Design of Robust and Efficient Edge Server Placement and Server Scheduling Policies

Shizhen Zhao†*, Xiao Zhang†, Peirui Cao, Xinbing Wang
Shanghai Jiao Tong University
{*shizhenzhao, zhangxiao123, caopeirui, xwang8*}*@sjtu.edu.cn*

2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS) | 978-1-6654-1494-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/IWQOS52092.2021.9521290

***Abstract*—We study how to design edge server placement and server scheduling policies under workload uncertainty for 5G networks. We introduce a new metric called *resource pooling factor* to handle unexpected workload bursts. Maximizing this metric offers a strong enhancement on top of robust optimization against workload uncertainty. Using both real traces and synthetic traces, we show that the proposed server placement and server scheduling policies not only demonstrate better robustness against workload uncertainty than existing approaches, but also significantly reduce the cost of service providers. Specifically, in order to achieve close-to-zero workload rejection rate, the proposed server placement policy reduces the number of required edge servers by about 25% compared with the state-of-the-art approach; the proposed server scheduling policy reduces the energy consumption of edge servers by about 13% without causing much impact on the service quality.**

***Index Terms*—Edge Computing, Server Placement, Server Scheduling, Robust, Workload Uncertainty**

I. Introduction

We study the edge server placement problem for 5G networks in this paper. By moving storage, compute, control, etc., closer to the network edge, Edge Computing (EC) could offer higher bandwidth, lower latency and better security to users, and thus has become a key enabling technology for 5G. As 5G takes off, deploying edge computing servers also becomes a priority for service providers.

The key challenge of edge server placement in 5G networks comes from workload uncertainty. 5G adopts *small-cell deployment* to allow end users to communicate at high data rate using millimeter wave. However, as the cell size reduces, the number of users served by each base station decreases and thus the aggregated workload at each base station becomes highly variable. One possible solution to handle such workload uncertainty is to over-provision edge computing resources based on the peak workload of all base stations. However, this approach not only incurs high deployment and energy cost, but also leads to low average resource utilization.

Most server placement literature [1]–[13] does not account for workload uncertainty explicitly. In general, these server placement proposals take a predicted edge workload vector as input, and compute server placement solutions with optimal expenditure, optimal access delay, or minimal energy consumption. However, we cannot keep updating server placement based on the real-time workload. When real-time workload patterns deviate from predictions, the performance guarantee offered by these proposals become questionable. Some recent works [14]–[17] studied how to design edge server placement policies that are robust to server failures. However, workload uncertainty is inherently different from server failures, and thus may need completely new handling mechanisms.

Handling workload uncertainty is a challenging task. One may use stochastic optimization to handle workload uncertainty. However, this approach requires knowing the detailed distribution of the random workload beforehand, which can be extremely difficult to obtain. Further, this approach may also suffer from the curse of dimensionality as the edge workload is actually a high-dimensional random vector containing thousands of entries. Another approach to handle uncertainty is robust optimization. This approach formulates uncertainty using a set, and could offer strong performance guarantee as long as the uncertainty is bounded by this set. However, finding an appropriate set for robust optimization can be difficult in practice. If we find a set that only covers a majority, (e.g., 99%) of the workload patterns, then the robust optimization approach cannot offer any guarantee for the out-of-bound workload patterns. In contrast, if we find a set that covers all the potential workload patterns, this set can be extremely large because the workload uncertainty is heavy-tailed (see Table I in Section III-C), drastically weakening the performance guarantee of robust optimization. Further, in some situations, it may not even be feasible to find such an uncertainty set.

We propose **RO-RP**, to explicitly account for workload uncertainty in the edge server placement problem. RO-RP is built on top of robust optimization, with newly developed techniques to handle out-of-bound workload patterns. The detailed techniques are described below:

1) **Robust Optimization (RO):** Our trace analysis in Section III-B suggests that the edge workload exhibits different patterns during workdays and holidays. Using robust optimization, we can **optimize server placement based on multiple representative workload patterns**. As a result, we can offer a strong performance guarantee as long as future workload patterns are within the convex hull formed by these representative workload patterns.
2) **Resource Pool Optimization (RP):** However, robust optimization alone cannot offer good guarantee when the future workload patterns are outside of the above convex hull. To overcome this challenge, we introduce

* Corresponding author
† Equal contribution

a new concept called **resource pooling factor**. By maximizing this resource pooling factor, the impact of large workload bursts can be minimized.

In addition to a robust edge server placement solution, we also propose server scheduling to reduce the energy consumption of edge computing. Note that cumulative edge workload has strong diurnal patterns (see Section III-A). Thus, turning off some servers (or changing servers to power-saving mode) during idle hours could potentially save significant amount of energy cost. However, toggling servers between on/off states may incur additional cost. We have explicitly accounted for the switching cost in our server scheduling formulation. The formulation and its evaluation are available in [18].

