments show that data driven services which are deployed over decentralized edge networks will benefit from a distributed task allocation algorithm. Our approach relies on generating and distributing routing tasks to multiple neighboring nodes in an attempt to balance the speed and accuracy of the response. The greater the search area, the less the processing delay but also the accuracy. When compared to central approaches, ours is able to respond with less latency. Potential extension of this work include actual implementation of the algorithm to existing edge devices and identify other constraints such as memory and computational capacity. Additionally this approach can be extended and applied to other middleware dependent services.

Acknowledgements: This work was supported in part by JSPS KAKENHI Grant Number 16H01721 & 19H05665 and R&D for Trustworthy Networking for Smart and Connected Communities, Commissioned Research of National Institute of Information and Communications Technology (NICT) and National Science Foundation through award number 1818901.

REFERENCES

[1] United Nations Department of Economic and Social Affairs, "68% of the world population projected to live in urban areas by 2050," 2018, [Online; accessed 22-July-2019]. [Online]. Available: {https://www.un.org/development/desa/en/news/population/2018-revision-of-world-urbanization-prospects.html}

[2] 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*. ACM, 2012, pp. 13–16.

[3] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[4] P. Sanders and D. Schultes, "Engineering highway hierarchies," in *European Symposium on Algorithms*. Springer, 2006, pp. 804–816.

[5] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: Faster and simpler hierarchical routing in road networks," in *International Workshop on Experimental and Efficient Algorithms*. Springer, 2008, pp. 319–333.

[6] A. V. Goldberg and C. Harrelson, "Computing the shortest path: A search meets graph theory," in *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2005, pp. 156–165.

[7] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[8] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[9] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint Optimization of Task Scheduling and Image Placement in Fog Computing Supported Software-Defined Embedded System," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3702–3712, 2016.

[10] M. Q. Tran, D. T. Nguyen, V. A. Le, D. H. Nguyen, and T. V. Pham, "Task Placement on Fog Computing Made Efficient for IoT Application Provision," *Wireless Communications and Mobile Computing*, vol. 2019, 2019.

[11] A. Dubey, S. Pradhan, D. C. Schmidt, S. Rusitschka, and M. Sturm, "The role of context and resilient middleware in next generation smart grids." in *M4IoT@ Middleware*, 2016, pp. 1–6.

[12] S. Pradhan, A. Dubey, S. Khare, S. Nannapaneni, A. Gokhale, S. Mahadevan, D. C. Schmidt, and M. Lehofer, "Chariot: Goal-driven orchestration middleware for resilient iot systems," *ACM Transactions on Cyber-Physical Systems*, vol. 2, no. 3, p. 16, 2018.

[13] S. M. Pradhan, A. Dubey, A. Gokhale, and M. Lehofer, "Chariot: A domain specific language for extensible cyber-physical systems," in *Proceedings of the workshop on domain-specific modeling*. ACM, 2015, pp. 9–16.

[14] A. Salman, I. Ahmad, and S. Al-Madani, "Particle swarm optimization for task assignment problem," *Microprocessors and Microsystems*, vol. 26, no. 8, pp. 363–371, 2002.

[15] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 497–511, 2012.

[16] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2011, pp. 1–12.

[17] D. Ardagna, M. Ciavotta, and M. Passacantando, "Generalized nash equilibria for the service provisioning problem in multi-cloud systems," *IEEE Transactions on Services Computing*, vol. 10, no. 3, pp. 381–395, 2015.

[18] C. Catlett, W. E. Allcock, P. Andrews, R. Aydt, R. Bair, N. Balac, B. Banister, T. Barker, M. Bartelt, P. Beckman *et al.*, "Teragrid: Analysis of organization, system architecture, and middleware enabling new types of applications," IOS press, Tech. Rep., 2008.

[19] D. Schafer, J. Edinger, J. M. Paluska, S. VanSyckel, and C. Becker, "Tasklets:" better than best-effort" computing," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2016, pp. 1–11.

[20] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, "Resource provisioning for iot services in the fog," in *2016 IEEE 9th international conference on service-oriented computing and applications (SOCA)*. IEEE, 2016, pp. 32–39.

[21] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards qos-aware fog service placement," in *2017 IEEE 1st international conference on Fog and Edge Computing (ICFEC)*. IEEE, 2017, pp. 89–96.

[22] 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)*. IEEE, 2017, pp. 47–54.

[23] Y. Nakamura, T. Mizumoto, H. Suwa, Y. Arakawa, H. Yamaguchi, and K. Yasumoto, "In-situ resource provisioning with adaptive scale-out for regional iot services," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, Oct 2018, pp. 203–213.

[24] J. P. Talusan, F. Tiausas, K. Yasumoto, M. Wilbur, G. Pettet, A. Dubey, and S. Bhattacharjee, "Smart transportation delay and resiliency testbed based on information flow of things middleware," in *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2019, pp. 13–18.

[25] R. Bellman, "On a routing problem," *Quarterly of applied mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[26] L. R. Ford Jr, "Network flow theory," Rand Corp Santa Monica Ca, Tech. Rep., 1956.

[27] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[28] G. Di Stefano, A. Petricola, and C. Zaroliagis, "On the implementation of parallel shortest path algorithms on a supercomputer," in *International Symposium on Parallel and Distributed Processing and Applications*. Springer, 2006, pp. 406–417.

[29] Y. Tang, Y. Zhang, and H. Chen, "A parallel shortest path algorithm based on graph-partitioning and iterative correcting," in *2008 10th IEEE International Conference on High Performance Computing and Communications*. IEEE, 2008, pp. 155–161.

[30] A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders, "A parallelization of dijkstra's shortest path algorithm," in *International Symposium on Mathematical Foundations of Computer Science*. Springer, 1998, pp. 722–731.

[31] HERE Technology, "Here." [Online]. Available: https://www.here.com