Finally, we evaluate our server placement policy based on both real traces from Shanghai Telecom and synthetic traces. Compared to the existing server placement policies, RO-RP significantly reduces the workload rejection rate given the same number of edge servers. To achieve close-to-zero workload rejection rate, RO-RP requires 25% fewer edge servers when compared with the state-of-the-art approach. We also evaluate our server scheduling policy in [18]. Compared to the strategy that turns on all servers at all times, server scheduling could reduce energy consumption by 13%.

II. Related Work

Prior work has studied how to deploy edge servers based on workload distribution. However, the workload distribution may not be accurate. Workload uncertainty may have a big impact on the eventual network performance, but unfortunately has not yet received much attention in the existing literature.

Many server placement proposals [1]–[9] have assumed that each base station can be only associated with one edge cloud in their formulations. With this assumption, many standard algorithms, e.g., k-means clustering algorithm, set cover algorithm, etc., can be used to design heuristic solutions for the server placement problem. However, this formulation is inherently non-robust to demand uncertainty. Whenever the workload of a base station bursts, the edge cloud that serves this base station has to bear the burden by itself. In fact, for a given server placement, EC users do benefit from offloading their job requests to multiple edge clouds [19], [20]. There does exist literature [10]–[13] that allows serving workload from the same base station in different edge clouds. However, workload uncertainty is not considered therein.

To improve the robustness of edge server placement, [14]–[17] studied how to account for server failures in their server placement formulations. However, workload uncertainty is inherently different from server failures, and may happen much more frequently in real time.

One natural idea to deal with workload uncertainty is to use robust optimization. This idea has been used to study the service scheduling problem [21] in EC and the replica server placement problem [22] in CDN. However, robust optimization cannot offer any guarantee when the workload is outside of the predicted set. Note that, unlike service scheduling, server placement results cannot be adjusted based on the real-time workload.

In addition to the server installation cost, energy also accounts for a big portion of the cost for edge computing. The energy-optimization literature on edge computing mostly focuses on the edge/IoT devices [23]–[25], but not on the edge servers. In this paper, we study how to perform server on-off scheduling for edge computing to save energy cost. As far as we know, this server scheduling problem was only studied in data centers [26]–[29], but has never been explored in edge computing. The biggest difference is that we need to account for the collaboration among different edge clouds when we study server scheduling in edge computing.

III. Workload Analysis

The appropriate design of server placement policies requires deep understanding of potential workload patterns of EC. However, EC has not seen widespread deployment, and thus there may not be any real data for its workloads. Instead, we perform workload analysis based on Shanghai Telecom dataset [1], [9], [30], [31], which includes the records collecting about 3042 base stations and 6236620 user requests. We believe the workload patterns of these communication records can provide good estimate for the workload patterns of EC.

A. Observation 1: The cumulative workload has strong diurnal patterns

We study how the cumulative workload across all base stations in Shanghai varies at different times of a day using a consecutive of 30 days of data. As shown in Fig. 1(a), the total number of requests are the lowest from midnight to about 6:00am every day. As people get up around 6:00-8:00am, the requests rise and peak at around 6:30-7:30am. Then, the requests are relatively consistent until 20:00pm, after which the requests gradually decrease. Another observation from the curves is that the number of requests on workdays (solid lines) is generally higher than that on holidays (dashed lines). This implies that one should not use workday's workload patterns to predict the workload patterns on holidays, and vice versa.

The strong diurnal patterns of edge workload motivates us to **perform server on-off scheduling** for EC. The objective is to reduce energy cost, without impacting on service quality.

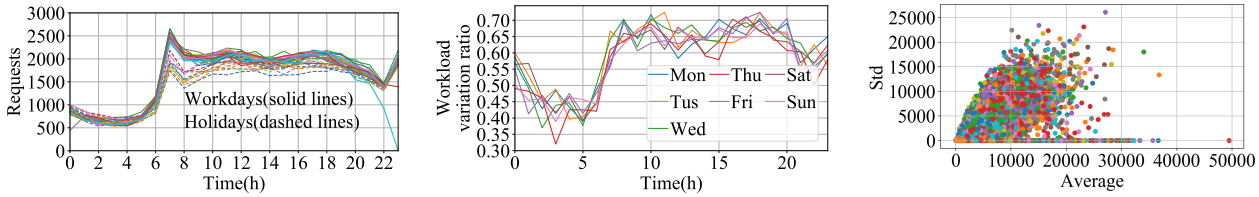
B. Observation 2: The spatial workload patterns of different days can be highly skewed and dramatically different

As shown in Fig. 2, we randomly pick a holiday and a workday to study their workload patterns at the same time of a day. Fig. 1(a) demonstrates the total workloads at workdays and holidays are approximately the same for most time. However, comparing the spatial workload distributions in Fig. 2, we find the workload patterns are apparently different. This observation suggests we should **use multiple different workload patterns to compute server placement solutions**.

Another observation from Fig. 2 is that the workload patterns are highly skewed, with much higher workload in the central area of Shanghai. Hence, traffic agnostic server

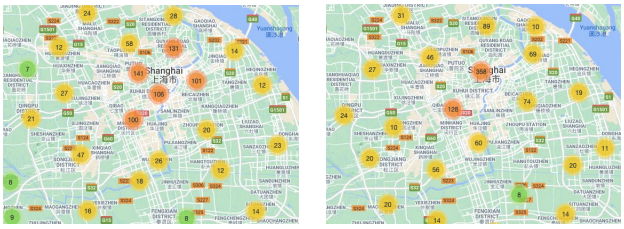
TABLE I
THE STATISTICS OF WORKLOAD AND INTER-ARRIVAL TIME OVER A TIME SPAN OF SIX MONTHS.

	Max	Average	std	99.999%	99.99%	99.9%	99%	90%	80%	70%	60%	50%	40%
Workload	40877	2314	3283	27144	10804	10800	10796	8857	3910	2097	1216	728	499
Inter-arrival time	13953857	7174	64180	7246866	2584387	584956	82414	10731	5001	3002	1990	1374	899



(a) Number of Requests at different time of the day during one month.
(b) Hour-level workload variation for two consecutive weeks.
(c) Hourly workload statistics for each base station over a period of 6 months.

Fig. 1. Data set analysis.



(a) 12:00-14:00 on Holiday
(b) 12:00-14:00 on Workday

Fig. 2. Requests distribution at different days.

placement strategies may not perform well, which is verified via simulation in Section VII-B.

C. Observation 3: The edge workload is highly bursty

To understand the burstiness of edge workload, we analyze workload sizes and inter-arrival times of all requests at each base station over a time span of six months. Table I summarizes the average, standard deviation and percentile values of these two metrics. Note that the inter-arrival time is heavy tailed, meaning that it is possible for a base station to receive a large request after being idle for a long period of time.

Since edge workload is highly bursty, accurately predicting edge workload can be very difficult. For example, we can use historical workday/holiday patterns to predict future workload. For the t-th hour every day, we could compute the following workload variation ratio: $V_t = \frac{\sum_{m=1}^{M} |w_m(t) - w'_m(t)|}{\sum_{m=1}^{M} w'_m(t)}$, where $w_m(t)$ is the total workload of the m-th base station in the t-th hour of the day, and $w'_m(t)$ is the total workload of the m-th base station in the t-th hour seven days ago. From Fig. 1(b), the workload variation ratio can be as large as 70%. Hence, using historical patterns to predict future workload can be inaccurate.

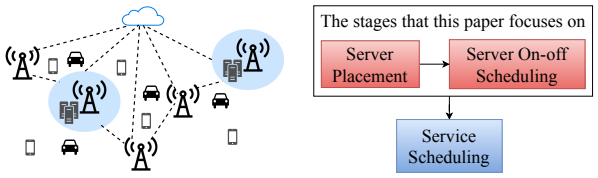
We are also interested in the relationship between the average workload and the workload uncertainty. For every base station and every hour in a week, we collect a sequence of workload values over 6 months with one value per week. We then compute the average value and the standard deviation for every workload sequence, and plot them in Fig. 1(c). From this figure, we can see that the workload's worst-case standard deviation grows approximately linearly with respect to its average value. However, the range of the standard deviation values is pretty large, and thus there may not be a rule of thumb to accurately predict the workload uncertainty.

The above analysis demonstrates that edge workload is highly variable. Thus, **how to handle workload uncertainty** becomes a primary focus in this paper.

IV. MODEL

We study edge server placement from a service provider's aspect. A service provider (SP) is responsible for managing the base stations, the central cloud and the edge servers (see Fig. 3(a)), with an objective to provide ubiquitous communication and computation services to its users. Users access SP's network through base stations, usually via wireless links. The base stations and the central cloud are typically interconnected through wired links. As we go into the 5G era, when millions of devices connect to the network, and data from each device floods in, edge computing also becomes critical to provide low latency, high reliability, and immense bandwidth.

In general, edge computing consists of two stages: server placement and service scheduling. For server placement, the SP needs to determine where to deploy servers and how many servers to deploy. Typically, edge servers are co-located with base stations. Server placement usually happens at the planning stage. For service scheduling, the SP is responsible for routing users' edge computing requests to nearby edge servers in real time. In this paper, we focus on server placement. Note that running edge servers may incur high energy cost. To reduce energy consumption, we introduce an additional stage, i.e., server on-off scheduling, to make an on-off schedule for edge servers based on predicted workloads. The overall workflow of edge computing is depicted in Fig. 3(b).



(a) Edge computing network.
(b) Overall workflow.

Fig. 3. Overall Architecture.

A. Mathematical Models

Let $\{B_1, B_2, ..., B_M\}$ denote the set of base stations (BSs) in an 5G network, where M is the total number of base stations. Assume time is slotted. $w_m(t)$ denotes the total edge computing workload of BS B_m at slot t. The workloads at different base stations form a workload vector $w(t) = [w_m(t), m = 1, 2, ..., M]$. Since the edge workload $w_m(t)$ has stringent latency requirements, such workload can be only processed by the edge servers deployed at BS B_m or nearby BSs of B_m. In this paper, we use Ω_m to denote the set of BSs whose edge servers can serve the workload of the BS B_m.

Remark on Ω_m: One can define Ω_m using different approaches, including latency requirement, hop-count requirement, distance requirement, etc. The methodology in this paper works for all the possible definitions of Ω_m.

1) Server Placement: Server placement typically happens at the planning stage. Once the edge servers are deployed, it may not be easy to change the server deployment. Hence, server placement must be robust against the potential workload variations. The easiest approach to deal with workload variations is to over-provision edge servers. However, this approach increases both equipment cost and energy cost. In this paper, given a historical trace of workload vectors, we study how to assign a total number of K servers to each BS, such that

$$S_1 + S_2 + \cdots + S_M = K, \text{ where } S_m\text{'s are integers}, \quad (1)$$

where S_m is the number of edge servers deployed at BS B_m.

2) Service Scheduling: Service scheduling happens at the real time stage. At time t, for the workload $w_m(t)$ at the BS B_m, the objective of service scheduling is to determine the fraction $u_{mn}(t)$ of the workload $w_m(t)$ that is assigned to the base station $B_n \in \Omega_m$. Clearly,

$$\begin{cases} \sum_{B_n \in \Omega_m} u_{mn}(t) = 1, \text{ for } m = 1, 2, ..., M, \\ 0 \le u_{mn}(t) \le 1 \text{ and } u_{mn}(t) = 0, \text{ for } B_n \notin \Omega_m. \end{cases} \quad (2)$$

3) Server On-off Scheduling: Server on-off scheduling happens in-between server placement and service scheduling. Fig. 1(a) suggests that the cumulative edge workload has daily peaks and troughs. Server placement needs to account for the daily peaks. However, keeping all the servers always *on* incurs significant energy cost, especially during idle hours. Let $S(t) = [S_m(t), m = 1, 2, ..., M]$, where $S_m(t)$ is the number of active servers at the BS B_m at time t. The objective of server on-off scheduling is to reduce energy cost, without impacting on users' service quality. Clearly,

$$S_m(t) \le S_m, \forall m = 1, 2, ..., M \text{ and } t. \quad (3)$$

B. Performance Metrics

While we design server placement and server on-off scheduling strategies, we are interested in optimizing the following three performance metrics.

1) Rejected Workload: When edge servers run out of resources to serve some portion of edge workload, such edge workload is considered as rejected. (Offloading this workload to the central cloud may violate latency requirement.) This could happen when the real-time workload $w(t) = [w_1(t), w_2(t), ..., w_M(t)]$ bursts at some BSs. Given the numbers of active servers $S(t)$ and the workload vector $w(t)$, the total amount of rejected workload can be computed by

$$\boxed{\begin{array}{ll} \min\limits_{u_{mn}(t)} & \sum\limits_{n=1}^{M} \max\left\{0, \sum\limits_{m=1}^{M} w_m(t) u_{mn}(t) - C S_n(t)\right\} \\ \text{s.t.} & u_{mn}(t) \text{ satisfy (2)}, \end{array}} \quad (4)$$

where C is the capacity of one server. Note that $\max\left\{0, \sum_{m=1}^{M} w_m(t) u_{mn}(t) - C S_n(t)\right\}$ is the total rejected workload at the BS B_n. Solving (4) gives the minimum possible amount of workload to be rejected.

2) Number of Servers Required: From service providers' aspect, rejecting EC requests is highly undesirable, because they may lose customer loyalty. Then, another important metrics arises, i.e., what is the minimum number of servers required in order to guarantee zero workload rejection rate? Certainly, this number depends on the server placement strategy and the workload patterns. In Section VII-B, we will use extensive simulation to obtain an estimate of this metric for different server placement strategies.

3) Cost: The cost of edge servers consists of two parts:

- Server running cost $E_r + E_w \cdot x$, where E_r is the energy cost of running an idle server for one time slot, and E_w is the energy cost per workload unit. Note that E_r may account for over 50% of the energy cost in a server [32].
- Switching cost E_s, which models the cost of toggling a server between on/off states[1].

Note that the total energy cost of all the workload equals $E_w \sum_{m=1}^{M} \sum_t w_m(t)$, which is out of our control. Hence, in this paper, we mainly focus on the idle-server cost and the switching cost, which in total can be calculated as

$$W = \sum_{m=1}^{M} \sum_t \left(E_r \times S_m(t) + E_s \times (S_m(t) - S_m(t-1))^+ \right) \quad (5)$$

where $x^+ = \max\{0, x\}$. Clearly, if we choose not to perform server on-off scheduling, then the switching cost becomes zero, but the running cost increases.

V. Server Placement

The primary objective of server placement is to reduce the workload rejection rate. Since this metric is closely related to service scheduling, our server placement strategy will account for the effect of service scheduling.

A. A robust joint optimization approach

Considering edge workload exhibits different patterns at different time, our first idea is to use robust optimization to compute a server placement solution. Specifically, given a sequence of historical workload vectors, we pick multiple representative workload vectors using the following steps:

[1]As stated in [26], if only energy cost matters, then E_s is about the cost of running a server for a few seconds to several minutes; if increased wear-and-tear is accounted for, then E_s becomes on the order of the cost of running a server for a hour. We use the latter to measure the switching cost E_s.

Step 1 Divide historical workload vectors into L groups such that the workload vectors in the same group are all 1) from workdays or holidays, and 2) from the same time period (e.g., 8:00-11:59am) of a day.

Step 2 Compute an average workload vector $w^l = [w^l_m, m = 1, 2, ..., M]$ for the l-th group of workload vectors[2]. Note that the following analysis also works for other choices of workload vectors.

In total, we obtain L workload vectors.

For each workload vector w^l, we introduce service scheduling variables $u^l = [u^l_{mn}, m, n = 1, 2, ..., M]$ satisfying (2). Then, we jointly optimize the server placement variables $S = [S_m, m = 1, 2, ..., M]$ satisfying (1) and the service scheduling variables $[u^1, u^2, ..., u^L]$.

For each workload vector w^l and the corresponding service scheduling variables u^l, it is easy to obtain the total workload allocated to the BS B_n, i.e., $\sum_{m=1}^{M} w^l_m u^l_{mn}$. To reduce the likelihood of workload rejection, we impose the following constraint that restricts the edge server utilization to be less than β for any BS B_n and any workload vector w^l:

$$\sum_{m=1}^{M} w^l_m u^l_{mn} \leq S_n C \beta, \text{ for any } B_n \text{ and } w^l. \tag{6}$$

Then, the overall formulation is given as

$$\begin{aligned} \min_{S,u^l,\beta} \quad & \beta \\ \text{s.t.} \quad & S \text{ satisfy (1)}, \\ & S, u^l, \beta \text{ satisfy (2)(6) for } l = 1, 2, ..., L. \end{aligned} \tag{7}$$

We denote the optimal value of (7) by β^*.

Understanding the drawback of (7): When future workload vectors are bounded by the convex hull of $[w^1, w^2, ..., w^L]$, the above robust optimization based formulation (7) can offer strong performance guarantee (see [18] for more detailed analysis). However, some future workload vectors can be out of bound, owing to the fact that edge workload is highly bursty. Certainly, one can increase the convex hull by scaling up the representative workload vectors $[w^1, w^2, ..., w^L]$. Unfortunately, β^* will also increase, making the performance guarantee of robust optimization weaker. Further, if $\beta^* > 1$, then the performance guarantee becomes useless.

B. Handling Out-of-bound Workload with Resource Pooling

Intuition: We first introduce the concept of *resource pool*. For the workload at the BS B_m, its *resource pool* is defined as the total amount of server resources that can serve the workload, which equals to $\sum_{B_n \in \Omega_m} S_n$. To understand how resource pooling helps mitigate the impact of out-of-bound workload, we consider the motivation example in Fig. 4. The predicted MEC workloads at the base stations A and B are both 10, while the real time workloads turn out to be 8 and 12. If we deploy 10 units of computing resources at A and B each (see Fig. 4(a)), then in real time, 2 units of B's workload will be rejected owing to the fact that 1) B's MEC servers are already fully utilized, and thus cannot serve B's burst; 2) A is too far from B, and thus cannot serve B's burst either. On the other hand, if we deploy computing resources at the base stations C and D (see Fig. 4(b)), the computing resources of C and D can serve the workloads from both A and B, because C and D are both within an acceptable distance from A and B. As a result, the resource pools of both A and B increases from 10 to 20. Then, as B's workload bursts from 10 to 12, we can offload 2 units of workload from the edge cloud D to the edge cloud C. With an increase resource pool, the workload from the base station B will not be rejected any more.

(a) Small resource pool. (b) Large resource pool.

Fig. 4. Motivation example of resource pooling.

Formulation: Motivated by the above example, we introduce a resource pooling factor η, such that

$$\eta \max_{l=1}^{L} \{w^l_m\} \leq \sum_{B_n \in \Omega_m} S_n. \tag{8}$$

Then, we compute the server placement result by maximizing the resource pooling factor η:

$$\begin{aligned} &\textbf{Maximize resource pooling factor:} \\ \max_{S,u^l,\eta} \quad & \eta \\ \text{s.t.} \quad & S, \eta \text{ satisfy (1)(8)}, \\ & S, u^l, \beta^* \text{ satisfy (2)(6) for } l = 1, 2, ..., L. \end{aligned} \tag{9}$$

Note that our server placement policy involves two steps: 1) compute β^* using the *robust optimization* formulation (7); 2) optimize *resource pooling* based on (9). Hence, we will also use **RO-RP** to represent our server placement policy. The resource pooling step is critical in handling the out-of-bound workload. The detailed analysis is available in [18].

C. Reduce Algorithmic Complexity for Server Placement

We compute server placement results by solving (7) and (9). However, both (7) and (9) are integer programming problems, which are computationally expensive. In fact, we have tried solving (9) directly based on the Shanghai Telecom dataset with 3042 base stations, but unfortunately the state-of-the-art integer programming solver, Gurobi [33], cannot finish with a solution even after running a few hours.

To reduce computational complexity, we adopt a *relaxing and rounding* approach. Specifically, we first allow the server placement variables S to take fractional values. Then, both

[2]We have also tried using the component-wise max workload vector to compute a server placement solution. However, our simulation results in Section VII-C1 cannot give any conclusive answer that one option is better than another. Hence, one may use a different approach to compute representative workload vectors.

(7)[3] and (9) can be converted to linear programming problems, which can be solved in polynomial time. After obtaining a fractional server placement solution of S, we can then round the fractional solution to an integer solution. Note that we need to ensure that (1) is satisfied after rounding.

Let $\tilde{S}^* = [\tilde{S}_1^*, \tilde{S}_2^*, ..., \tilde{S}_M^*]$ be a fractional server placement solution. We have tried 5 rounding schemes based on different intuitions, and finally adopt the Smallest Resource Pool First approach, i.e., the base stations with smaller resource pools are given higher priority to round up its fractional solution. More analysis and evaluation results are available in [18].

VI. Server Scheduling

This section and its evaluations are available in [18].

VII. Evaluation

A. Introduce the Trace for Evaluation

1) Shanghai Telecom's Real Communication Records: We use this trace to compare our server placement and scheduling solution against previous approaches. (Details in Section III)

2) Synthetic Trace: It was shown in Section III-C that edge workload is hard to predict accurately. Hence, our solution must be robust against potential workload bursts. Unfortunately, the historical real traces may not offer a comprehensive coverage over the burst patterns. Hence, we construct synthetic traces to evaluate solution robustness. The construction of our synthetic traces is provided in [18].

B. Compare Different Server Placement Policies

We group existing server placement policies into three categories and evaluate them using the real trace.

Traffic-agnostic policies: This policy does not use any workload information for server placement. We evaluate the following policies in this category:

1) **Random**: Place K servers at randomly chosen BSs.
2) **Clustering**: Use k-means algorithm to group BSs into k clusters. Let $s_i^{(C)}$ be the number of BSs in the i-th cluster. Place $K\frac{s_i^{(C)}}{\sum_{i=1}^{k} s_i^{(C)}}$ servers at the centroid of the i-th cluster.
3) **Uniform**: Divide the geographical area into fixed-sized zones. Let $s_i^{(U)}$ be the number of BSs in the i-th zone. Place $K\frac{s_i^{(U)}}{\sum_{i=1}^{k} s_i^{(U)}}$ servers at the centroid of the i-th zone.

Traffic-aware but uncertainty-agnostic policies: The two policies compute server placement based on a predicted workload vector, but do not account for the workload uncertainty.

1) **Traffic-aware without load balancing (TwithoutLB)**: In this policy, the workload at each BS can be only allocated to the nearest BS with edge servers. Many existing server placement policies fall into this category [1]–[9]. In this paper, we use k-means algorithm to group BSs into k clusters, and assign all the workload in each cluster to the centroid of this cluster. Let l_i be the total workload of the BSs in the i-th cluster. Place $K\frac{l_i}{\sum_{i=1}^{k} l_i}$ servers at the centroid of the i-th cluster.
2) **Traffic-aware with load balancing (TwithLB)**: In this policy, the workload at each BS can be load balanced to the close-by BSs. This setting was also adopted in [10]–[13]. Here, we compute a server placement solution by solving (7) with only one workload vector. (We use historical average workload vector in the evaluation.)

Traffic and uncertainty-aware policies: This is our server placement strategy proposed in Section V (namely RO-RP).

We use two weeks of real traces to evaluate the workload rejection rate for different server placement policies. In Fig. 5, we fix the total number of servers as 8000. Apparently, our approach achieves the lowest workload rejection rate. In Fig. 5(c), we vary the number of servers from 7000 to 12000, and study how many servers are required in order to achieve close-to-zero workload rejection rate. Under our policy, approximately 7500 servers are required. In contrast, the second best option requires about 10000 servers.

C. Understand Different Design Choices of RO-RP

1) Average workload vector vs. component-wise max workload vector: In Section V-A, we use the average workload vector as the representative workload vectors. Another option is to use the component-wise max workload vector. There may not be a conclusive answer that one is better than another. We compare these two options using different synthetic traces. As shown in Fig. 6(a), using average workload vector yields lower workload rejection rate for one trace, but leads to higher rejection rate for the other one.

2) Understanding the effect of robust optimization and resource pooling: Our server placement policy utilizes both robust optimization and resource pooling to improve its robustness against workload uncertainty. To understand the contribution of each technique, we evaluate four options below:

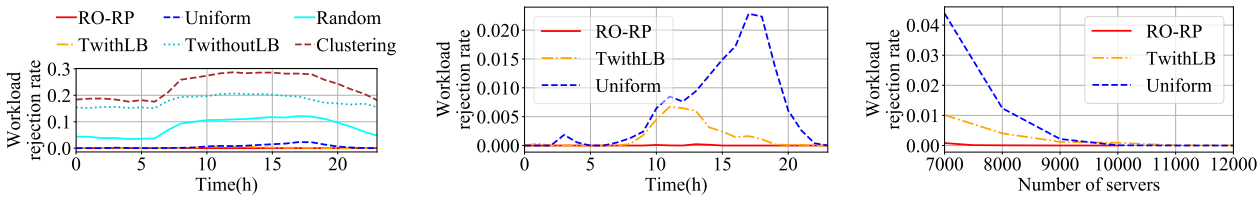
1) **Robust optimization+resource pooling (RO-RP):** Our server placement strategy proposed in Section V.
2) **Robust optimization only (RO-only):** Only solve (7) for a server placement solution.
3) **Resource pooling only (RP-only):** Use only one workload vector (e.g., historical average) to solve (7) & (9).
4) **Not handling workload uncertainty (TwithLB):** Use only one representative workload vector to solve (7).

From Fig. 6(b), we can see that both techniques help reduce the workload rejection rate, and the performance is the best when both techniques are enabled.

VIII. Conclusion

In this paper, we propose a new methodology to design server placement and server scheduling policies that are robust to workload uncertainty. This methodology utilizes robust optimization to provide guarantee for workloads that are within a predetermined uncertainty set, and performs resource pool optimization to improve service quality for out-of-bound workloads. Simulation results demonstrate the effectiveness of

[3]The constraints (6) contains a multiplicity term $S_n\beta$. To convert (7) into a linear programming problem, we need to substitute S_n by $\hat{S}_n = S_n\beta$ in (6), and replace (1) by $\sum_{m=1}^{M} \hat{S}_n = K\beta$ in (7).

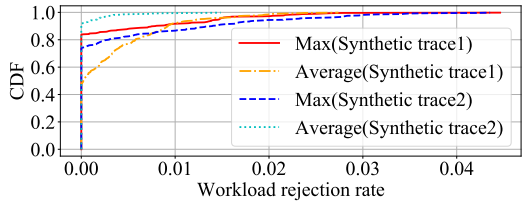


(a) Workload rejection rate for six server placement policies.

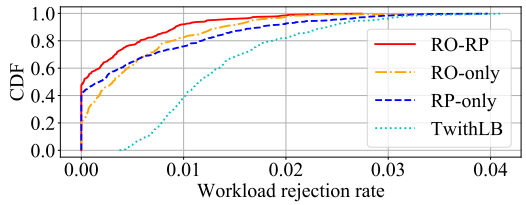
(b) A zoom in of the performance for the top-3 best server placement policies.

(c) Number of servers required for the top-3 best server placement policies.

Fig. 5. Comparison of different server placement policies.



(a) Average workload vs. Component-wise max workload.



(b) Contribution of robust optimization and resource pooling.

Fig. 6. Different design choices of our server placement policy.

this methodology. From a service provider's aspect, the resulting server placement policy reduces the number of required edge servers by about 25% compared with the state-of-the-art approach and the resulting server scheduling policy reduces the energy consumption by about 13%.

Acknowledgement: This work was supported by the NSF China (No. 61902246).

REFERENCES

[1] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C.-H. Hsu, "Edge server placement in mobile edge computing," *JPDC*, 2019.

[2] Y. Guo, S. Wang, A. Zhou, J. Xu, J. Yuan, and C.-H. Hsu, "User allocation-aware edge cloud placement in mobile edge computing," *Software: Practice and Experience*, 2019.

[3] F. Zeng, Y. Ren, X. Deng, and W. Li, "Cost-effective edge server placement in wireless metropolitan area networks," *Sensors (Basel)*, 2018.

[4] Y. Li and S. Wang, "An energy-aware edge server placement algorithm in mobile edge computing," in *IEEE EDGE*, 2018.

[5] T.-L. Chin, Y.-S. Chen, and K.-Y. Lyu, "Queuing model based edge placement for work offloading in mobile cloud networks," *IEEE Access*, 2020.

[6] K. Cao, L. Li, Y. Cui, T. Wei, and S. Hu, "Exploring placement of heterogeneous edge servers for response time minimization in mobile edge-cloud computing," *IEEE Trans Industr Inform*, 2020.

[7] S. K. Kasi, M. K. Kasi, K. Ali, M. Raza, H. Afzal, A. Lasebae, B. Naeem, S. Islam, and J. J. P. C. Rodrigues, "Heuristic edge server placement in industrial internet of things and cellular networks," *IoT-J*, 2020.

[8] Y. Ren, F. Zeng, W. Li, and L. Meng, "A low-cost edge server placement strategy in wireless metropolitan area networks," in *ICCCN*, 2018.

[9] Y. Guo, S. Wang, A. Zhou, J. Xu, J. Yuan, and C.-H. Hsu, "User allocation-aware edge cloud placement in mobile edge computing," *Software: Practice and Experience*, 2020.

[10] T. Lähderanta, T. Leppänen, L. Ruha, L. Lovén, E. Harjula, M. Ylianttila, J. Riekki, and M. J. Sillanpää, "Edge computing server placement with capacitated location allocation," Available online: https://arxiv.org/pdf/1907.07349.pdf, 2020.

[11] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Efficient algorithms for capacitated cloudlet placements," *TPDS*, 2016.

[12] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *TCC*, 2017.

[13] S. Yang, F. Li, M. Shen, X. Chen, X. Fu, and Y. Wang, "Cloudlet placement and task allocation in mobile edge computing," *IoT-J*, 2019.

[14] G. Cui, Q. He, X. Xia, F. Chen, H. Jin, and Y. Yang, "Robustness-oriented k edge server placement," in *CCGRID*, 2020.

[15] G. Cui, Q. He, F. Chen, H. Jin, and Y. Yang, "Trading off between user coverage and network robustness for edge server placement," *TCC*, 2020.

[16] D. Lu, Y. Qu, F. Wu, H. Dai, C. Dong, and G. Chen, "Robust server placement for edge computing," *IPDPS*, 2020.

[17] S. U. Khan, A. A. Maciejewski, and H. J. Siegel, "Robust cdn replica placement techniques," *IPDPS*, 2009.

[18] S. Zhao, X. Zhang, P. Cao, and X. Wang, "Design of robust and efficient edge server placement and server scheduling policies: Extended version," Available online: https://arxiv.org/pdf/2104.14256.pdf, 2021.

[19] V. Farhadi, F. Mehmeti, T. He, T. La Porta, H. Khamfroush, S. Wang, and K. S. Chan, "Service placement and request scheduling for data-intensive applications in edge clouds," in *INFOCOM*, 2019.

[20] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, 2015.

[21] D. T. Nguyen, H. T. Nguyen, N. Trieu, and V. K. Bhargava, "Two-stage robust edge service placement and sizing under demand uncertainty," Available online: https://arxiv.org/pdf/2004.13218v1.pdf, 2020.

[22] K.-H. Ho, S. Georgoulas, M. Amin, and G. Pavlou, "Managing traffic demand uncertainty in replica server placement with robust optimization," in *Networking*, 2006.

[23] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *ISIT*, 2016.

[24] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *ComSoc*, 2017.

[25] J. Lim, J. Seo, and Y. Baek, "Camthings: Iot camera with energy-efficient communication by edge computing based on deep learning," in *ITNAC*, 2018.

[26] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," in *Infocom*, 2011.

[27] I. Goiri, F. Julia, R. Nou, J. L. Berral, J. Guitart, and J. Torres, "Energy-aware scheduling in virtualized datacenters," in *IEEE CLUSTER*, 2010.

[28] T. V. T. Duy, Y. Sato, and Y. Inoguchi, "A prediction-based green scheduler for datacenters in clouds," *IEICE Trans Inf Syst*, 2011.

[29] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *ComSoc*, 2015.

[30] S. Wang, Y. Zhao, L. Huang, J. Xu, and C.-H. Hsu, "Qos prediction for service recommendations in mobile edge computing," *JPDC*, 2019.

[31] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. S. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *TMC*, 2019.

[32] Intel, "The problem of power consumption in servers," https://www.infoq.com/articles/power-consumption-servers/.

[33] "Gurobi - the fastest solver," https://www.gurobi.com/.